9-16-2021

Computer Science Principles with C++

Seth D. Bergmann
Rowan University

DOI: 10.31986/issn.2689-0690_rdw.oer.1025

Computer Science Principles

C++ edition

Seth D. Bergmann

September 15, 2021

Preface

This book is intended to be used for a first course in computer science. It includes some programming, but the emphasis is on the principles which form the foundation for hardware and software. No prior experience with programming should be necessary in order to use this book.

This book is intended to be used for the College Board's Advanced Placement course known as AP Computer Science Principles (CSP). This course is not to be confused with a more traditional course known as AP Computer Science A, which places a strong emphasis on programming, currently using Java. The author(s) of this book have included all the topics that are in CSP.[1] We feel that there are some related topics which may be of interest to those who are new to computer science; in many cases we have included these extra topics, but they are denoted by a $\otimes$ symbol to indicate they are not part of the official CSP course, and students taking the AP CSP exam will not be responsible for these topics. Since the AP exams are given in early May, and most school years end in mid-June, many may wish to visit these extra topics after taking the AP exam.

The author(s) believe that a breadth-first approach is the best way to introduce the concepts of Computer Science to students. Rather than isolate topics in courses (bits and bytes in a computer organization course; formal grammars and languages in a theory course; lists, sets, and maps in a data structurs course; etc) we believe that topics should be introduced in a brief and simple manner at the starting level. Elaboration on these topics should occur in subsequent courses. This breadth-first approach allows the student to build on existing knowledge and retain a greater proportion of the material.

Our colleagues in the physical sciences have done this for over a century: Physics I, Physics II, Physics III; Chemistry I, Chemistry II, Chemistry III.

Some examples of this breadth-first approach to Computer Science:

- We teach the rudiments of binary numbers. Is this necessary to learn to program the solution to a simple problem? Probably not, but it will become necessary at some later time, when studying hardware, or the limitations of software.

[1] We are using the Computer Science Principles Framework dated 2020, by the The College Board.

- Every introductory programming book teaches the concept of an arithmetic expression. We give a formal definition, and show the structure of an expression by placing boxes around sub-expressions. The student thinks we are teaching how to write a correct expression; we are actually teaching recursion, formal grammars, and derivation trees.

The student thinks we are teaching programming, but we are actually teaching Computer Science. Knowledge is not separated into compartments, and our curriculum should not attempt to do so.

Though the Computer Science Principles course is not intended to be a computer programming course, it does have a brief introductory programming component. For that purpose there are editions of this book which use either Java or Python (an edition using C++ is planned). Check the title page or table of contents to determine which edition you are viewing.

This book is available in pdf at `rdw.rowan.edu /oer`.

The source files are available at `cs.rowan.edu/~bergmann/books`.

The primary author may be reached at `bergmann@rowan.edu`

Secondary Authors

Contributors

Kim Poolos, Rowan University

Minor Contributors

Poonam Gupta, Brunswick Academy

Technical Consultant

Joshua Grochowski, Rowan University

Contents

Chapter 1

Creativity

Over the course of many centuries, civilization has evolved as a result of the creative efforts of humans. Whether it be in the arts, sciences, mathematics, psychology, medicine, or philosophy each generation has passed its accomplishments to succeeding generations.

In this chapter we examine creativity from a computational perspective. We attempt to answer the question: What are the computational creative processes currently in use, and what have they created?

1.1 Computational Artifacts

An *artifact* is something that does not occur naturally but which has been created by humans. Examples of artifacts are screwdrivers, automobiles, rocket ships, skyscrapers, oil refineries, books, and computers. A *computational artifact* is an artificat which results from a computational process, usually by a computer or other digital device. Examples of computational artifacts are computers, tablets, telephones, information systems, the internet, and search engines.

When creating a new artifact, humans often use existing methods or processes. However, sometimes they must modify existing methods, or construct totally new methods, to arrive at a new process in the construction of artifacts.

For example, when the internet[1] was created in the 1980's, computers, large and small, were commonplace in universities, business, industry, and government. These computers were able to communicate with each other in pairs, but there was no way for large groups of computers to communicate efficiently with each other. Then a process known as TCP/IP[2] was created to facilitate communication among lots of computers, and the internet was born.

A more recent example of a creative process took place in 2006. At that time people had the capability of sharing text, photographs, sound clips, etc.

[1] Here we are really talking about a predecssor of the internet, known as the ARPAnet
[2] Transmission Control Protocol/Internet Protocol

Figure 1.1: Developed in 2006 by Mark Zuckerberg, Facebook became a creative success, connecting people all over the world.

through email. Then Mark Zuckerberg went public with Facebook^{TM}, and suddenly there was a huge surge of interest in sharing data. Everyone wanted to make their own experiences, in the form of text, pictures, videos, etc, available to everyone else in the world. Why? Mainly because it was so *easy* to do. Few gave any thought to the advisability of making this personal information public, or to the huge financial impact that this data would have for Facebook: If a company such as Facebook knows everything about you - your preferences in clothing, food, travel, that company has valuable marketing information for sale.

Facebook, as a computational artifact, was a creative success (see Figure 1.1).

1.1.1 Exercises

1. Give examples of computational artifacts other than those listed in this section.

2. Give examples of computational processes, such as Facebook, which succeeded in changing the way people use computing devices.

3. Create your own computational artifact. Use a computer, phone, tablet, or similar digital device.

4. (a) In a few sentences describe the 'Internet of Things' (IOT). Use Wikipedia (itself a computational artifact) or other resources to research this collection of artifacts.

   (b) Give some examples of devices which are included in the IOT.

(c) Describe some potential dangers or calamities that could occur involving the IOT if it were to fail or if criminals gained improper access to it.

1.2 Creative Expression and Problem Solutions

In this section we provide some answers to the question: How can we create a computational artifact? The hardware tools to be used here include computers, tablets, and phones. Each of these tools is packaged with software tools for various purposes.

1.2.1 Tool: Computer with Office Package

A computer equipped with a package of tools - word processor, spreadsheet, presentation, email, web browser - can be used to create computational artifacts. This package of tools could be Microsoft Office, Open Office, or other similar package. Web browsers include (Apple) Safari, (Google) Chrome, (MicroSoft) Internet Explorer, and (Mozilla) Firefox. Some examples of artifacts that could be created with these tools are shown below:

Tool	Artifact
Word processor	Term paper
Spreadsheet	Record of expenses
Presentation	Slide show for a research report
Web Browser	Map of Moscow, labeled with Wikipedia descriptions of historic sites

1.2.2 Tool: Computer with Programming Environment

1.2.2.1 Compilers, Interpreters, and IDEs

A computer is capable of executing a sequence of instructions, coded as binary numbers, to perform a task. We have developed software, such as compilers and interpreters, to facilitate the production of these binary programs.

A high level programming language is designed for ease of use by people. Some common programming languages are Java, C++, Python, and Visual Basic. A compiler is a translator. It translates a program written in a high level language to an equivalent program[3] written in the computer's machine language. An interpreter never produces a machine language program. Instead it scans the program and carries out the instructions. Python and Java are examples of two high level languages which are normally interpreted, rather than compiled.

An Integrated Development Environment (IDE) is a tool which facilitates software development and includes capabilities such as:

[3]Two programs are said to be equivalent if they have the same input-output relation, i.e. they have the same purpose.

- Enter and edit program text
- Compile and execute a program
- Locate and fix errors in a program (debugger)
- Manage large collections of software modules

Some examples of common IDEs are

- Eclipse - for use with a variety of languages
- For java programs - BlueJ, IntelliJ, and NetBeans.
- For C or C++ programs - Visual Studio

1.2.2.2 Programming the Computer

Here we provide a general description of programming, and the features of a high-level programming language. A program consists of key words, such as `for`, `if` , `else`, and `while`; also special characters such as parentheses, brackets, semicolons, and arithmetic operators. These keywords and special characters must be used in a way which conforms to the *syntax specification* of the language being used; this is like forming an English sentence according to the rules of English grammar.

A program will consist of:

- Variables - each variable can store a value. In most programming languages the *type* of value that can be stored is determined and specified, by a *declaration* in the program. Some common types are
  - `integer` (positive and negative whole numbers),
  - `floating point` (positive and negative numbers with decimal places, and/or exponents of 10 for scientific notation)
  - `boolean` - true/false value.
  - `string` - Any literal string of characters from the keyboard, usually enclosed in quotation marks, such as `"#3cx?$$bz"`
  - Most languages also provide the capability of programmer-defined types which are constructed using the types provided by the language as building blocks for more complex data types.

  Variable names typically are made up of letters (and digits) and give the programmer the capability of creating meaningful names for variables, such as `salary` and `year2000`.

- Specification of calculations, and the assignment of values to variables. These are normally algebraic expressions, of various types. The value of an expression can be assigned to a variable. For example:

`sum = a + b + c * 3`
The intent is to to add the current values of a, b, and c multiplied[4] by 3, and store the result into the variable `sum`. Another example, using an expression which has a true/false value:

`valid = salary > 0 and hoursWorked > 20`
In each of these examples the equal sign (=) does not mean 'equals' in the usual sense; it means: find the value of the expression on the right, and assign that value to the variable on the left.

- Data structures - Aggregates of related variables grouped together. For example, a `Date` data structure could consist of `day`, `month`, and `year`.

- Control structures - Normally the statements in a program are executed sequentially, in the order in which they are coded in the program. Control structures provide the capability of altering this sequential flow of control. Examples of control structures are:

  - One-way Selection - A decision is made to execute (or not execute) a path, or sequence of statements, in a program. The decision may be based on the current values of variables, i.e. the *state* of the program. This control structure typically uses the key word `if`:

    `if (true/false expression)` statement(s) to be executed only if the expression is true.

  - Two-way Selection - A decision is made to execute one of two paths through the program, using the key words `if` and `else`:

    `if (true/false expression)`

    statements executed if expression is true

    `else`

    statements executed if expression is false

  - Repetition - A sequence of statements is executed repeatedly, a specified number of times, or until a condition is met. This control structure is often called a *loop*,[5] or an *iteration*. Some examples:

    `while (x > 0)  x = x - 3`

    The value of the variable `x` is decreased by 3, as long as its value is positive.

    `for (i=0; i<10; i=i+1) print i`

    The variable `i` is given the initial value 0. The `print` statement is executed repeatedly, as long as the value of `i` is less than 10, and each time the `print` statement is executed the value of `i` is increased by 1. The loop is repeated exactly 10 times.

[4] Multiplication is expressed by an asterisk in most languages, to differentiate with the variable `x`

[5] The word *loop* is derived from diagrams of control structures, in which a repetition appears as a circular loop.

- SubPrograms - Program modules which are self-contained, with a specific purpose, and which can be invoked from a *main* program, or from other subprograms. Subprograms are often called procedures, functions, methods, or simply subs, depending on the programming language. A subprogram may have an explicit result and 0 or more *parameters*, in which case it is similar to a mathematical function. Example:

  ```
  x = -b + sqrt(b*b - 4*a*c)
  ```

  The `sqrt` subprogram (i.e. function) has one parameter, the value of the expression `b*b - 4*a*c`. It *returns* the square root of its parameter, which is then added to `-b`, and the result is stored in the variable `x`.

- Input/output - A program may need to obtain input from a peripheral device such as the keyboard, mouse, scanner, or disk storage. This is called *input*. A program may need to send information to a peripheral device such as a display, printer, or disk storage. This is called *output*. In many programming languages input and output are implemented with subprograms.

1.2.3 Other Tools

Here we summarize some other tools used to create computational artifacts.

1.2.3.1 Image processing

Tools such as Adobe *Photoshop* can be used to modify and/or enhance digital images. Some features of Photoshop include drawing, cropping, moving, erasing, and color replacement. Photoshop has become so widely used, as compared with other image processing software, that the word 'photoshop' is often used as a verb.

Other, more scientific, examples of image processing include:

- Enhancement of photographs with poor resolution (blurred images) using edge-finding methods to improve the resolution. This is done in astronomical images, producing details in images of remote planets, stars, galaxies.

- Robotic vision systems provide raw data from a camera, and are designed to interpret that raw data as meaningful objects (chair, desk, pen, ...)

- X-ray and MRI machines used in health care can be improved to help the radiologist identify malformations (tumors, hairline bone fractures, small blocked arteries, etc.) which are difficult to spot.

- X-ray machines used for security at airports, concert venues, and sport venues are enhanced with image processing software to find restricted objects such as weapons and explosives.

1.2.3.2 Web page development

World Wide Web pages originally contained text and graphics content for display coded in HTML[6]. In recent years this language has been improved to provide sound, video, interaction with the user, and database capabilities. A web page designer can use tools such as Adobe Flash, Google Web Designer, EZGenerator, and Microsoft Expression Web.

1.2.3.3 Video production

Most digital cameras have the capability of recording moving images, with sound; these recordings are known as *video clips*. Software enables the editing of one or more clips to remove sections or splice sections of video content. Examples of software tools include: Blender VSE, Ldenlive, Shotcut, and Natron. Video production has experienced a huge surge in popularity in recent years, largely as a result of the popularity of free hosting sites on the Internet, such as YouTube.

1.2.3.4 Database management

Large collections of data often need to be organized in such a way that needed information can be extracted easily and quickly. A *Database Management System* (DBMS) serves this purpose. Major developers of DBMS software include SAP, Oracle, and MicroSoft (Access).

1.2.3.5 Game development

Digital games have had a big impact on society. Platforms for games include special purpose devices (Xbox, Playstation), general purpose computers, and web-based games. Most games are designed for stand-alone single players, but some allow for multiple players, co-located, or remotely connected on the Internet.

1.2.3.6 Artificial Intelligence

Artificial Intelligence (AI) has been defined as the instilling in machines of capabilities normally associated only with human intelligence. Over the years the nature of AI has changed. For example, many years ago the development of chess-playing software was included as AI research. However, with advances in the speed of computer hardware, and the programming of winning strategies, champion-level chess playing machines are more common, and no longer considered part of AI.

With the advent of lower storage costs and the Internet, huge amounts of data have accumulated on public computers (known as 'the cloud'). Searching this enormous repository of information, and drawing meaningful conclusions

[6]HyperText Markup Language

from it, is now the domain of AI. There are many examples of this in business, military, and healthcare domains.

1.2.3.7 Robotics

One important part of AI is robotics. [7]

Once the realm of science fiction, robotic devices have become pervasivie in modern society. Self-driving vehicles are a reality and are likely to reduce transportation costs of people and goods in the near future (as well as eliminating jobs for human drivers).

Manufacturing assembly lines (particularly for automobiles) have made extensive use of robotic arms (with vision capability)

Military applications of robotics include devices used for reconnaisance and/or battle, such as flying drones, missiles with vision and course-correction, and land-based mobile units.

Household robots are becoming increasingly popular for cleaning, security, controlling of appliances, and even baby-sitting.

1.2.4 Exercises

1. Given a spreadsheet containing a column of numbers, show formulas to calculate the average, maximum, and minimum value at the bottom of the column.

2. Given a word processing document, show how spelling and grammar can be checked for accuracy. Find examples of English sentences which contain spelling and grammatical errors, that your word processor does not detect.

3. Use a slide presentation applicaton to draw a rough diagram, or map, of your school's main classroom building.[8]

4. If you are familiar with a programming language, such as Java, C++, or Python, write a program to find the smallest value in a list of numbers.

5. The section on Other Tools mentions Image Processing. A self-driving car manages to move through a city, obeying traffic laws, and avoiding collisions. Describe some of the problems the self-driving car will need to solve. What constitute the inputs to the car. How would it process that input? What knowledge would have to be pre-programmed into the self-driving car?

[7] Isaac Asimov's three laws of robotics (circa 1950):
I. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
II. A robot must obey the orders given it by human beings except where such orders would conflict with Law I.
III. A robot must protect its own existence as long as such protection does not conflict with Laws I or II.

[8] Microsoft Powerpoint and LibreOffice Presentation are examples of a slide presentation application.

6. Build your own web page using one of the following tools:

   - Code your web page in HTML (HyperText Markup Language) using a plain text editor such as TextEdit (Mac) or NotePad (Windows). HTML commands are begun with a tag in angle brackets, and terminated by the same command, preceded by a slash. For example,

     `This concept is <bold> important </bold> to remember.`

   - Use a packaged web developer tool which displays the web page as you are developing it. Some popular tools are: `Sublime Text, Chrome Developer Tools, jQuery, GitHub, Twitter Bootstrap, Angular.js, and Sass`[9]

   - Use a developer site, such as Yahoo.com, which has pre-fabricated components for your web page.

7. Create a video clip of yourself making a video clip. Use an application such as QuickTime on a Mac, or MultiMedia Player on a Windows PC.

8. Use Microsoft Access, or similar application to create a database of your favorite professional sport league. Include games played, scores, rosters, etc.

9. Give some examples of applications of robotics not mentioned in this section.

1.3 Extending the Human Experience

What is it that sets humans apart from less intelligent forms of life? We know that whales and apes can communicate with each other in fairly sophisticated ways and many life forms can solve problems which have been posed for them in a lab setting, especially when rewarded with food. However, most other life forms do not possess the gift of creativity which humans have. This creativity could involve the development of tools and technology, or it could involve more aesthetic artifacts such as music, art, language, poetry, etc.

1.3.1 Creativity

Many forms of creative expression use a computer simply for convenience. A simple drawing, sketch, or even a painting can be done manually, or with a computer. However, more ambitious creations would never have been possible without computers. One good example is the generation of special effects for movies, and entire movies, which involve so-called *animation*. In movies such as Cars, Toy Story, The Lion King, the Star Wars series, etc. Life-like animated characters and scenes are so realistic that it is often difficult to distinguish

[9] Wikipedia

between what is animated and what is real. [10] Large companies, such as Pixar, DreamWorks, and Industrial Light & Magic have been formed to produce movies with animation technology.

With music similar tools have been developed to aid composers with the creation, orchestration, and play-back of musical compositions. The composer can now get immediate feed-back on a composition without involving performing artists, speeding up the composition process immensely.

Modern art has seen an influx of creations by humans, using the computer to draw still or animated images. Many of these are so complex that they would not have been possible without the computer.

However, in both art and music, the actual creation has largely been done by humans, using the computer as a tool. While it is true that this has resulted in artifacts that the human might not have been able to create without using the computer, attempts at computer-composed music and art have not yet evolved in the main stream of publicly appreciated arts. If and when this happens, it would mark a quantum step forward for AI.

1.3.2 Problem solution

Here we attempt to answer the question: How have computers aided in the solution of problems?

In today's world it is rare that any complex problem is solved without aid from a computer, particularly those problems which involve the analysis of much data. Some of the more important and common examples are:

- Weather forecasting is extremely important to commerce, transportation, military, and agriculture. It is not done simply so that you know whether to bring an umbrella. You may have noticed that in the last decade weather forecasts have become increasingly more accurate, long-range, and specific to a time and place (see Figure 1.2). Weather forecasting is done by simulating atmospheric conditions on a fast computer with lots of memory, or on a cluster of fast computers. This simulation software stores large arrays of atmospheric conditions such as air temperature, pressure, humidity, and wind velocity over a large geographic area. Each such array represents conditions at a specific instant of time. The computer is programmed to deduce the values of these arrays at the next incremental instant of time. Without computers this kind of simulation would not be at all feasible (which is why mid-twentieth century weather forecasting was so poor, as compared with today).

- Any large complex system of tasks can now be organized using a computer. These had been done in the past, without computers, but not on a large scale. Examples are:

[10] The problem of producing output which is indistinguishable from human expression is usually referred to as the *Turing Test of Intelligence*. This measure of intelligence was first defined in 1950 by the British Mathematician Alan M. Turing, the father of computing.

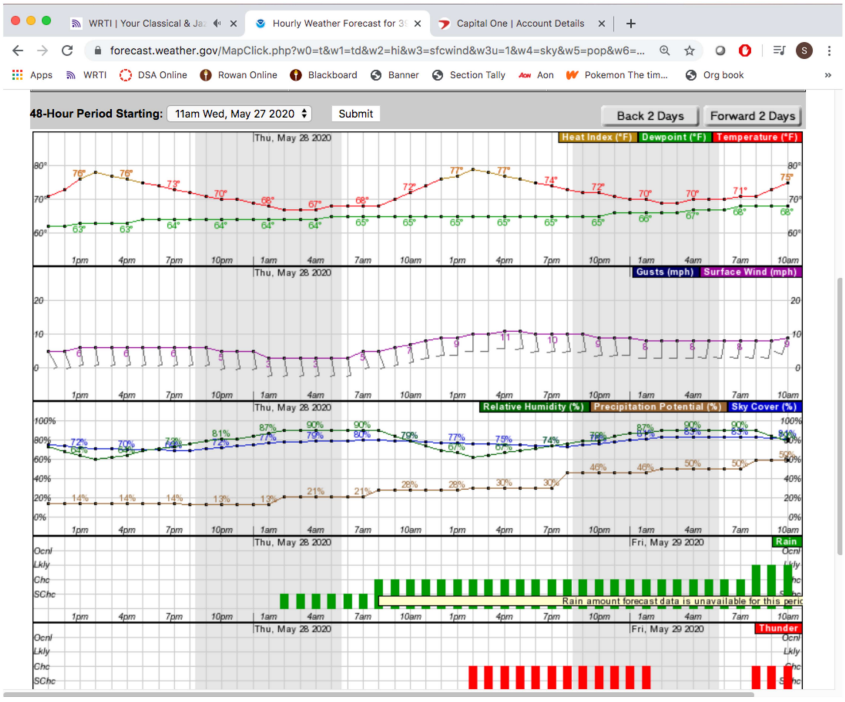


Figure 1.2: A web site for weather forecasting provided by the U.S. National Weather Service: `www.weather.gov`

  - Transportation systems, such as airline flight scheduling. Hundreds, if not thousands, of flights are scheduled daily, requiring coordination of crews, airport runways, flight paths, etc. All this was done before the advent of computers, but not on a large scale.
  - Construction of large buildings, such as skyscrapers involves a detailed process of sequential steps: ironwork, concrete/synthetic walls and floors, wiring for electric power and communications, HVAC. This is now coordinated with computers, facilitating the construction of increasingly complex structures.
  - Communication systems, such as telephone, television, and the internet itself, now rely entirely on digital computer technology.
  - Supply chain and inventory applications are now much larger and more complex than ever. Without computers, a large retail outlet such as Wal-Mart would not have been possible.

- Data analytics[11] is the science of extracting useful information from large quantities of data. This process allows us to discover valuable marketing information when applied to the behavior of people on the internet. In health care, data analytics can be used to draw conclusions about disease using information such as geography, nutrition, air quality, etc.

- Techniques in artificial intelligence and robotics are enabling us to diagnose disease, perform surgery, and build self-driving vehicles.

- Ancient Greek, Arabic, and Asian civilizations were known to have made independent discoveries in mathematics. Some, particularly the Greeks, were able to prove important theorems and describe natural phenomena mathematically. However there are several theorems which remained unproven - until the computer age arrived. Using the computer as a tool, or assistant, mathematicians have been able to solve the following mathematical problems. Some examples of important theorems which were too difficult for humans to prove, but were proved with the aid of computers are:

  - The *four-color-map* theorem: Given a plane with regions (e.g. a map) the regions may be assigned colors such that no two neighboring regions have the same color. This can be done using only four different colors (see Figure 1.3).

    In the nineteenth century it was proved that this could be done with five colors, and it was conjectured that four colors are sufficient, but no one was able to prove this until 1976 when mathematicians at the University of Illinois used a computer to search through a large space of maps to complete the proof. This was the first time that a computer had been used for a difficlut proof, and it was initially met with skepticism and controversy. Today it is widely accepted.

[11] Many now define Data Analytics as Applied Data Science.

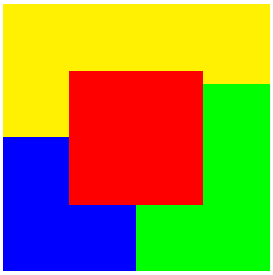


Figure 1.3: After trying for centuries, mathematicians were finally able to prove, with help from a computer, that any map can be colored with only four colors

- Many are familiar with the popular game called *Rubik's Cube*, consiting of 27 adjoining multi-colored ccubes in 3 planes of 9 cubes each. Each plane can be rotated to form a new configuration. The object is to rotate planes until each face consists of cubes of the same color. In 2010 a group of researchers used a supercomputer to prove that any initial configuration or a Rubik's Cube can be solved in at most 20 face turns.
- The Boolean Pythagorean Triples Problem. A Pythagorean Triple consists of three positive integers, (a,b,c) such that $a^2+b^2=c^2$. One well-known Pythagorean Triple is (3,4,5), which are the lengths of the sides of a right triangle. Other examples are (5,12,13) and (28,45,53). The Boolean Pythagorean Triples Problem asks whether it is possible to assign one of two colors to every positive integer, such that the three integers in no Pythagorean Triple are all the same color.

  In 2016 researchers at the University of Texas showed that such a coloring is possible only up to 7824. Using a computer system consisting of thousands of CPUs, the proof took two full days.

Software has also been developed to create proofs from scratch. Given a logical proposition, the computer outlines a proof of the proposition. Some examples of such systems are shown below:

- E is a theorem-proving software system for first-order logic [12] which was developed at the Technical University of Munich.
- Otter was developed at Argonne National Lab. It has been replaced by Prover9.
- SETHEO (Set Theory) is based on model elimination calculus from the Technical University of Munich. E and SETHEO have been combined to form a more powerful theorem prover known as E-SETHEO.

[12] A logic system which has not only true-false statements (propositions) but also quantifiers such as *for-all* and *for-each*, forming what is known as a *predicate calculus*.

1.3.3 Exercises

1. Name a few movies in which it is difficult to determine the difference between live recorded images and computer-generated images. Are there any scenes where live recorded images are enhanced by a computer?

2. What are the most accurate weather forecasting sites?

   - Web sites: weather.com? weather.gov?
   - TV stations: CBS local news, ABC local news, NBC local news?
   - Accuweather reported on the radio?

   Maintain a database of predictions by these various forecasters, and recored the actual conditions (precipitation, temperature, humidity, wind, etc.) on each date, to determine which is the most accurate.

3. Which professional sports use data analytics to adjust their play or strategies? How is the data used?

Chapter 2

Abstraction

Today's world is filled with a huge morass of details, most of which we may find unintelligible, useless, and/or uninteresting. The process of *abstraction* is what we use to ignore the details which we don't need, and extract useful information from the details which are needed. This process of abstraction is something we do all the time, without giving it much thought. For example, think of all the light rays entering your eye's pupil and triggering impulses to the visual cortex in your brain, right now. Your brain has learned to ignore much of what is coming in, to focus on what it needs (the letters making up this text), and to extract (hopefully) useful meaning from those letters.

This chapter will investigate some of the ways that abstraction is used in computer science. It is used to simplify the design of complex software or hardware components, to process large quantities of data, to facilitate artificial intelligence or robotics.

2.1 Binary sequences and digital devices

Computers are binary machines. That means they are constructed from bi-stable devices - devices which can be in one of two states at any one time.[1] Any machine composed of bi-stable devices is said to be *digital*. Any machine which is not digital is said to be an *analog* machine. For example, a wall clock showing hour, minute, and second hands is analog (though its inner workings may use digital electronics). A clock which shows the hour, minute, and seconds numerically is digital.

Years ago small metal doughnut-shaped cores were used as bi-stable devices. The two states were determined by the direction of magnetization in the core (clock-wise or counter-clockwise). [2]

[1] The advent of quantum computing will change this. A qubit can be in two different states at the same time, but that is beyond the scope of this text.

[2] Since these metal cores were used to build memory, the word 'core' was used interchangeably with 'memory', hence the phrase 'core dump' is still used today.

In more recent years semiconductor materials[3] have a huge matrix of densely populated 'holes' in which the presence, or absence, of an electron determines the state.

Whatever physical device is used, we refer to the two states of the device as 0 and 1. Each device (i.e. each core, or each semiconducto hole) is referred to as a *binary digit*, or *bit*. An example of a *sequence* of bits is `01101110`. Computers can use bit sequences to represent various kinds of information, such as integers, numbers which are not integers, plain text, graphic images, sound, and video clips. When we represent information as a binary sequence (a sequence of 0's and 1's) we refer to the information as *digital* information; or we may say the information has been *digitzed*.

2.1.1 Exercises

1. A binary sequence has a length. For example the length of the binary sequence 10011 is 5 bits. Fill in the empty cells in the table shown below:

Binary sequence example	Length	Number of different sequences of the same length
10011	5 bits	32
	3 bits	
111111		
		1024

2. In how many ways can checkers be placed on an 8x8 checker board, with no more than 1 checker on each square? You may give your answer as a power of 2.

3. Construct a digital device using small squares of paper. One side of each paper square should have an X.

   (a) Arrange some of the paper squares in a row on a table top to represent the bit sequence 11001110, where the X side represents a 1, and the other side represents a 0.

   (b) The length of your bit sequence is 8 bits. How many different bit sequences of length 8 could you form?

4. During the American revolution, Paul Rever was told to look at the lights in the steeple of Christ Church in Boston, to determine how the British were attacking: one if by land, two if by sea.

   Apparently there were two lights in the steeple. How many different messages could have been formed with the two lights?

[3] Silicon, Germanium, and other elements have semiconductor properties.

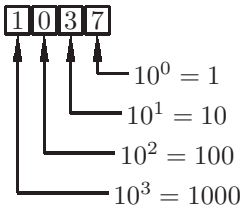


Figure 2.1: The decimal representation of 1037 (1037 = 1000 + 0 + 30 + 7)

2.2 Representing integers

Here we introduce one way of representing whole numbers using a sequence of bits, i.e. we introduce a *digital* representation for whole number.

2.2.1 Number systems

2.2.1.1 Decimal numbers: base 10

In the ninth century AD Arab and Hindu mathematicians independently devised a representation for whole numbers using ten symbols.[4] Today we call this the *decimal* or *base 10* system. The low order digit in a base 10 number represents a multiplier of 1 (10^0).[5] The next digit represents a multiplier of 10 (10^1). The next digit represents a multiplier of 100 (10^2). The digit at position i represents a multiplier of 2^i, as shown in Figure 2.1. For example, the number 1024 can be thought of as: $1 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 4 \times 10^0$

2.2.1.2 Binary numbers: base 2

In computer science we are interested in binary numbers, i.e. base 2, because computers work exclusively with bistable devices as explained in the previous section. Therefore, a whole number will be represented by a sequence of binary digits (a sequence of bits). The low order digit in a base 2 number represents a multiplier of 1 (2^0). The next digit represents a multiplier of 2 (2^1). The next digit represents a multiplier of 4 (2^2). The digit at position i represents a multiplier of 2^i, as shown in Figure 2.2. For example, the number 19 can be thought of as: $1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$, or 10011_2[6] xxx

The

2.2.1.3 Octal numbers: base 8$^{\otimes}$

In computer science we will frequently need to display memory contents as a sequence of bits. For convenience, a sequence of bits can be expressed in base 8,

[4] It is generally believed that they used 10 symbols because humans have 10 fingers.

[5] x^0 is 1 for any value of x.

[6] The subscript on a number represents the *base*; here we will assume that the base is a decimal number. For base 16 we will occasionally use the letter x.

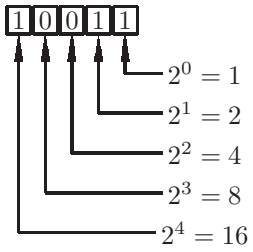


Figure 2.2: The binary representation of 19 (19 = 16 + 2 + 1)

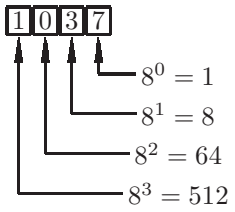


Figure 2.3: The octal representation of 543 (543 = 512 + 0 + 24 + 7)

or *octal* (the symbols are 0,1,2,3,4,5,6,7). Each octal symbol represents 3 bits, as shown in the table below.

octal	binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

The low order digit in a base 8 number represents a multiplier of 1 (8^0). The next digit represents a multiplier of 8 (8^1). The next digit represents a multiplier of 64 (8^2). The digit at position i represents a multiplier of 8^i, as shown in Figure 2.3. For example, the number 135 can be thought of as: $2 \times 8^2 + 0 \times 8^1 + 7 \times 8^0$ or 207_8.

By interpreting each octal digit as 3 binary digits, we can easily see the sequence of bits that is represented by 207_8:

```
  2   0   7
010 000 111 = 010000111
```

In recent years most computer scientists have come to favor base 16 (hexadecimal) to base 8 (octal) for a short-hand representation of a sequence of binary

digits.

2.2.1.4 Hexadecimal numbers: base 16

In computer science we will frequently need to display memory as a sequence of bits. For convenience, a sequence of bits can be expressed in base 16, or *hexadecimal*, sometimes abbreviated simply as *hex*. In base 16 we will need 16 symbols. Using the existing symbols from base 10 (0..9) we still need six more symbols; we will use the symbols a,b,c,d,e,f.[7] Thus the hexadecimal symbols are 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f. Each hexadecimal digit represents 4 bits, as shown in the table below.

hexadecimal	binary	decimal
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
a	1010	10
b	1011	11
c	1100	12
d	1101	13
e	1110	14
f	1111	15

The low order digit in a base 16 number represents a multiplier of 1 (16^0). The next digit represents a multiplier of 16 (16^1). The next digit represents a multiplier of 256 (16^2). The digit at position i represents a multiplier of 16^i, as shown in Figure 2.4. For example, the number 135 can be thought of as: $2 \times 16^2 + 0 \times 16^1 + 7 \times 16^0$ or 207_{16}.

By interpreting each hexadecimal digit as 4 binary digits, we can easily see the sequence of bits that is represented by $1a5_{16}$:

```
   1    a    5
0001 1010 0101 = 000110100101
```

When we wish to display a long sequence of bits, it is clearly more efficient to display hexadecimal digits, rather than binary digits.

[7] We use the lower case letters but many sources use the upper case letters A,B,C,D,E,F to mean the same thing

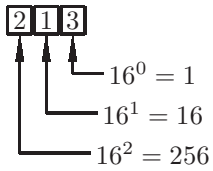


Figure 2.4: The hexadecimal (base 16) representation of 541 (541 = 512 + 16 + 3)

2.2.2 Representing positive integers

To store a positive whole number in the computer's memory, we notice that a sequence of bits can represent a whole number. We choose to use the sequence which is equal to the base 2 representation of the whole number, as shown below. In doing so, we will speak of the *word* size as the length of the sequence of bits with which we are dealing. In the table shown below, the word size is 5 bits [8].

[8] Since the word size, 5, is not a multiple of 4, the high order hex digit represents 1 bit, and may only be 0 or 1

Integer	bit sequence	hexadecimal
0	00000	00_x
1	00001	01_x
2	00010	02_x
3	00011	03_x
4	00100	04_x
5	00101	05_x
6	00110	06_x
7	00111	07_x
8	01000	08_x
9	01001	09_x
10	01010	$0a_x$
11	01011	$0b_x$
12	01100	$0c_x$
13	01101	$0d_x$
14	01110	$0e_x$
15	01111	$0f_x$
16	10000	10_x
17	10001	11_x
18	10010	12_x
19	10011	13_x
20	10100	14_x
21	10101	15_x
22	10110	16_x
23	10111	17_x
24	11000	18_x
25	11001	19_x
26	11010	$1a_x$
27	11011	$1b_x$
28	11100	$1c_x$
29	11101	$1d_x$
30	11110	$1e_x$
31	11111	$1f_x$

Note that with a 5-bit word, there are 32 different bit sequences; these range from the binary values 0 through 31. If we had a 6-bit word, we would have twice as many sequences, 64, ranging from 0 through 63. In general, for an n-bit word there are 2^n bit sequences; hence we can represent any positive integer in the range $[0..2^n - 1]$.

If an arithmetic operation produces a result which is beyond the range described above, this is known as an *overflow condition*. Computers can be programmed to halt execution when an overflow condition occurs. Alternatively, they can continue to execute, with possibly meaningless results. For example with a 5-bit word, the integers in the range [0..31] can be represented. If an attempt is made to add 17+20, for example, overflow occurs because the true

result will not fit in a 5-bit word. If execution continues, the result will be 5:

```
   10001   =  17
 + 10100   =  20
---------------
   00101   =   5
```

The carry out of the high order digit is discarded. Most modern computers have a word size of 64 bits, and can easily work with positive integers through $2^{64} - 1$. Most modern programming languages have data types which enable the programmer to use 8, 16, 32, or 64-bit words.

2.2.3 Representing negative integers

We also wish to represent negative integers. Note that there are no '-' signs in the computer's memory; there are only 0's and 1'2. Our strategy is to use roughly half of the bit sequences to represent negative numbers, half to represent positive numbers, and one of the bit sequences to represent 0[9]. We make the following choices:

- The bit sequence consisting of all 0's represents the integer 0, regardless of the word size.
- The bit sequence consisting of all 1's represents the integer -1, regardless of the word size.[10]
- Any other bit sequence beginning with a 0 represents a positive integer.
- Any bit sequence beginning with a 1 represents a negative integer.

This convention is known as the *two's complement representation* of integers. Other schemes could have been used (and have been used in older computers), but most modern computers use two's complement representation for integers.[11]

For a 5-bit word the bit sequences 00000 through 01111 will represent positive integers, and the bit sequences 1 Our table representing integers using two's complement representation is shown below:

[9] Zero is neither positive nor negative

[10] To understand this choice, imagine a new car with odometer reading 00000. If the odometer subtracts miles when the car backs up, and the new car backs up for one mile, the odometer will read 99999, representing -1.

[11] The reason for this choice is a result of the ease of designing hardware to add and subtract (see the open source book on *Computer Organization*).

Integer	bit sequence	hexadecimal
-16	10000	10_x
-15	10001	11_x
-14	10010	12_x
-13	10011	13_x
-12	10100	14_x
-11	10101	15_x
-10	10110	16_x
-8	10111	17_x
-8	11000	18_x
-7	11001	19_x
-6	11010	$1a_x$
-5	11011	$1b_x$
-4	11100	$1c_x$
-3	11101	$1d_x$
-2	11110	$1e_x$
-1	11111	$1f_x$
0	00000	00_x
1	00001	01_x
2	00010	02_x
3	00011	03_x
4	00100	04_x
5	00101	05_x
6	00110	06_x
7	00111	07_x
8	01000	08_x
9	01001	09_x
10	01010	$0a_x$
11	01011	$0b_x$
12	01100	$0c_x$
13	01101	$0d_x$
14	01110	$0e_x$
15	01111	$0f_x$

Note that there are 16 negative numbers (-1 through -16), but only 15 positive numbers (1-15). For any word size there will always be one more negative number than there are positive numbers.

What happens when we attempt to add a positive number to a negative number? As an example, we attempt to add 7 and -2:

```
    00111  =  7
+   11110  = -2
---------------
    00101  =  5
```

We get the correct result![12] This is the beauty of two's complement representation.

The largest integer that can be represented with a 5-bit word is $2^4 - 1 = 15$, and the smallest integer is $-2^4 = -16$. For an n-bit word the largest (positive) number that can be represented is $2^{n-1} - 1$ and the smallest (negative) number is -2^{n-1}.

2.2.4 Exercises

1. Convert each of the following hex numbers to a 12-bit binary number. Hint: Do not convert to decimal.
   (a) 022_x
   (b) $1a3_x$
   (c) fff_x
2. Convert each of the following 12-bit binary numbers to hexadecimal. Hint: Do not convert to decimal.
   (a) 101110100001_2
   (b) 111111111111_2
   (c) 000011100101_2
3. Memorize the following bit string:
   $111111101011101011011110_2$
   Hint: Convert the bit string to hexadecimal, and memorize the hex digits.
4. Assume we are representing positive integers only (ignore two's complement). What is the largest integer that can be represented with a word size of:
   (a) 3
   (b) 7
   (c) 10
   (d) 11
5. Show a friend that you can count from 0 to 31 using only the fingers on one hand. (Tell your friend not to be offended when you get to the number 4)
6. $^{\otimes}$ Convert each of the following octal numbers to a 12-bit binary number. Hint: Do not convert to decimal.
   (a) 1037_8
   (b) 2465_8

[12] There is a carry out of the high order bit, which is discarded.

(c) 7777_8

7. ⊗ Convert each of the following 12-bit binary numbers to octal.
Hint: Do not convert to decimal.

(a) 101110100001_2

(b) 111111111111_2

(c) 000011100101_2

8. ⊗ Perform each of the following conversions.
Hint: Convert to binary first.

(a) Convert $7f3_x$ to octal.

(b) Convert 2307_8 to hexadecimal.

9. ⊗ Show the 6-bit two's complement representation of each of the following decimal integers:

(a) +7

(b) +25

(c) -12

(d) -17

10. ⊗ Assuming we are working with two's complement representation of integers, what are the largest and smallest integers that can be represented with a word size of:

(a) 3

(b) 6

(c) 10

(d) 11

11. ⊗ Working with two's complement representation, for any word size, find the following sums:
Hint: Solve the problem for small word sizes.

(a) The largest number plus the smallest number.

(b) The largest number plus the largest number.

(c) The smallest number plus the smallest number.

2.3 Representing non-integer numbers

We have seen how to represent integers; we now investigate a way of representing non-integers, using only bit sequences. In our math courses, we often use non-integers (and very large integers) such as 3.14 72/3 and 6.02×10^{23}. But in the computer's memory we have only bit sequences, i.e. 0's and 1's. There are no decimal points, fractions, nor superscripts (for exponents).

We will represent a non-integer using two integers, called the *mantissa* and *exponent*. The exponent is assumed to be an exponent of 10.[13] The represented number is the exponent multiplied by a power of 10, determined by the exponent:

$number = mantissa \times 10^{exponent}$

Both mantissa and exponenet may be negative. The exponent directs that the decimal point be shifted to the left or right, depending on the sign of the exponent. An exponent of +5 shifts the mantissa's decimal point 5 places to the right. An exponent of -3 shifts the mantissa's decimal point 3 places to the left. In this scheme, known as *floating point*, we can represent integers, non-integers, numbers very close to 0, and very large numbers. Some examples of floating point numbers represented by two integers are shown below:

mantissa	exponent	number represented
3	1	$3 \times 10^1 = 30.0$
17	-3	$17 \times 10^-3 = 0.017$
-33	5	-33×10^5 = -3300000.0
602214	18	$602214 \times 10^{18} = 6.02214 \times 10^{23}$ = Avogadro's number

Floating point representation has some interesting properties. Some fairly common numbers have no exact representation, such as:

- The result of 1.0/3.0 is approximately 0.3333333333 but no matter how many 3's we write, it will never be exact. The precision is limited by the word size of the mantissa.
- Irrational numbers such as π and $\sqrt{2}$ have no exact representation.
- On a binary computer the exponent would be an exponent of two, and thus there is no exact representation for one tenth![14]

Because of the inaccuracies inherent in floating point numbers, great care must be used when working with this type of number. We work with integers whenever possible, and resort to floating point numbers only if necessary.

[13] In real floating point hardware, the exponent is an exponent of 2.

[14] For an explanation see the open source textbook on *Computer Organization* at `http://cs.rowan.edu/simbergmann`.

2.3.1 Exercises

1. Show the mantissa and exponent (of 10) as integers for each of the following floating point values.

   (a) 113.0

   (b) -1024.0

   (c) 1024.032

   (d) 0.000128

   (e) 1×10^{100}

   (f) -123.45×10^{-12}

2. ⊗ Assume we use a 4-bit word for the mantissa, and a 4-bit word for the exponent (both using two's complement representation) to represent a floating point number.

   (a) What is the largest possible floating point number that can be represented?

   (b) What is the smallest possible floating point number that can be represented?

   (c) What is the smallest possible positive floating point number that can be represented?

   (d) What is the largest possible negative floating point number that can be represented?

3. ⊗ Assume we use a 3-bit word for the mantissa, and a 3-bit word for the exponent (both using two's complement representation) to represent a floating point number.

   (a) What is the range of values for the mantissa?

   (b) What is the range of values for the exponent?

   (c) Show a table of all possible floating point values having a 4-bit mantissa and a 3-bit exponent.

   (d) Are there any whole numbers which have no floating point representation? If so, show a few of them.

   (e) Subtract 2000.0 from the next largest floating point number. What is the difference?

   (f) Subtract 0.0002 from the next largest floating point number. What is the difference?

2.4 Representing integers with unlimited precision$^{\otimes}$

We have seen that there are limitations on the size of the integer that can be represented with a fixed number of bits. For any n-bit sequence, there are 2^n different values.

Most modern computers use a fixed size for integers, 64 bits. However, there are applications, particularly in the areas of cryptography and computer security, where we need greater precision; i.e. we may wish to do arithmetic with numbers that are hundreds of bits in length. This is generally done with software:[15] We use a sequence of integers, possibly decimal digits, to represent a single number. The list representing this number can have a variable length, so we can add more digits when needed (for example, to avoid overflow). Think of the number 9,453,930,402 represented as the list of decimal digits [9,4,5,3,9,3,0,4,0,2]. One possible operation would be to multiply this number by 10, giving a result of [9,4,5,3,9,3,0,4,0,2,0].

The phrase *unlimited precision* is really a misnomer. All computers have limited storage, so that is one limitation on the size of a number. As a practical matter, that is not the limiting factor when working with these huge numbers; some fairly simple operations, such as multiplication of two numbers will take so much time that this turns out to be a limitation on the size of numbers that can be used.

2.4.1 Exercises

1. Each of the following lists of decimal digits represents a single integer:
   `[2,3,0,4,9,0,9]`
   `[3,0,5,7,0,9]`
   Show the list which would be produced when adding the two integers.

2. If we have a subtract operation on integers with unlimited precision, the result could be negative. We will need to find a representation for negative numbers. One possible solution is to use *tens complement* when representing an integer as a list of decimal digits. Tens complement is analagous to twos complement. A number is negative if and only if its high order digit is greater than 4. For example:
   ```
   [9,9,9,9] = -1
   [9,9,9,8] = -2
   [9,9,9,7] = -3
   [9,9,9,9,9] = -1
   ```

   (a) Show a list of 3 decimal digits representing +23.

   (b) Show a list of 3 decimal digits representing +723.

   (c) Show a list of 3 decimal digits representing -25.

[15] As these applications are becoming increasingly important, special purpose hardware now exists for the same purpose: unlimited precision arithmetic.

Character	Code	Character	Code	Character	Code	Character	Code
A	65	a	97	0	48	Space	32
B	66	b	98	1	49	!	33
C	67	c	99	2	50	”	34
D	68	d	100	3	51	$	35
						%	36
Z	90	z	122	9	57	&	37

Figure 2.5: Some codes from the ASCII coding system

(d) Show how to add the two numbers 23 + -25 given above, producing the correct result.

2.5 Representing plain text

To represent plain text, we simply use integer numbers to represent the symbols in our character set. This representation is called a *coding scheme* or simply a *code*. An early code to represent the symbols found on standard keyboards is known as ASCII[16] and was standardized in 1963 for early teletype machines. Some examples of these code values are shown in Figure 2.5.

Students often ask “If the character ‘$’ is stored as the number 35, why do we see it as a ‘$’? The ‘$’ exists as the number 35 (actually 0100101_2) in the computer’s memory, but when that code is sent to a peripheral device, such as a display or a printer, that device’s hardware uses the value 35 to construct an array of pixels that resemble a ‘$’. Note that there are also codes for the 10 decimal numerals: 0..9. The space (and tab, and newline) also have their own codes.

ASCII is a 7-bit code, which means there are $2^7 = 128$ possible symbols which can be encoded. It was subsequently agreed that 7 bits is not enough. We need to be able to store more than 100 special characters.[17]

Moreover, as computers are used in non-English speaking countries, we need to represent the characters in foreign alphabets as well. For example, most modern chinese dialects have over 6,000 characters. A modern code known as Unicode was developed in 1987 at the Xerox corporation and at Apple Computer. Unicode is so named because it is a “unique, unified, universal encoding” as described by its developers. It is a 16-bit code which means that it may have up to $2^{16} = 65,536$ different symbols. This is thought to be big enough to accommodate the alphabets of today’s world. ASCII is a sub-code of Unicode, which means that the ASCII codes are replicated in Unicode.

[16] American Standard Code for Information Interchange

[17] Non alphanumeric characters such as !, ?, &, * are often referred to as ’special’ characters.

2.5.1 Exercises

1. Use a search engine, such as Google, to look up the plain text characters having ASCII codes of 68, 72, 109, 45, 55, and 63.

2. (a) Subtract the ASCII code for `B` from the ASCII code for `b`, and show the result.
   (b) Subtract the ASCII code for `Q` from the ASCII code for `q`, and show the result.
   (c) What is the result of subtracting the ASCII code for any upper case letter from the ASCII code of the corresponding lower case letter.

3. (a) Show the ASCII code for `B` and the ASCII code for `b` in binary.
   (b) Show the ASCII code for `Q` and the ASCII code for `q` in binary.
   (c) Describe how the ASCCII code for any upper case letter differs from the ASCII code of the corresponding lower case letter.
   (d) If you are given the ASCII code for any lower case letter in binary, show an easy way to find the ASCII code for the corresponding upper case letter, in binary,
   (e) If you are given the ASCII code for any upper case letter in binary, show an easy way to find the ASCII code for the corresponding lower case letter, in binary,

4. (a) Show the ASCII codes of all 10 numeric characters: `0..10`.
   (b) Given the ASCII code of any numeric character, show how to obtain the corresponding integer, using a 7-bit two's complement representation.

5. The ASCII code 10 represents a non-printable character, usually referred to as LF (LineFeed). The ASCII code 13 represents a non-printable character, usually referred to as CR (Carriage Return). These terms are derived from old teletype machines (similar to typewriters) which could send and receive text. To advance to a new line the machine needed to do both a CR and a LF, in either order. How have these codes complicated the usage of a newline character in modern operating systems (try a Google search)?

2.6 Representing graphic images

2.6.1 Black and white images

We also need to represent graphic images, photographs, works of visual art, etc. in the computer's memory. When we speak of a *digital* image, we are talking about an image that is represented by a sequence of 0's and 1's. To understand how this is done, take a close look at a newspaper photograph. You'll see that

```
00000000000000000000
00000000010000000000
00000000111000000000
00000001111100000000
00000011111110000000
00000111111111000000
00001111111111100000
00011111111111110000
00111111111111111000
00000000000000000000
```

Figure 2.6: An array of bits representing the pixels making up an image of a triangle, and the image with each pixel shown as a black dot

it consist of many small black dots for a black and white photograph, or small colored dots for a color photograph. These dots are called *pixels* (or picture elements). A black and white photograph can be represented by a large array of pixels, where each pixel is represented by one bit: 0 means there is no dot, and 1 means there is a dot.

Figure 2.6 depicts an array of 0's and 1's which make up an image of a triangle. This is the simplest way to represent the pixels in an image, and it is often called a *bitmap* image; there are other ways of representing the pixels in an image.

When viewing a bitmap image which has curved boundaries, such as a circle or ellipse, it may appear to have rough edges. A device such as a printer or display can vary the physical distance between pixels, to make a rough edge appear smooth. Using more densely packed pixels on the page, or on the screen, will make the image appear to be more smooth. This property is known as *resolution*, and it is often measured in *dots-per-inch* (dpi).

Before 1975 all photography equipment used analog technology, typically a film which required chemical processing to produce photographic prints. The leader in this field was the Kodak corporation. When digital cameras were first introduced, they were either expensive, or had poor resolution when compared with Kodak's film cameras. Soon, however, Kodak's digital competitors increased the resolution of their products. Because Kodak did not anticipate the rapid advances in digital technology, it failed to move its photograph business from analog to digital technology quickly enough. Kodak filed for bankruptcy in 2012.

Figure 2.7: Packing Red, Green, and Blue components (and alpha value) of a color into a single 32-bit integer

Color	Red	Green	Blue
RED	255	0	0
YELLOW	255	255	0
ORANGE	255	200	0
CYAN	0	255	255
GRAY	128	128	128

Figure 2.8: Some examples of color values, showing the relative Red, Green, and Blue components

2.6.2 Color images

To represent color images, we would use an array of integers, rather than an array of bits.[18] Each integer would represent a particular color. Colors are generally represented by red, green, and blue components (RGB). Any color can be represented by some combination of these three primary colors.

To represent a single color, relative red, green, and blue components are *packed* into a single 32-bit word, as shown in Figure 2.7.[19] There is a fourth component, known as an *alpha* value, which is a measure of the pixel's transparency level; it does not affect the actual color of the pixel. The diagram shows that each component is an 8-bit quantity, so the possible values for each component are 0..255 ($2^8 = 256$).

Figure 2.8 shows a few examples of colors with their red, green, and blue components.

2.6.3 Exercises

1. Show the bitmap image of a smiling face. Show both the array of binary digits, and the array of dots, as shown in Figure 2.6.

2. Describe how you would scan the 0's and 1's in a bitmap image to find the boundary of a solid object, such as the triangle shown in Figure 2.6.[20]

3. A word processing application, such as Microsoft Word, permits text in various colors. Open a Word document and click the text color button. This should open a small window with several color choices, and an option for more colors. Choose more colors. This will open a Colors window, with

[18] Each integer, though, is made up of a sequence of bits, as discussed in the previous section.
[19] The RGB system described here is the one in the java.awt package.
[20] This is an important aspect of *image processing*, known as *edge* detection.

a slider that can be used to choose an arbitrary color. Move the slider all the way to the left to view the available colors.

(a) What color appears to be a blend of red and green?

(b) What color appears to be a blend of red and blue?

(c) What color appears to be a blend of blue and green?

4. Show the numeric values, in decimal, of the red, green, and blue components of a color pixel that has the value $0a1bff03_{16}$

2.7 Representing sound and video

We may also wish to store (and process) other kinds of information, such as sound and video. These can also be thought of as information, and can therefore be digitized, for storage in a digital device such as a computer. In doing so, each sound clip or video clip may be thought of as a sequence of bits.

2.7.1 Representing sound

What is a sound? A sound consists of:

- A source which creates a disturbance in the air pressure
- The propagation of this disturbance through the air, in the form of *sound waves*[21]
- A receiver which senses the disturbance in the air pressure, and stores or processes the disturbance in some way.[22]

How can we represent these disturbances in air pressure as digital information? When the air pressure waves enter our ears, our brain receives a signal from the auditory nerves. To represent sound, all we need do is store a digital version of the pressure waves. This is diagrammed in Figure 2.9, in which the horizontal axis is time, and the vertical axis is the amplitude of the pressure waves. The wave at top left of Figure 2.9 represents just one cycle of a sound wave. The wave at top right represents a louder sound at the same pitch because the amplitude is greater, but the frequency of the cycles is the same. The wave at bottom left represents a higher pitch, because there are twice as many cycles in the same time period; the loudness is the same as the sound at top left.[23] The quality of the sound is determined by the number of values sampled per unit time. To represent sound with very high quality (high fidelity) requires many

[21] Sound waves are longitudinal waves, as opposed to light which is propagated through transverse waves.

[22] There must be a receiver. If a tree falls in the forest and no "receiver" hears it, there is no sound.

[23] The higher pitch would be one *octave* higher, as the musical scale is a logarithmic function of the frequency of the air waves.

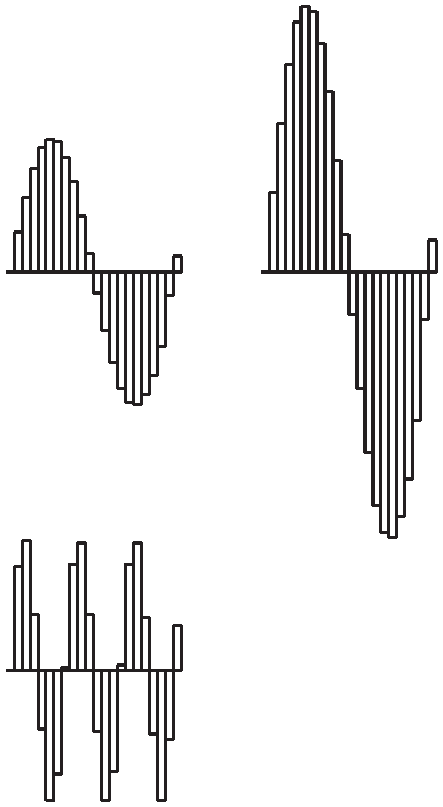


Figure 2.9: Representing sound (i.e. air pressure waves) as discrete numbers. Top right: a louder sound. Bottom left: a higher pitch.

numbers. Consequently a sound clip is merely a sequence of whole numbers representing varying air pressure. Most sound clips are actually a compressed format of these numbers. Some examples of compression formats are .wav and .mp3.

2.7.2 Representing video

A video clip consists of images and, possibly, sound. A silent video clip has no sound. To represent moving images we use a sequence of bitmap images. If these images are presented to the human eye in rapid succession, they are perceived as a moving image. In the past, a movie film was stored as a sequence of still images on a long film, and stored on a reel. An animated movie consist of a sequence of images, each of which is called a *cel*. The cels can be drawn by hand, or generated by a computer.

Video clips can require much memory, depending on the length of the clip, and the desired quality (many images, with very small changes from one image

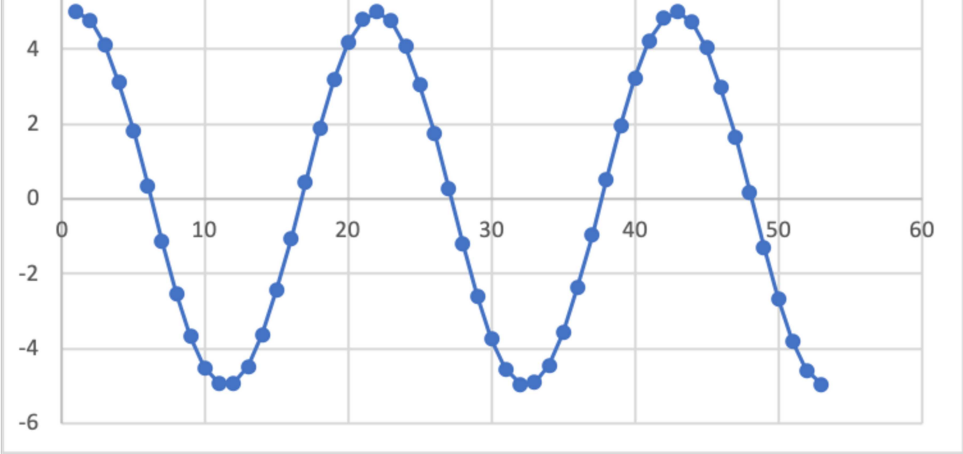


Figure 2.10: A waveform with a high amplitude (loud) and a low frequency (pitch)

to the next, result in a high quality (smooth) video clip). Because of the large size of video clip files, various compression schemes are used to store the video clip. When a video clip is played, it is decompressed to obtain the air pressure values. When a video clip is edited, it is recompressed for efficient storage. Some common compression schemes, indicated by the file extension, are .mov, .gif, .mpeg, .qt, and .amv.

In the 1980's video formats included VHS and Betamax, two competing video formats using analog film technology. Companies such as Blockbuster Video invested heavily in this analog technology. Digital video formats became common in the 1990's, and by 2000 streaming technology[24] overtook distribution via optical disk. Because Blockbuster did not adapt to these new digital technologies quickly enough, it filed for bankruptcy in 2010. Nicholas Negroponte, founder of the MIT Media Lab, predicted in 1995 that *shipping bits* would be more cost effective than *shipping atoms*.[25]

2.7.3 Exercises

1. Two waves are *superimposed* by adding their numeric amplitudes, at each point on the time axis.

   (a) Draw a diagram of the superposition of the waves shown in Figure 2.10 and Figure 2.11.

   (b) Describe the sound (pitch, amplitude, and quality) of the superposition of those two waveforms.

2. Sound files consist of many large numbers, representing a sequence of air pressure values over time. Much space can be saved by compressing sound files. One such compression algorithm works by storing the initial value, followed by the change from one value to the next. Since these changes are usually small, they require less space (fewer bits) than the original numbers.

   (a) Show how the list of numbers shown below can be compressed:

   ```
   [109284892, 109284894, 109284895, 109284890, 109284802,
   109284907, 109284892, 109284891]
   ```

[24] Distribution of information, such as movies, over the internet, as opposed to using optical disk (DVD) technology

[25] *Being Digital* published by Knopf Publishers

.

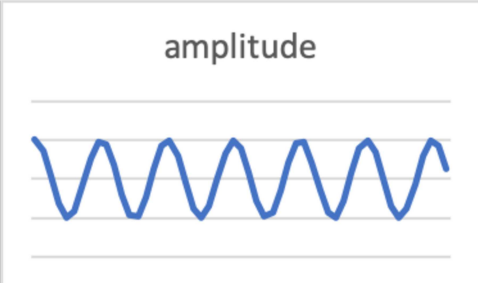


Figure 2.11: A waveform with a low amplitude (soft) and a high frequency (pitch)

(b) Show how the list of numbers shown below, representing a compressed file, can be decompressed:

```
[109284892,7,12,-3,4,0,-10,7]
```

3. In some movies depicting moving carts, wagons, chariots, etc. where the wheels have spokes, it may appear that the spokes are not moving, or are moving in the reverse direction. Explain this phenomenon.

2.8 Looking into memory

We have seen how to represent numbers, plain text, graphic images, sound, and video as bit sequences. When information can be represented as a sequence of bits, we say that it is *digital*.

2.8.0.1 Stored programs

In addition to data stored in memory we also have programs. A program consists of a sequence of binary *instructions* stored in the computer's memory. Each instruction has:

- A binary code for the operation (add, subtract, move data, shift bits within a word, jump out of sequence, etc.).
- Binary specifications of the operands, i.e. the values to be added, subtracted, moved, etc.

- Other information, such as the number of bits to be shifted in a shift instruction.
- Where the result of the operation is to be placed.

Program execution takes place by executing the instructions, in the sequence in which they are stored in memory. Jump instructions are capable of altering this sequence. A jump to a prior instruction will implement a repetition of a group of instructions. A jump may be *conditional*; a conditional jump is taken only if some prior condition is satisfied. A conditional jump can be used to execute one of two possible groups of instructions, depending on the result of a prior operation.

2.8.0.2 Stored data

If we were to examine a bit sequence in the computer's memory it would be impossible to interpret that sequence as meaningful information. For example, the 16-bit sequence represented by $2142_{16} = 0010000101000010_2$ could be representing either of the following:

- The integer value 8,514 ($2 \times 4096 + 1 \times 256 + 4 \times 16 + 2 \times 1$)
- The two characters `!B` (`!` $= 21_{16}$ and `B` $= 42_{16}$)

Other examples could show that a 32-bit sequence could represent an integer or a floating point value. How is one to know which interpretation is valid? It depends on the instruction which is using that portion of memory. Some computer instructions assume that the memory operands are integers, some instructions assume the memory operands are floating-point numbers, and some instructions assume the memory operands are text.

2.8.0.3 Stored programs and data

We have seen that bit sequences in memory are used to represent both instructions *and* data. The design of most modern computers is based on this principle, known as the *Von Neumann architecture*. Conceivably, such an architecture permits a program to modify itself, with instructions that treat other instructions as data.[26]

2.8.1 Exercises

1. Given the bit sequence represented by $fff2_{16}$:
   (a) Interpret this bit sequence as an integer, with two's complement representation.

[26] While some consider this a powerful feature, others discourage its use, particularly when building parallel processes.

(b) Interpret this bit sequence as a floating point number, with an 8-bit mantissa, followed by an 8-bit exponent of 10, both of which use two's complement representation.

2. Given the bit sequence represented by 3262_{16}:

   (a) Interpret this bit sequence as an an integer, with two's complement representation.

   (b) Interpret this bit sequence as two ASCII characters.

3. The following 16-bit sequences are different representations of the same number:

   - 3235_{16}
   - 0019_{16}
   - 1900_{16}

   (a) Which is a two's complement integer?

   (b) Which is a floating point number, consisting of an 8-bit mantissa and an 8-bit exponent of 10?

   (c) Which is a string of two characters?

   (d) What is the number?

2.9 Abstractions in programs

We have seen that the notion of *abstraction* may involve the hiding of details, to bring out the essential aspects of an artifact. When we think of an integer, we normally don't think of it in its form as a binary bit sequence. Similarly, when we think of a graphic image, we don't see the digital bit sequence which is representing that image.

2.9.1 Program abstractions

Abstractions also apply to computer programs. To explain this we use an example of a fairly common problem: searching a list of names for a particular target. An example of a list of names is shown in Figure 2.12.[27] If we wish to determine whether a particular name (the target) is in that list, we need to understand a few processes:

- To compare the target name with a name in the given list, we must be able to compare two names to see if they are equal. A more general term for a name would be a *string of characters*, which would include any character, not just alphabetic characters.

[27] The first name in the list is at position 0. Normally, in computer science, we begin counting at 0.

0	alice
1	joseph
2	mary
3	jimmy
4	joseph

Figure 2.12: A list of 5 names

- To compare two strings for equality, we must be able to compare individual characters within those strings, as shown in Figure 2.13.
- To compare two characters for equality, we must be able to compare their ASCII codes. Two characters with the same ASCII code are equal characters.

The computer hardware will be able to compare two integers for equality, so that is how two characters are compared. The next step would be to build software that compares arbitrary strings of characters. Here is how that process could work, to compare an arbitrary string, `str1` with another arbitrary string, `str2`.

1. Let the variable `i` represent the position of the first character.
2. Compare the character at postion `i` of `str1` with the character at position `i` of str2.
3. If they are not equal, terminate with a result indicating the two strings are not equal.
4. Increment `i`, i.e. add one to the value of `i`.
5. Repeat from step 2 as long as `i` does not extend beyond the end of either list.
6. If there are more characters in one of the lists, terminate with a result indicating the two strings are not equal.
7. Terminate with a result indicating the two strings are equal.

This sequence of steps, describes how two strings of characters can be compared for equality. Such a sequence of steps is called an *algorithm*. An algorithm is supposed to solve a particular problem, in this case the comparison of strings of characters. An algorithm must eventually halt with a correct solution. The algorithm which compares strings is diagrammed in Figure 2.13, in which we

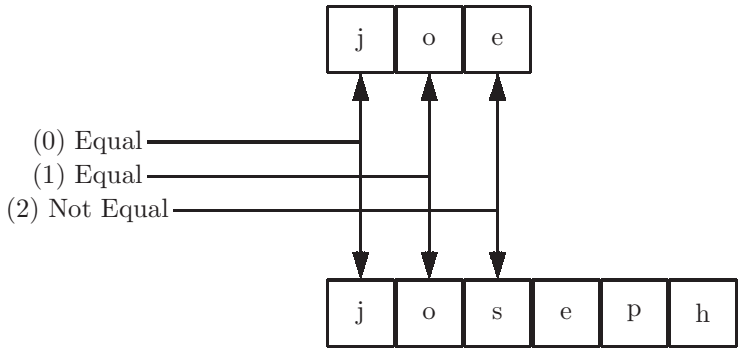


Figure 2.13: Comparing two names for equality. Comparing one character at a time, three character comparisons are needed.

show the three steps that are needed to compare the name `joe` with the name `joseph`.

Once we have developed an algorithm (or program) to solve a problem, we can use that algorithm in the solution of other problems. When doing so we often hide the details of the algorithm being used; as long as we know its purpose and how to use it properly, we can ignore its inner workings. This is program abstracion!

In our example, we can use the string comparision algorithm (let's call it `strcmp`). In order to make general use of it to compare *any* two strings we need to be able to tell it which strings are being compared. This is done with *parameters*.[28] Parameters are generally shown in parentheses, and separated by commas after the name of the program, as in:
`strcmp(joe,joseph)`
which would be a comparison of the two strings `joe` and `joseph`, and which would result in an indication that the strings are not equal.

We now have the tools needed to solve the original problem: search a list of names for a given target name. Let's call it `search(list,target)`. This algorithm should not only determine whether the given target is in the given list, but it should also tell us the position of the first occurrence of the target in the list (there could be duplicate values in the list). If the target is not in the list, the result should be -1. Here is the algorithm:

1. Given a list of names, and a target name
2. Let `i` be the position of the first name in the list.
3. Compare the target name with the name at postion `i`, using the strcmp program that we have described above:
   `strcmp( target, name at position i of the list)` .

[28] This notation is derived from the notation for functions in mathematics: For example, `cos(x)` is a trigonometric function with one parameter, or argument.

4. If the result produced by `strcmp` is equality, terminate this algorithm, indicating that the target was found in the list of names. The result is the value if `i` which is the position of the first occurrence of the target.

5. Increment `i`, i.e. add one to the value of `i`.

6. Repeat from step 3, if `i` has not gone beyond the end of the list.

7. Terminate, indicating that the target was not found in the list. The result is -1.

This completes our example of an algorithm which searches a list of names for a particular target name. It makes use of another algorithm which compares arbitrary strings of characters. The two algorithms are similar, in that both:

- repeat a group of steps (this repetition is called a *loop*)
- make use of a loop *counter*. The loop counter (the variable `i`) is used to:
  - select an item from a list of items
  - determine whether to continue repeating the loop

The important point is that when designing the search algorithm we made use of the `strcmp` algorithm without exposing its details. That is program abstraction!

2.9.2 Levels of abstraction in software

2.9.2.1 An example: arithmetic expressions

In computer programs we often need to specify a sequence of arithmetic operations, or calculations, which need to be performed. To do this we normally borrow notation from mathematics, and specify the calculations using *algebraic expressions*, or *arithmetic expressions*. [29]. An example of an arithmetic expression would be:
`(a+b) * (c-d)`.[30]

Here we give a somewhat precise defintion of an arithmetic expression. An arithmeitic expression (or simply an `expr`) may be any of the following:

1. A constant, such as 45, -12.5, or 0
2. A variable, such as x, y, z, or salary
3. The sum of two expressions:
   expr + expr
4. The difference of two expressions:
   expr - expr

[29] Pronounced with the stress on the third syllable: a-rith-me'-tic
[30] The asterisk represents multiplication.

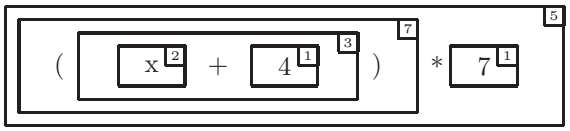


Figure 2.14: Boxes show the structure of the expression $(x + 4) * 7$. The rule number is shown in the upper right corner of each box.

5. The product of two expressions:
   expr * expr

6. The quotient of two expressions:
   expr / expr

7. An expr inside parentheses:
   (expr)

Each part of the above definition is called a *rule*; the definition of `expr` has 7 rules. Note that this definition of `expr` uses the word `expr` in rules 3-7. This is known as a *recursive* definition. Using the word being defined as part of the definition is not a problem, as long as there is a part (rules 1 and 2) which does not use the word being defined.[31]

This definition not only tells us *exactly* what constitutes an expression, but it also imposes a structure on expressions. This structure can be seen by drawing a box around each expression within a given expression. As an example, consider the arithmetic expression `(x+4)*7`. The structure of this expression is shown in Figure 2.14, in which we also show which definition rule is used, by including the rule number from the definition in the upper right corner of the box.

2.9.2.2 Programming languages

We have seen that a program consists of a sequence of binary coded instructions in the computer's memory. These instructions are typically very simple, fundamental operations such as:

- Add two numbers
- Subtract a number from another number
- Move a word (i.e. a binary number) from one location in memory to another location
- Decide whether to alter the sequence of instructions, by jumping to another instruction

[31] A definition of `expr` *must* be recursive because an arithmetic expression is a recursive structure.

This *language* of primitive binary instructions is often referred to as *machine language*.

To evaluate the expression shown in Figure 2.14, `(x+4)*7`, the following machine language instructions could be used:

- Obtain the value of the variable `x` from memory
- Add the constant `4` plus the value of `x`, saving this temporary result.
- Multiply the tempororay result by 7, saving this temporary result.

We emphasize that these instructions are coded in binary, forming a machine language program. There was a time, in the early days of computing, when all programming was done in this binary machine language. This was a very tedious, time-consuming, and error-prone process. Thus software (i.e. other programs) were developed to ease the burden on the programmer.

The first such improvement was the development of *assembly language* in which the binary codes were replaced by *mnemonics* and the binary operand addresses were replaced by symbols. Assembly language also allowed for *comments*, descriptions of the computations, ignored by the computer, but for the benefit of human readers. The machine language program described above, could be written as shown below, where comments begin with a # symbol:

```
LOAD  x       # Obtain the value of x from memory
ADD   4       # x+4
MULT  7       #  (x+4)*7
STORE result  # Store (x+4)*7 in memory
```

A program known as an *assembler* was written[32] to translate an assembly language program into machine language.

Though assembly language was a huge improvement over machine language for ease of use, software soon became complex and cumbersome to deal with. At that time *high level* programming languages were developed. These typically replaced the primitive instructions with algebraic expressions. In a high level programming language the calculation shown above can be specified as we normally think of it: `(x+4)*7`. A few of the early programming languages were Fortran (Formula translator), BASIC, and COBOL (Common Business Oriented Language). A program called a *compiler* was developed to translate a program written in a particular high-level language to the machine language for a particular computer. For example, a Fortran compiler for the IBM 360 computer would translate any program written in the Fortran language into an equivalent[33] program in IBM 360 machine language.

We can describe an even higher level of abstraction: *Database languages*. A database is a collection of structured data, typically stored on a peripheral

[32]The first assemblers were written in machine langauge. After that new, improved assemblers were written using existing assemblers.

[33]Two programs are equivalent if they have the same input/output relation. I.e. If two programs always produce the same output for any input, the programs are equivalent.

device such as a disk or USB drive[34]. For example, a university has a database of students, storing information for each student:

- Student's name
- ID number
- Home address
- GPA
- Registration information

A database language would allow the user to specify operations on the database without specifying the details necessary for the operation. For example, the user could extract certain students from the database, those with a high GPA thus:

```
show students for GPA > 3.0
```

A statement such as this in a database language, can be implemented by a program consisting of several statements written in a high-level language.

What we are describing here is really a sequence of abstractions, in which we build primitive tools, then use them to build more sophisticated tools, which in turn are used to build even more sophisticated tools. Each higher level of abstraction is less detailed and easier to use but makes use of lower level tools that have previously been developed. Multiple levels of abstraction can be shown as an *abstraction hierarchy*. The abstraction hierarchy of programming languages that we have described is shown with a diagram in Figure 2.15. In this figure an arrow is used to indicate that a tool is used to create a more sophisticated tool. This is abstraction!

2.9.3 Exercises

1. Devise an algorithm to compare strings of characters for alphabetic order. Your algorithm should be similar to the one presented in this section that compares strings of characters for equality. Your algorithm should return an integer:

   - A negative number if the first string comes before the second string alphabetically
   - A positive number if the first string comes after the second string alphabetically
   - Zero if the strings are equal.

   Hint: Subtract the ASCII codes of corresponding chracters.

[34] Also known as a *flash* drive or thumb drive.

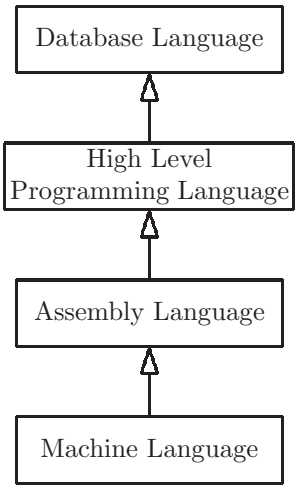


Figure 2.15: An abstraction hierarchy of programming languages

2. If we have a list named `students`
   [jim,mary,joe,jack,joe,sue]
   What will be the result of each of the following?
   Hint: The first name is at position 0.

   (a) `search(students,mary)`

   (b) `search(students,joe)`

   (c) `search(students,sue)`

   (d) `search(students,bob)`

3. Show a diagram, similar to Fig 2.14 for each of the following arithmetic expressions:

   (a) `b / 3)`

   (b) `a + b * 3`
   Hint: Multiplication and division *take precedence over* addition and subtraction,

   (c) `(a + b) * 3`

   (d) `a - b + 4`
   Hint: When two operations have the same precedence, choose the left-most operation first.

   (e) `a - (b + 4)`

x	y	x AND y	x OR y	NOT x
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Figure 2.16: Definitions of AND, OR, and NOT in formal logic

2.10 Levels of abstraction in hardware$^{\otimes}$

Many of the principles used in developing software apply to hardware as well. *Hardware* includes physical devices, such as computers, keyboards, monitors, phones, etc., as well as the components included in those devices such as memory chips, processor chips, conntecting wires, etc. *Software* is distinguished from hardware by noting that software consists of information, or bit sequences.[35]

When developing hardware, we can use abstraction to hide details, and use primitive tools to build increasingly complex tools, just as we did with software.

2.10.0.1 Formal logic and digital logic

In formal (mathematical) logic we have three operations: AND, OR, and NOT. These operations are shown in Figure 2.16, and they can be exposed in everyday English:

- Elephants are pink AND 2+3 = 5. is a false statement.
- Elephants are pink OR 2+3 = 5. is a true statement.
- Elephants are pink OR 2+3 = 6. is a false statement.
- NOT Elephants are pink. is a true statement.

Note that the AND and OR operations have two operands, whereas the NOT operation has only one operand. In order for the result of AND to be true, both operands must be true. In order for the result of OR to be false, both operands must be false. Note that the result of OR is True when both operands are True. In computer science we call this an *inclusive* OR, as distinguished from an *exclusive* OR (not shown in Figure 2.16) which is False when both operands are True.

These concepts form the basis for *digital logic gates*, which are the fundamental building blocks of hardware devices. In digital logic, since we are dealing exclusively with 0's and 1's, 0 represents False, and 1 represents True. Also, for convenience in writing logical expressions, we use a raised dot for AND, a

[35]The distinction between hardware and software is becoming less clear, as functions traditionally implemented with hardware are now implemented in Read-Only Memory (ROM), leading to the term *firmware*, which is somewhere between hardware and software.

x	y	x AND y $x \cdot y$	x OR y $x + y$	NOT x x'	x XOR y $x \oplus y$
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

Figure 2.17: Definitions of AND, OR,NOT, and Exclusive OR (XOR) in digital logic

plus for OR, and a postfix apostrophe for NOT.[36] In computer science we define another kind of OR operation known as *exclusive OR* (XOR), designated with a $\oplus$ symbol. The exclusive OR operation is false when both operands are true, thus it *excludes* the case where both operands are true from the cases yielding a true result. These digital logic operations are shown in Figure 2.17.

To demonstrate the difference between (inclusive) OR and XOR, consider the following statements, both of which are true:

- There are 26 letters in the English alphabet OR 2+3 = 5 `[Inclusive OR]`
- For recess, students are permitted to play outside or students are permitted to play inside (implying that no student may play both inside and outside). `[Exclusive OR]`

We can now write logic expressions, where the notation is exactly the same as algebraic expressions, but the operands are either 0 (i.e. False) or 1 (i.e. True). Some examples:
$1 \cdot 0 = 0$
$0' = 1$
$(1 + 0) \cdot 1 = 1$
$1 + 1 = 1$
$((1 + 0) \cdot (0 + 0))' = 1$
$((1 + 0) \cdot (0 + 0)')' = 0$
$1 \oplus 0 \oplus 1 = 0$

We will also use logic *variables* in logic expressions. For example, the variable `x` represents a 0 or 1 (i.e. False or True) value. The table in Figure 2.17 is a *truth table* (actually four truth tables). Each column shows all possible results for every possible value of the variables `x` and `y`. The precedence of operations in a logical expression is the same as for an arithmetic expression:

- NOT takes precedence over AND: $x \cdot y' = x \cdot (y')$
- AND takes precedence over OR: $x \cdot y + z = (x \cdot y) + z$

[36] Some authors use an *overbar* for the NOT operation, rather than a postfix apostrophe. $x' = \overline{x}$

x	y	$x + y$	$x \cdot (x + y)$	$(x \cdot (x + y))'$
0	0	0	0	1
0	1	1	0	1
1	0	1	1	0
1	1	1	1	0

Figure 2.18: Truth table for the logical expression $(x \cdot (x + y))'$, showing intermediate results

x	y	z	$x + y$	$x + z'$	$(x + y) \cdot (x + z')$
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Figure 2.19: Truth table for the logical expression $(x + y) \cdot (x + z')$ showing intermediate results

- XOR has the same precedence as OR[37]

Figure 2.18 shows a truth table for the logical expression $(x \cdot (x + y))'$. Since the expression has two variables, `x` and `y`, there are four rows in the truth table, corresponding to the four possible values of the two variables. Note that this table shows intermediate results, $x + y$ and $x \cdot (x + y)$, which are used to reach the final result in the last column:
$(x \cdot (x + y))'$,

A truth table for an expression with three variables would have eight rows, because there are $2^3 = 8$ possible logical values for three logical variables. A truth table for the expression $(x + y) \cdot (x + z')$ is shown in Figure 2.19

From the definitions of the logical operations in Figure 2.17 it should be clear that certain logic expressions involving variables have a known result, regardless of the variables' values. For example, we can see from Figure 2.17 that $x \cdot 0$ will always be 0 regardless of the value of x, because $0 \cdot 0 = 0$ and $1 \cdot 0 = 0$. This is called a logical *property*;[38] it is true for every value of the variable x. A table of common properties is shown in Figure 2.20.[39]

[37] Since $x + (y \oplus z) \neq (x + y) \oplus z$, it is a good idea to include parentheses when mixing OR and XOR in an expression.

[38] These are often called *identities*, but since one of the properties is named *identitiy* we use the term properties.

[39] See the open source textbook *Computer Cryptography* for important applications of the XOR properties in private key cryptography.

Property	Property	Name
$x + 0 = x$	$x \cdot 1 = x$	Identity
$x + x = x$	$x \cdot x = x$	Idempotent
$x + 1 = 1$	$x \cdot 0 = 0$	Null
$x + x' = 1$	$x \cdot x' = 0$	Complements
$(x')' = x$		Involution
$x \cdot (y + z) = x \cdot y + x \cdot z$	$x + (y \cdot z) = (x + y) \cdot (x + z)$	Distributive
$x + x \cdot y = x$	$x \cdot (x + y) = x$	Absorption
$x \cdot y + x' \cdot z + y \cdot z = x \cdot y + x' \cdot z$		Consensus
$(x + y)' = x' \cdot y'$	$(x \cdot y)' = x' + y'$	DeMorgan's Laws
$x \oplus 0 = x$	$x \oplus 1 = x'$	
$x \oplus x = 0$	$x \oplus x' = 1$	
$(x \oplus y) \oplus z = x \oplus (y \oplus z)$		Associativity

Figure 2.20: Some properties of digital logic

2.10.0.2 Digital logic gates$^{\otimes}$

Now that we have introduced digital logic we can expose the realization of these operations in hardware. The fundamental building block of hardware is called a *logic gate*, or simply a *gate*. There is a logic gate for each of our logic operations, and they are shown in Figure 2.21

The arrows going into a gate represent the operands of the operation. Note that the AND, OR, and XOR gates each have two input arrows because these operations each have two operands.[40] The NOT gate, also known as an *inverter* has only one arrow input, because the NOT operation has only one operand. Each of the logic gates has one arrow coming out, representing the result of the operation.

2.10.0.3 Digital logic circuits$^{\otimes}$

We can now connect logic gates with wires, forming a *digital logic circuit*. The ouput of a logic gate may be used as an input to another logic gate. The logic circuits that we construct are realizations of logical expressions. As an example, Figure 2.22 is a logic circuit which implements the logical expression $x \cdot y + y \cdot z$.

2.10.0.4 Half adders

Out objective here is to explore the notion of abstraction in hardware. We plan to do that by designing a series of hardware components which, ultimately, will be capable of adding binary numbers.

Our first step is to build a device known as a *half adder*. A half adder has two inputs (x and y) and two outputs (Sum and Carry). The truth table for a half adder is shown in Figure 2.23. We have actually combined two truth tables,

[40] We will generalize this later to allow more than two inputs to an AND, OR, or XOR gate. This can be done because these operations are associative.

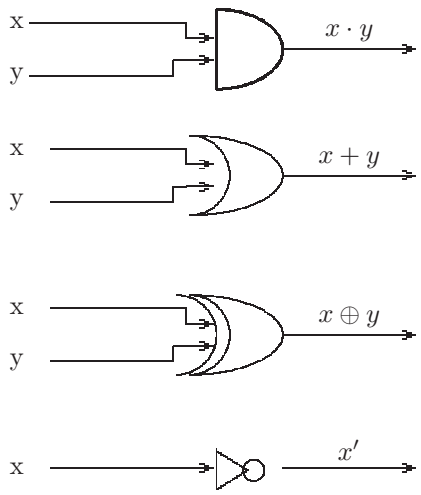


Figure 2.21: Four basic logic gates: AND, OR, XOR, and NOT

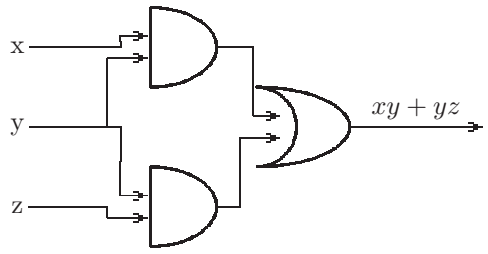


Figure 2.22: An implemenation of the logical expression $xy + yz$ using logic gates

x	y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure 2.23: A truth table for a half adder

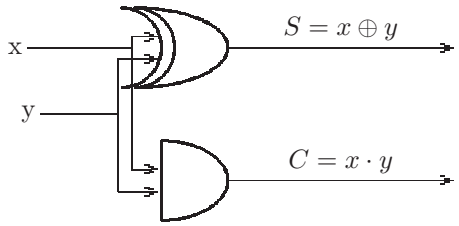


Figure 2.24: A logic diagram implementing a half adder. S is the sum, and C is the carry, .

one for the `Sum` and one for the `Carry`, into a single table. To realize the half adder, note that:

- The column for the `Carry` in Figure 2.23 is the same as the AND column in Figure 2.17; thus $Carry = x \cdot y$
- The column for the `Sum` in Figure 2.23 is the same as the XOR column in Figure 2.17; thus $Sum = x \oplus y$

The logic circuit for a halfAdder can now be built, to agree with Figure 2.23. It will have two inputs, and two outputs, and is shown in Figure 2.24.

Now that we have designed a half adder, we can introduce a *block diagram* which represents a half adder. A block diagram is a higher level diagram for a logic circuit in which all the internal gates and components are not shown. Instead it shows the inputs, with labels, and the outputs with labels. It also contains the name of the diagram inside the box, and often it will show the number of inputs and outputs (2x2 in this case). Together with the truth table for half adders (see Figure 2.23), the half adder block diagram provides everything which is needed when using half adders to build more complex logic circuits. That is abstraction!

A block diagram for the half adder is shown in Figure 2.25

2.10.0.5 Full adders

We next turn our attention to the design of a component known as a *full adder*. A full adder has three inputs and two output. The truth table for the full adder is shown in Figure 2.26. When implementing the addition of two binary numbers, a full adder will be used to implement the addition in one 'column' of

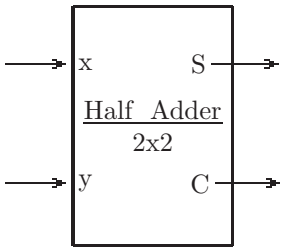


Figure 2.25: Block diagram for a Half Adder. S is the one-bit sum, $x + y$, and C is the one-bit carry $x \oplus y$.

x	y	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 2.26: A truth table for a full adder: three inputs and two outputs

the numbers. Thus, in each column we have the two bits being added, plus the carry bit from the previous column. We also need to produce a carry into the next column; thus, a full adder will need three inputs and two outputs. Note that the carry into the low order column will always be 0, and the carry out of the high order column will be discarded.

For example, with a 4-bit word, when adding +7 and -2 we have:

```
    0 1 1 1   =   +7
    1 1 1 0   =   -2
--------------------
    0 1 0 1   =   +5
```

Below we show the same operation, with the carry bits shown at the top:

```
  1 1 1 0 0            (Carry bits)
    0 1 1 1   =   +7
    1 1 1 0   =   -2
--------------------
    0 1 0 1   =   +5
```

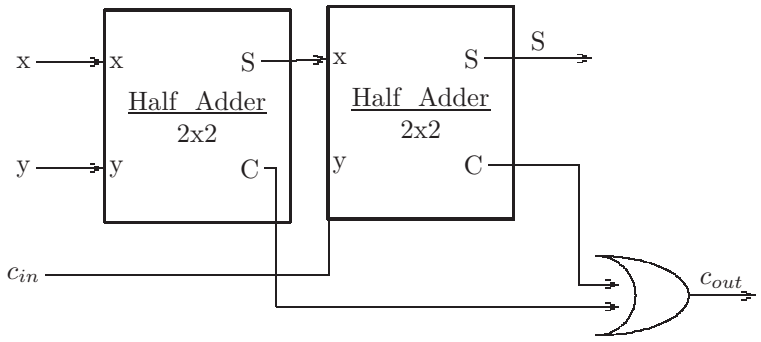


Figure 2.27: Logic diagram for a full adder, using two half adders

To build the full adder, we could derive logical expressions from the truth table, as we did for the half adder.[41] However, it may be easier to use two half adders to build a full adder. The design is shown in Figure 2.27

To understand the two outputs of the full adder in Figure 2.27 consider the outputs separately:

- The Sum output (S) will be the sum of $x+y+c_{in}$ where c_{in} represents the CarryIn input. To find the sum, the first half adder finds the sum $x+y$, and the second half adder adds that sum to c_{in} to produce $x+y+c_{in}$.
- The CarryOut output (c_{out}) of the full adder will be a 1 if there was a CarryOut from either half adder; thus an OR gate is used to produce the c_{out} output.

Having designed our full adder, we can present a block diagram for a full adder, in which we do not show the inner details; we simply show the three inputs and two outputs, along with the name of the component: Full Adder. This block diagram, together with the truth table for the full adder (Figure 2.26) is all that is needed to use a full adder. The details are not visible; that is abstraction! The block diagram of the full adder is shown in Figure 2.28. Note that we have placed the inputs on the right and the outputs on the left; this is to facilitate our next level of abstraction: a binary adder.

2.10.0.6 Binary adder

A binary adder is a component which can add two n-bit binary numbers. If n is 32, it will consist of 32 full adders, one full adder for each bit in the 32-bit word. Designate the bit positions as 0..31, where position 0 is the low order bit and

[41] See the open source textbook on *Computer Organization* for general methods of deriving logical expressions from truth tables.

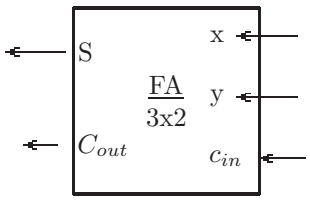


Figure 2.28: Block diagram for a Full Adder. S is the one-bit sum, $x + y + c_{in}$, and C_{out} is the carry-out.

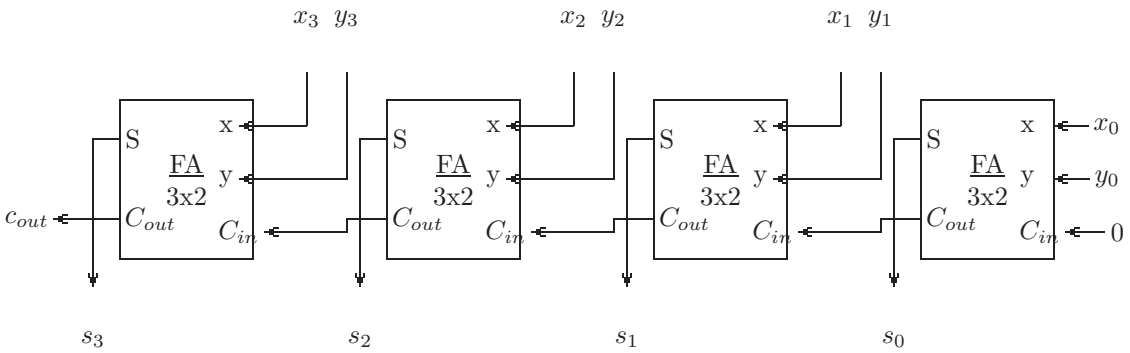


Figure 2.29: Design of a 4-bit adder to find the sum $x+y$, using four full adders

position 31 is the high order bit. Let FA_i designate the full adder at position i. Each full adder will produce one bit of the sum. The S output of FA_i is S_i, the i^{th} bit of the Sum. The Carry out from each full adder will be the Carry in to the next bit position. The C_{out} output of FA_i is the C_{in} input to FA_{i+1}. The C_{out} output of the high-order fll adder is normally discarded.[42]

The logic diagram for a 4-bit adder is shown in Figure 2.29 in which the low order bit (position 0) is at the right, and the high order bit (position 3) is at the left.

Having completed our design of a 4-bit adder, it is easy to visualize the design of a 16-bit adder, 32-bit adder, or an adder for any word size.[43] We show a block diagram for a 32-bit adder in Figure 2.30 which shows the inputs, A and B, and the outputs for the sum A+B, but the internal details are not shown; that is abstraction!

[42] As noted below the carry out from the high order bit can be used to detect an *overflow* condition.

[43] Most personal computers currently use a 64-bit word, and thus have a 64-bit adder.

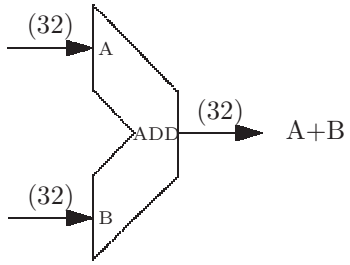


Figure 2.30: Block diagram of a 32-bit adder, to add A + B

In Figure 2.30 we have introduced a few concepts:

- The shape of the adder is not rectangular, unlike other components. Adders are normally drawn like this (but not sometimes they are drawn as rectangles) for historic reasons.
- Rather than showing 32 input lines for A and 32 input lines for B and 32 output lines for the sum, we show a single heavy line in each case, with a label showing the actual number of lines (32) in each case. This is known as a *bus*.

2.10.0.7 Overflow$^{\otimes}$

While we are discussing adders, we should discuss what happens when the magnitude of a sum is to big to fit in a word. This is known as *overflow*, and we can show how to detect overflow. Before discussing the detection of overflow, we should emphasize that the carry out from our adder in Figure 2.29 is *not* an overflow signal. For example, we can add 3 + -2, with the carry bits shown at the top:

```
  111
   0011   =   +3
 + 1110   =   -2
-------
   0001   =   +1
```

There is a carry out of the high order bit, but there is no overflow - the sum is +1, and that easily fits in a 4-bit word. Now consider the addition of 3 + 5, and again we show the carries at the top:

```
   111
   0011    =   +3
 + 0101    =   +5
-------
   1000    =   -8
```

We get an incorrect result, because the true result, +8, is too big to fit in a 4-bit word. In this example, there is overflow, but the carry out of the high order bit is 0.

There are a few ways that the overflow condition can be detected:

- One way to detect the overflow condition is to compare the carry *in* to the high order bit with the carry *out* of the high order bit. If they are *different*, overflow has occurred, and the result is incorrect.

- Another way to detect overflow is to check the signs[44] of the numbers being added. If the signs are equal, the sign of the result should be the same as the signs of the operands:

  - When adding two non-negative numbers, the result must also be non-negative.
  - When adding two negative numbers, the result must also be negative.
  - When adding a negative number to a non-negative number there can never be overflow.

The easiest way to detect overflow in our binary adder is to use an Exclusive-OR gate[45] with two inputs: the carry-in to the high order bit and the carry-out of the high order bit. This could easily be included in our binary adder.

2.10.0.8 Further abstractions with hardware

Using logic gates as fundamental building blocks, we have seen how higher-level components can be constructed. This abstraction process is continued to even higher levels. Logic circuits can be printed onto a circuit board, or embedded in a semiconductor chip, thus producing a higher level component such as:

- Central Processing Unit (CPU)

- Device controller (such as a disk controller)

- Video card (used to display high quality graphic images)

- Radio Frequency Identification Devices (RFID) which can be used to tag and track packages or products for sale.

- Camera components in phones and security devices

[44] The sign of a two's complement number is the high order bit, which determines whether the number is negative. To determine whether the number is positive, one would have to exclude the case where the number is zero.

[45] Exclusive-OR will produce a 1 result only when the inputs are *different*.

2.10.1 Exercises

1. Use a truth table to prove each of the following:
   (a) AND distributes over OR, i.e. $x \cdot (y + z) = x \cdot y + x \cdot z$
   (b) OR distributes over AND, i.e. $x + (y \cdot z) = (x + y) \cdot (x + z)$
   (c) $x \oplus y = x' \cdot y + x \cdot y'$
   (d) Exclusive OR is associative, i.e. $(x \oplus y) \oplus z = x \oplus (y \oplus z)$
   (e) DeMorgan's first law: $(x + y)' = x' \cdot y'$
   (f) DeMorgan's second law: $(x \cdot y)' = x' + y'$
2. Show a digital logic diagram corresponding to each of the following logic expressions:
   (a) $x + y \cdot z$
   (b) $(x + y) \cdot z$
   (c) $(x \oplus y) \cdot y'$
3. Show how to build a full adder without using a half adder. Refer to the truth table in Figure 2.26
   Hint: Use Exclusive OR
4. We are using our 4-bit adder in Figure 2.29 to add two values:
   `x = 0110, y = 1101`
   (a) Copy Figure 2.29 and fill in 0's or 1's on each input arrow (i.e. $x_0, x_1, x_2, x_3, y_0, y_1, y_2, y_3$)
   (b) Fill in 0's and 1's on all the output arrows.
   (c) What is the 4-bit result?
   (d) $^{\otimes}$ Did overflow occur in this example?

2.11 Levels of Abstraction in Models and Simulations

Computers have been programmed to be very effective in understanding natural, or man-made, processes. The computer can be programmed to *simulate* a process, such as the evolution of a biological species. Of course the process itself is separate and possibly very different from the simulation, but the simulation can help us achieve a better understanding of the process. When the computer is programmed to behave like some natural or physical object, we call it a *model* for that object. A model is separate and distinct from the object that it models, but it captures the essential aspects of that object.

As with other kinds of abstractions, models and simulations will normally hide the unimportant details, and reveal the observable behavior of the process being simulated or of the object being modeled. We now present just a few of the important models and simulations that have been developed.

2.11.1 Weather

Everyone is interested in weather forecasts. You've seen them on TV news broadcasts, in newspapers, on the internet. Weather forecasts serve an important function for all of us, but there are some entities for which weather forecasts are vitally important and extremely valuable. Some examples of these entities are in:

- The transportation industry (primarily airplanes, but also land and sea-based transportation)
- The military; not only transportation of troops and material, but also battle strategy and plans
- Agriculture; farmers need to plan ahead for droughts, monsoons, and other unusual weather patterns
- Local government services; first responders and city officials need to plan ahead for snow, tornadoes, hurricanes, etc.

The importance of weather prediction is so great that most nations of the world invest huge amounts of money annually to achieve accuracy. This is generally done using arrays of *supercomputers*.[46] It is possible that more money is spent world-wide, on predicting the weather than on any other single computational problem.[47]

How can a computer predict the weather? Atmospheric conditions can be *simulated* by the computer; this is generally done using large arrays of numbers. Each number represents some atmospheric condition at a point in time and space. Using three-dimensional arrays, each can represent some atmospheric condition for a large volume of the earth's atmosphere. The rows in the array are lattitude positions, the columns are longitude positions, and the third dimension is for altitude. The atmospheric conditions represented by an array could be: Temperature, Air pressure, Humidity, or Wind velocity (speed and direction).

Simulations generally execute a sequence of *steps*, in which each step makes incremental, and minor, changes to the state of the simulation. In the case of atmospheric simulation, each step represents some unit of time, perhaps a few seconds. In each step the current state of the simulation (temperature,

[46] A supercomputer is one which has significantly more memory, can compute at very high speeds, and has a high degree of parallelism (can do many different things at the same time), as compared with ordinary desktop or laptop computers.

[47] With the advent of social media, some may argue that facebook, amazon, google, or twitter now holds that distinction.

pressure, humidity, etc.) is used to determine the atmospheric conditions for the next step, i.e. at a small incremental time in the future.

The condition known as wind velocity is critical here; wind velocity can be used to determine how the other conditions are affected at each step.

Figure 2.31 is a diagram of some small arrays representing temperature and wind velocity at some step in the simulation, and at the next step. The numbers in the Wind Velocity table represent the wind speed (km per hour) and the wind direction:

- 0 = 360 = East →
- 90 = North ↑
- 180 = West ←
- 270 = South ↓

Thus an entry of (6,0) represents a wind velocity of 6 km/hour in a direction directly to the east.

Note that the wind velocities are in a generally eastern direction. We would expect that atmospheric conditions would be moving from west to east. This is verified by noting that the temperatures appear to move to lower longitudes (to the east), when comparing step n with step n+1.

In this manner all the atmospheric conditions can be calculated at each point in time and space, from a prior time. These tables of numbers constitute a *model* for atmospheric conditions, which can be used to predict the weather at some time in the future.

Here we see that a model is not a perfect representation of the object being modeled, but is generally just an estimate. Conclusions which are drawn from the model (in this case weather prediction) can be no more accurate than the model and are often less accurate.

2.11.2 Evolution

In 1859 Charles Darwin, a British scientist, published *The Origin of Species by Means of Natural Selection*. This theory of evolution is now widely accepted as the true means by which humans and other species came to exist on earth.[48] Darwin's theory, in a nutshell, is based on the occurrence of mutation(s) during reproduction. A mutation is a rare change in a species' DNA sequence, caused by radiation or other environmental conditions. In most cases the organism with the modified DNA will be affected in some non-beneficial or harmful way (think of birth defects), and the organism will be unlikely to survive, and even less likely to reproduce. However, in even more rare cases the mutation may result in a beneficial change to the species. For example, at one time apes had paws similar to other mammals. A mutation in their DNA might have, by chance,

[48]This theory was challenged by many, including those who felt it contradicted the Book of Genesis in the Old Testament. Today most religious scholars accept Darwin's theory, but nevertheless point out the metaphorical significance and value of Genesis.

Step n

	Temperature Longitude				Wind Velocity (Speed, direction) Longitude			
Lattitude	87° 50'	87° 49'	87° 48'	87° 47'	87° 50'	87° 49'	87° 48'	87° 47'
43° 20'	20	20	20	21	5,0	6,2	5,1	7,364
43° 19'	20	21	21	22	6,2	5,3	5,1	7,0
43° 18'	19	21	21	21	5,4	6,0	6,1	7,1
43° 17'	19	19	20	20	6,3	6,1	7,0	7,2
43° 16'	18	19	19	20	6,2	6,2	7,1	7,0

Step n+1

	Temperature				Wind Velocity (Speed, direction)			
Lattitude								
43° 20'	19	20	20	20	5,0	6,2	5,1	7,0
43° 19'	19	20	21	21	6,2	5,3	5,1	7,0
43° 18'	18	19	21	21	5,4	6,0	6,1	7,1
43° 17'	18	19	19	20	6,3	6,1	7,0	7,2
43° 16'	18	18	19	19	6,2	6,2	7,1	7,0

Figure 2.31: Representing temperature (degrees Celsius) and wind velocity (km/hour, degrees from East) at a particular altitude, showing the transition from one step to the next step

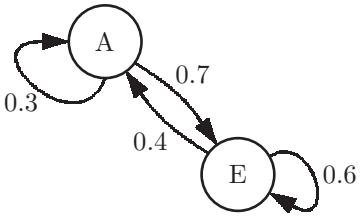


Figure 2.32: A diagram of a Markov Chain model, with two states, and probabilities of change in state.

resulted in an opposable thumb on their front paws, enabling them to grasp tree branches and other objects. These apes were more likely to survive, reproduce, and pass on the trait of opposable thumbs to future generations of apes. This process of random mutations, with certain mutations being passed to future generations is called *natural selection*.

More recently scientists have questioned whether it is possible to model the DNA for a species, using a computer. With appropriately chosen probabilities, the model should enable us to understand what kinds of mutations might occur, and which could effect the evolution of new species.

Most such models involve a *Markov chain*. A Markov chain is a more general model involving transitions from one state to another, with a built-in random factor. Simulations using a Markov chain over long periods of time can provide information on the object or process being modeled. Figure 2.32 shows a diagram of a Markov chain. In this diagram there are two *states*, labeled A and E. As the model executes it can cycle through a series of states:

- If it is in state A, the probability that it will change to state E is 0.7, and the probability that it will remain in state A is 0.3[49]
- If it is in state E, the probability that it will change to state A is 0.4, and the probability that it will remain in state A is 0.6

2.11.3 Warfare

The military forces use simulations to gain a better understanding of battlefield tactics, resource allocation, supply line maintenance, personnel training, and other aspects of military preparedness.

The use of battlefield simulations is often referred to as *war games*, and allows officers to make battlefield decisions without risking lives or material, since the simulation is done entirely with computers. In this way officers can replay the same battlefield situation several times, trying different strategies to see what works best.

These war games can involve land, sea, and air battles, or some combination of all three into one simulation. In addition to battlefield simulation, the military

[49]Probability is a fraction of 1, meaning that a probability of 0.3 represents a 30% chance.

makes extensive use of simulation to train members of the armed forces in the proper use of military equipment (see Flight Simulators below).

Some war games could even be coupled with weather forecasts, as described above, to understand how weather would affect battle strategies and transportation of troops and material.

Military simulations can be divided into two general types:

- *Heuristic*[50] simulations are deterministic simulations which have consistent behavior.
- *Stochastic* simulations involve an element of chance; thus multiple plays of the same situation can have different outcomes. Various aspects of the simulation are determined by the generation of random numbers.

2.11.4 Biological Populations and Environments

Scientists have developed software which model specific kinds of animals and/or plants. Together with models of their environment, a simulation over time can produce information concerning:

- The organisms' migration patterns, as a result of environmental change
- The organisms' likelihood of population growth or decline
- The organisms' likelihood of extinction

A fairly simple, but interesting, example of such a model was introduced by John Conway[51] in 1970. It is known today as *Conway's Game of Life*. This game is played on a grid of any size; imagine a large checkerboard, with rows and columns of squares. Each position is either occupied by an organism or not occupied. The game is actually a simulation, consisting of a series of steps, or generations. It begins with an initial configuration of occupied squares. Each step of the game then proceeds according to the following rules:

1. Any organism which has fewer than two organisms on neighboring squares[52] will not survive to the next generation (it dies from lack of resources, support, loneliness perhaps)
2. Any organism which has more than three organisms on neighboring squares, will not survive to the next generation (it dies from overcrowding, lack of food perhaps).
3. Any organism which has two or three organisms on neighboring squares will survive to the next generation.

[50] A heuristic, as contrasted with an algorithm, is not guaranteed to reach a perfect soluition to a problem. Heuristics are often used when there are no known efficient solutions to a problem.

[51] John Conway was a Mathematics Professor at Princeton University,

[52] On a grid, each square has exaclty eight neighboring squares.

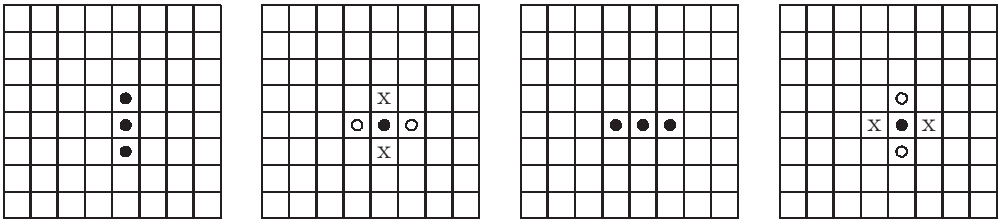


Figure 2.33: Game of Life simulation - a blinker. From left to right showing the simulation from an initial configuration to the first generation: Initial configuration; two deaths(x) and two births(o); first generation; two deaths and two births.

4. A new organism is born on any unoccupied square which has exactly three organisms on neighboring squares.

An example of this simulation is shown in Figure 2.33. In this example the initial configuration consists of three vertical occupied squares (left diagram in the figure). We calculate the next generation as follows:

- According to rule 1 above, the top organism and the bottom organism both die, because they each have only one neighbor.
- According to rule 3, the center organism survives to the next generation because it has two neighbors.
- According to rule 4, there are births in the squares directly to the left and directly to the right of the center square, because they each have organisms in exactly three neightboring squares.

This is shown in the second board from the left. The first generation of the game is then shown in the third board from the left (we assume the initial configuration is generation 0).

To calculate the second generation, we apply the same rules, and see that there are again two births and two deaths. This is shown in the board on the right in Figure 2.33. Thus we see that the second generation is the same as the initial configuration, and future generations will be alternating: three vertical squares and three horizontal squares. We call this initial configuration a *Blinker*; it is an example of a class of initial configurations called *oscillators*. They cycle through two or more configurations, before arriving back at the initial configuration. There are other initial configurations which oscillate; this is left as an exercise.

There are also *stable* configurations, also known as *still life*. An example is the so-called Beehive shown in Figure 2.34. In this configuration each organism has exactly two neighbors, so they all survive to the next generation. Also, no unoccupied squares have organisms in exactly three neighbors, so there are no births. All subsequent generations are exactly the same as the initial configuration.

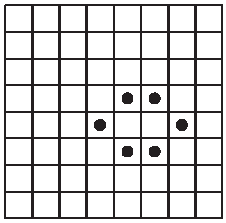


Figure 2.34: Game of Life simulation - a stable configuration, known as a Beehive.

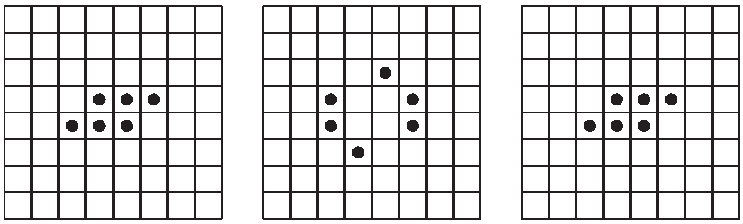


Figure 2.35: Game of Life simulation - initial configuration known as a Toad. Generations 0-2.

A more interesting initial configuration, known as the 'Toad', is shown in Figure 2.35, in which we also show generations 1 and 2. The Toad is an oscillating pattern, similar to the Blinker.

Conway's Game of Life has had some academic interest over the years, but is not generally considered to be an accurate simulation of any real biological populations or environments. We introduce it here mainly to provide the student with an accessible example of a simulation.

2.11.5 Climate

Recently, scientific research on *climate change* has been in the news. There are two different questions which have been raised:

- Has the climate been changing, and if so, how has it been changing? I.e. what are the measurements that indicate consistent change?
- If the climate has been changing, is this change a direct result of human artifacts or human-generated processes?

To answer these questions, scientists take measurements all over the world. These include atmospheric measurements, as well as measurements in the oceans, at all depths, and on land. Of particular interest are measurements taken at the polar ice caps in which an historic record of the climate can be obtained by ice core samples at various depths. Scientists note in particular the land/ocean areas which are no longer covered by ice, and glaciers which have been melting.

Some fairly obvious evidence:

- Anyone flying over Greenland in an airplane can see large areas of land where only 10 years earlier the land had been entirely covered with ice.
- Anyone who has traveled the Ice Highway in Canada can walk on glaciers which have receded far from the highway.
- There are new shipping routes. Ships traveling between China and Europe formerly took the Suez Canal. During the months of May - September, they can now travel above the Arctic Circle, north of Russia and Siberia, for a much shorter route.[53]

Using the measurements taken across the globe, scientists construct a model for the climate, and use the model to attempt to predict the effects of climate change.

- Is the average global temperature increasing? If so, by how much?
- Are sea levels rising? If so which coastal cities will be flooded, and when will this happen?[54]
- Does climate change produce unusual, or potentially harmful, weather patterns such as hurricanes, tornadoes, or monsoons?

Many scientists have used models which include atmospheric conditions over long periods of time. It is claimed that gases in the atmosphere (principally carbon dioxide) affect the composition of sunlight which reaches the earth. These scientists are using models to answer questions about climate change:

- If humans continue to generate carbon compounds in the atmosphere, what will the impact be on climate?
- If humans decrease the amount of carbon being put into the atmosphere, will the climate continue to change anyway?
- Are there other compounds in the atmosphere which affect the climate?
- What kinds of climate change have occurred in the past, and how are they similar to, or different from, climate change that is currently being observed?

2.11.6 Training

Simulation software has played an important role in training, or educating, people to perform certain difficult tasks. Often it is too expensive or dangerous for the trainee to attempt the actual task. In this case the trainee can perform

[53] The shipping companies may see this as a good thing, as it improves their profit margins, whereas climate scientists see the melting polar icecap as a sign of climate change.

[54] Some scientists note that the flooding will occur because of the thermal expansion of the oceans, in addition to the melting polar ice caps.

the task on a computer and observe the consequences of actions and decisions with no danger involved.

One of the most important such training simulations is known as a *flight simulator*. Commercial flight simulators are special-purpose full-size cockpits with realistic panels displaying sensor information (airspeed, groundspeed, direction, altitude, aircraft orientation, etc.). They also have realistic controls: joystick, switches, hydraulic controls, etc. These simulators were first developed in the 1920's and have been used extensively by the military and large airline companies. These simulators are so realistic that would-be pilots can be trained in a matter of days, before attempting to fly (with a co-pilot) an actual aircraft.

More recently personal computers have been programmed to provide a similar experience for the hobbyist. The display panels are on the computer's display. The controls are simulated by the mouse and keyboard.[55] The most popular amateur flight simulator was marketed by Microsoft since 1982 for the IBM PC. In addition to showing cockpit panels, the computer's display also provides a view of what the pilot would see out the window, with a horizon line and runways, when approaching an airport.

2.11.7 Exercises

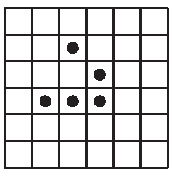
1. Refer to Figure 2.31. Show the values for temperature and wind velocity at step n+2. Your values should be close approximations of the values which would actually occur.

2. DNA consists of strands of *nucleotides*. There are four kinds of nucleotides, labeled C, G, T, A. The Markov Chain shown in Figure 2.36 shows the probabilities that each nucleotide will or will not mutate when the DNA molecule replicates. Using this model, and given the string of nucleotides shown below, assume exactly one mutation occurs. Show a possible string of nucleotides which is most likely to exist after one reproduction.
   `gctcagatcggcattacgct`

3. List the names of several board games, and/or video games, which simulate warfare.

4. Using the rules of Conway's Game of Life, show at least 4 generations for each of the following start configurations:

   (a)



[55] Game joysticks are also used to simulate the real joystick in an aircraft.

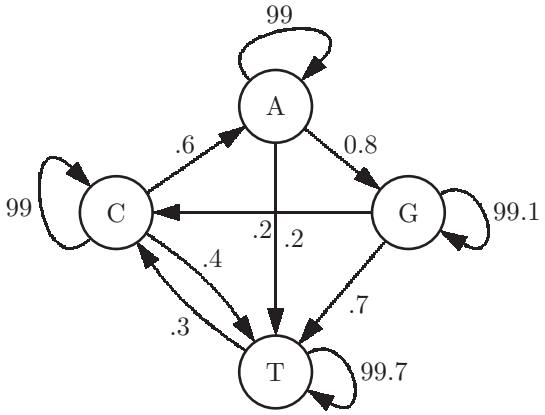
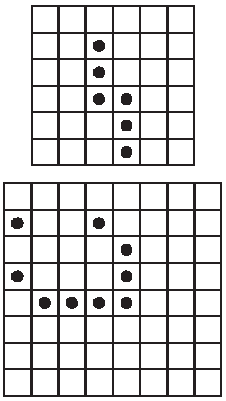


Figure 2.36: A diagram of a Markov Chain model for replication of a DNA molecule

(b)

(c)



5. Build an Excel spreadsheet to simulate Conway's Game of Life.
   Hint: Put each generation on a separate sheet. On each sheet use three n x n grids showing:
   - The configuration, with a '*' in each occupied square
   - The same configuration, substituting 1 for occupied squares, and 0 for unoccupied squares
   - A grid showing the number of occupied neighbors for each square.

6. (a) Search the internet for reliable sources which claim that the earth's climate is undergoing a sudden change.
   (b) Search the internet for reliable sources which claim that the earth's climate is not undergoing a sudden change.
   (c) Search the internet for reliable sources which claim that the earth's sudden change in climate is caused by artificially produced gasses in the atmosphere.

(d) Search the internet for reliable sources which claim that the earth's sudden change in climate is not caused by artificially produced gasses in the atmosphere.

(e) Which of the above sources make use of digital models of the climate?

7. In addition to flight simulators, what other useful kinds of training can be done with simulators?

Chapter 3

Data and Information

In this chapter we examine *data* and *information*. Although these words are often use interchangeably, we make a careful distinction:

- Data might well be thought of as *raw* data. Binary data is merely a sequence of 1's and 0's with no particular meaning or interpretation associated with the sequence.
- Information is data which has been assigned meaning or interpretation. Subsequences of bits can be grouped in a meaningful way, and given a particular interpretatioon which is useful or meaningful in some way.

3.1 Information Processing

Once we have information stored in a computer, in digital form, we can work with that information to gain insight and knowledge. This is known as *information processing*.

3.1.1 Processinig Information to Gain Insight or Knowledge

The primary objective when processing information is to deduce knowledge, or gain insight of some sort, into the domain of the information.

As an example, suppose we have the information shown in Figure 3.1.[1] For each student we have stored their Grade Point Average (GPA) on a scale of 4.0 maximum. We have also stored whether they are a commuter, versus residential student, whether they belong to a fraternity or sorority, and the number of credits completed.

What insight can be gained from this limited amount of data? We will attempt to find a *correlation*, or relationship, between various attributes in the table and the students' GPA.

[1] This data is fictitious, and all conclusions drawn from this data are not necessarily valid.

First we look at whether the student was a member of either a fraternity or a sorority. Of those who belonged to either a fraternity or a sorority, the average GPA is 2.3, on a scale of 4.0 maximum, whereas for those students who belong to neither a fraternity nor a sorority, the average GPA is 3.6. This is a fairly large difference, indicating a correlation between a student's GPA and membership in these social organizations.

Note that a correlation does not imply a cause and effect relationship. We cannot conclude from this analysis that membership in a fraternity or sorority *causes* a student to have a lower GPA. It is possible, for example, that students with poor study habits are attracted to social organizations, or that students with good study habits are not attracted to social organizations. It is also possible that students who are struggling academically seek out social organizations for help with their studies.

Next we look at whether the student is a commuter. With some calculations we see that the average GPA for this group of students is about 2.7. We compare that with the average GPA of those students who are commuters (2.8) with the average GPA of those students who are not commuters (2.6). We might be tempted to see a positive correlation here between GPA and being a commuter, but the difference, only one tenth on a scale of 4, is not large enough to be statistically significant.[2]

Finally we examine the number of credits completed by the students. To do this we divide the students into two groups: those who have completed more than 40 credits, and those who have completed 40 credits or fewer.[3] We see that the average GPA of those students who have completed more than 40 credits is 3.2, and the average GPA of those students who have completed 40 credits or fewer is 2.4. A statistician is likely to consider this difference statistically significan, but would call it a *weak correlation* between GPA and credits completed.

In this example of information processing we have gained some limited insight into aspects of college life which may or may not be correlated with a high GPA.

3.1.2 Collaboration

The example in the previous section consisted of only 12 students. This is a small number of cases, and is certainly too small to reach any real conclusions. It would certainly be better to use actual data from a university, consisting of thousands of students. The reliability of the conclusions is a direct result of the number of cases (students) for which we have data.

Even better would be the amalgamation of data from several universities. This could conceivably lead us to gain insight in the practice of fraternities and

[2]Statistical significance is beyond the scope of this course, and is covered in most introductory statistics courses.

[3]Statisticians have developed procedures for determining correlations without grouping the cases as we are doing here; this is also beyond the scope of this course.

Name	GPA	Commuter?	Fraternity/ Sorority	Credits Completed
al	2.4	Y	Fraternity	12
bart	1.4	N	Fraternity	60
beth	3.4	Y		45
chas	3.7	Y		90
jen	2.0	Y	Sorority	25
jim	2.0	N	Fraternity	22
joe	3.6	N		90
mary	3.8	Y		25
mike	1.9	N	Fraternity	25
sue	2.2	Y	Sorority	30
susie	3.9	N	Sorority	60
tom	2.4	Y	Fraternity	30

Figure 3.1: Some information for several college students

sororities at various institutions. It would certainly improve the reliability of our conclusions because of the large number of students.

In order for this kind of collaboration to occur, it is often necessary for information to be 'massaged' in some way. For example, universities which have residential social organizations probably should not be compared with universities that have non-residential social organizations. Also, the maximum GPA at a university can vary; it may be 4.0 at some universities, but 5.0 at other universities. In this case the GPA calculation should be, perhaps, a fraction of the maximum: a GPA of 4.2/5.0 is lower than a GPA of 3.8/4.0, for example.

In general, collaboration allows for:

- An increased quantity of data, resulting in more information and greater reliability of insight, knowledge, and conclusions
- An increased number of ways that the information can be processed, as various researchers offer their own processing schemes
- A way of cross-checking the validity of information processing schemes and conclusions, as various researchers examine the work of other researchers

3.1.3 Explanation with Visualization or Notation

In the example given above we showed how information processing allows us to gain some insight or understanding of college life and students' GPA. However, often the insight is more clear if presented in a non-numeric form, such as graphical charts or pictures. This kind of display is usually described as *information visualization*. Information can take many forms: bar charts, line charts, scatter charts, pie charts, and other forms.

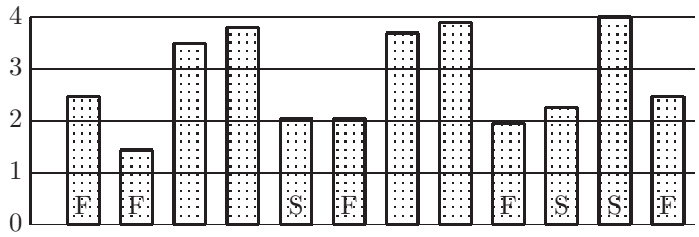


Figure 3.2: A bar chart showing student GPAs and social organizations, F = Fraternity, S = Sorority (refer to Figure 3.1)

Name	GPA	Commuter?	Fraternity/ Sorority	Credits Completed	Age
al	2.4	Y	Fraternity	12	18
bart	1.4	N	Fraternity	60	18
beth	3.4	Y		45	48
chas	3.7	Y		90	20
jen	2.0	Y	Sorority	25	19
jim	2.0	N	Fraternity	22	25
joe	3.6	N		90	20
mary	3.8	Y		25	18
mike	1.9	N	Fraternity	25	19
sue	2.2	Y	Sorority	30	21
susie	3.9	N	Sorority	60	18
tim	2.4	Y		32	52
tom	2.4	Y	Fraternity	30	18

Figure 3.3: Some information for several college students, including age

In Figure 3.2 we show an example of information visualization; it is a bar chart of our students' GPAs, but it also shows which students belong to fraternities, which belong to sororities, and which do not belong to any of these social organizations.

The bar chart makes it visually clear that there is an apparent correlation between GPA and membership in social organizations.[4]

3.1.4 Exercises

1. The information on students has been expanded to include the students' ages, as shown in Figure 3.3.

   (a) Is there a correlation between GPA and the student's age? If so, is it a positive or negative correlation?[5]

[4]The sorority member who has a GPA of 3.9 does not contradict the correlation; this data point is called an *outlier*.

[5]A positive correlation would mean that students with a greater age generally have a

(b) Is there a correlation betwwen student's age and membership in a social organization?

2. (a) Show a chart which supports your solution to the previous exercise on the correlation between age and GPA.
Hint: Use a *scatter* chart in which the X axis is GPA, and the Y axis is the student's age. Each student will appear as a point on the XY plot.

(b) Show a chart which supports your solution to the previous exercise on the correlation between a student's age and and membership in social organizations.

3.2 Information: Exploration and Discovery

Since the advent of the Internet in 1982 exploration and discovery has become commonplace. Search engines offered by corporations such as Google and Yahoo are capable of visiting a huge number of web sites in a short period of time to find information relevant to a particular search problem.

3.2.1 Extracting Information from Large Datasets

Today there are numerous examples of large datasets, from which useful information can be extracted. This is not a simple and straightforward task. Because of the large quantity of data available, efficient methods are required. Some examples of large datasets containing useful information are shown below.

3.2.1.1 Cable TV

When you use a remote control device to change TV channels, search for a particular program, mute the sound, or browse what is available, all of your actions are recorded by the cable TV provider. Multiply these actions by millions of viewers, and the provider has a huge dataset from which valuable information can be obtained. From this huge dataset the cable TV company can extract the following:

- How many people are watching each program at a given time
- How many people are watching each advertisement, with or without sound
- When are people most likely to have the TV on
- What are the trends in switching from one program to another

greater GPA. A negative correlation would mean that students with a greater age generally have a lower GPA.

- What frequency and length of advertisements cause people to switch channels, or turn off the TV
- Which advertisements are most/least objectionable
- For what kinds of programmatic material are people most often searching

3.2.1.2 Telephone use - communication

When you use your phone to communicate with voice or text, this is recorded by the provider. The provider has a dataset known as *metadata*. This is not the actual content of the voice or data communications, but it includes the source, the target(s) being called, the date and time, the duration of the call.

From this huge dataset the provider can extract useful or valuable information:

- When are the most common days/times of the week for phone communication
- In what geographic locations are most phone users likely to need service
- What resources are needed to accommodate voice versus text
- Who has communicated with terrorist, or other illegal, organizations or people[6]

3.2.1.3 Cellular phone use - tracking

Smart phones, fitness wathches, and other digital devices make use of the Global Positioning System (GPS) to find the device's position on the surface of the earth (lattitude, longitude, and altitude). This is one of the most sought-after features for individual users, as well as vehicles such as automobiles, trucks, airplanes, and ocean-going ships. The device with GPS sensors and software receives signals from three or more satellites, which are used to determine position.[7]

The provider of these services has access to this data, for potentially millions of users, and can extract useful and valuable information from this huge dataset:

- Where are traffic jams occurring on the roadways
- What routes are people likely to take from one location to another
- Where are people likely to stop when traveling from one location to another
- When will traffic volume be at its highest

[6] Legal questions have been raised when law enforcement officials have requested access to metadata, as well as access to the data on an individual's phone.

[7] See the Oct 2020 issue of the *Communications of the ACM* for an article on privacy with respect to cellular phones.

- Where do people tend to exceed the speed limit, and by how much
- How can criminals at large be tracked and apprehended[8]

3.2.1.4 Social media

Popular applications accessible on the web or on smart phones include Facebook, Twitter, Instagram, LinkedIn, and other so-called *social media*. As opposed to email, which is a one-to-one or one-to-many form of communication, social media allow for many-to-many forms of communication. Much of the communication provided by these social media could be completed using email; however, the convenience and power of the social media has won over many users.

A social medium such as Facebook establishes direct connectivity among users all over the globe. By indicating a 'like' on Facebook, one is instantly connected with someone else, who in turn is connected to others. This extensive connectivity among users has been an attraction for some, but a negatvie feature for others who fear the potential consequences of excessive connectivity.[9]

All this data is available to the provider (Facebook, Twitter, etc). The provider is able to extract useful or valuable information from this huge dataset:

- At what times are people most likely to be accessing social media
- Who is 'liking' various retailers or vendors
- Who is registered to receive an individual's Tweets
- Analysis of various job markets, via LinkedIn
- Data can be used to impact elections and politics[10]

3.2.1.5 Purchasing

Purchasing habits, preferences, and trends constitute extremely valuable information for vendors. The Amazon corporation began as a distributor of books, but later distibuted everything from appliances to clothing, electronics, and vehicles.

Amazon's business model entails *distribution* and *storage* only, with no manufacturing involved. However, because it is directly connected to the manufacturers and because it has extensive data on inventories and customer preferences, with a strong on-line presence, it has been able to grow into one of the leading enterprises in the world today.

[8] Legal questions have been raised here as well.

[9] Social media are prime targets for the spread of digital viruses and other *malware*.

[10] In the 2016 presidential election, it was later discovered that Cambridge Analytica used Facebook to promulgate false information on candidates for office in the United States. In light of this scandal, it is a mystery to this author why supporters of Hillary Clinton continue to use Facebook.

3.2.2 Data analytics$^{\otimes}$

Also known as *data science*[11], data analytics is the systematic computational analysis of data or statistics. A data analyst, or data scientist, develops and/or uses software designed to search large quantities of raw data for patterns or information leading to knowledge of a targeted content area. Data analytics includes *web analytics*, in which the search is restricted to the world-wide-web.

For example, a communications provider such as Comcast or Verizon may search a database of television viewer selections which include the following:

- A viewer switches their TV on or off
- A viewer changes the channel selection
- A viewer mutes, or adjusts the volume
- A viewer searches for particular programming
- A viewer records content

Using this raw data, the provider can use data analytics to provide potential advertisers with information promoting the value of program sponsorship.

3.2.3 Machine learning$^{\otimes}$

Machine learning is the area of study in which we build software that automatically improves in some way as a result of its own experience. Machine learning is considered a specialty area of artificial intelligence.

A company such as Netflix uses software to predict a user's preferences in streamed video content. Initially the software may not provide the user with good suggestions, but over time the program 'learns' better ways of predicting preferences, and ultimately evolves into a system which successfully predicts user preferences.

In the future we are likely to see increased use of machine learning in medical diagnoses, either as an aid to human physicians, or eventually in place of human physicians.

In the area of machine learning there have been more failures than successes, but this remains a widely pursued area of research.

3.2.4 Exercises

1. Explain how each of the following can extract useful or valuable information from large datasets:
   (a) Banks and other financial institutions such as inssurance companies
   (b) Hospitals or other healthcare corporations

[11] More accurately, data science is to chemistry as data analytics is to chemical engineering

(c) Government agencies, such as the Internal Revenue Service, Social Security Administration, etc.

(d) Military organizations

(e) Airlines and other transportation organizations such as railroad and bus companies.

2. How are data analytics used in professional sports?

3. What are some of the advantages and disadvantages of using machine learning in medical diagnoses?

3.3 Digital Data

3.3.1 Time and Space Efficiency

When data is stored in a digital format, there are various forms and trade-offs possible. For small datasets, this is not an issue; we simply store or transmit data as a sequence of 0's and 1's. However, for extremely large datasets, efficiency in both time and space is critical. When storing digital data such as music or video, we must be aware of the fact that billions, or trillions, of bits are involved in the storage of one music or video clip. Even more serious is the transmission of this information from one device to another, or even from one part of a device to another (for example, from a magnetic disk, or solid-state storage, to a computer's memory). In this case it is not the storage space with which we are concerned, but the time that is taken for the transmission to occur.

For these reasons, various *data compression* techniques have been developed. There are two main categories of data compression: *lossy* and *lossless*. Lossy is not perfect; when the data is compressed and then decompressed to its original format, there may be a small number of bits which are not correct. With lossless compression, no information is lost; when compressing and decompressing data, the result is exactly the same as the original data that was compressed. Lossy compression results in greater savings, with an approximation of the original data. Lossless compression produces the original data. In either case space is saved by storing the compressed data, and transmission time is reduced by transmitting the compressed data. More details on compression algorithms was presented in Chapter 2.

3.3.2 Security and Privacy

Much sensitive information is stored on digital devices. This could include:

- Financial information such as bank account numbers. If this information were to fall into the wrong hands the owner could be a target of *digital theft*.

- Identity information such as social security numbers and personal information used for authentication.[12] The compromising of identity information is known as *identity theft*. Once an individual's identity has been stolen, the thief can then pose as the victim to invoke financial transactions for the purpose of digital theft.

- Various entities may have an interest in discovering an individual's web browsing habits for various reasons, including, but not limited to, financial reasons. Many users consider this an invasion of personal privacy.

To protect personal information and privacy, software is being developed to authenticate users and encrypt data. For example, speech recognition software is now available to authenticate a bank customer's voice on the phone.

Most computer operating systems now have the capability of *encrypting* all data on permanent storage. Encryption is the process of scrambling the bits of a data segment in such a way that the bits can be unscrambled only by an authorised user. This encryption and decryption process now takes place automatically, in such a way that the user does not know it is taking place; we say that the encryption is *seamless*. An unauthorised intruder, however, would not be able to retrieve any meaningful information from the encrypted data in permanent storage.

3.3.3 Access to Data

Care must be taken when providing access to data for many users. If the users are given *read* access, they can examine the data but cannot make updates (changes) to the data. If the users are given *write* access to data, they are permitted to make updates. However, in this case only one user may have write access at a particular time to avoid contradictions. A contradiction would result in the following scenario:

1. Joe opens a data file with write access. He is working with his own copy of the file.

2. Mary opens the same data file with write access. She is working with her own copy of the file.

3. Joe changes the number 100 to -100, saves the data file, and closes it.

4. Mary does not see Joe's change to the data. She changes the number 200 to -200, saves the data file, and closes it.

5. Joe opens the data with read access and sees a contradiction: the value that he had set to -100 is now 100. Joe's edits have been lost.

To avoid contradictions only one user should be able to open a file with write access at any one time.

[12] Authentication is the process of establishing the true identity of a person or other entity.

This is a major concern in any *distributed*[13] reservation system, such as an airline reservation system. At any one time multiple airline agents may attempt to reserve a seat on a given flight. The system will queue the requests for a seat and handle one at a time to avoid over-booking a flight.[14]

3.3.4 Exercises

1. Show how the following sequence of data values can be compressed to save storage space and transmission time:
   `9805743121 9805743122 9805743121 9805743120 9805743121 9805743119`
   `9805743125 9805743121 9805743122 9805743112 9805743123 9805743123`

2. When you login to your bank's web site to check the status of your checking account, the bank needs to ensure that you are who you claim to be. This is called *authentication*. List several techniques that can be used for authentication.

3. List some distributed systems, other than airline reservation systems, which must ensure that contradictions do not occur.

[13] A distributed system is one which is used simultaneously by several users on a network.
[14] Airlines often over-book intentionally, but this is a different matter altogether.

Chapter 4

Algorithms

An *algorithm* is a precise sequence of instructions, or steps, which can be used to solve a given problem. The algorithm must eventually terminate with a correct solution to the problem. In this chapter we expose the notion of algorithm without concrete implementations that can be executed; we will discuss *programming* in chapter 5.

An algorithm may be designed, or described, in a fairly abstract way, as long as it clearly specifies the steps that are required to solve the given problem. The algorithm can be *implemented* by writing a computer program in any programming language, or by building a digital device to implement the algorithm. We emphasize the distinction between the algorithm and a particular implementation of the algorithm.

In this book we offer two ways of describing an algorithm:[1]

- We will describe algorithms using a *Text* language, which is amenable to text-only document systems, such as Notepad, Textedit, etc.

- We will describe algorithms using a *Block* language, which uses non-textual graphic structures to describe an algorithm. The block language is easier to read, particularly for non-programmers. However, it is not easily generated in machine-readable form.

4.1 Algorithm design and implementation

In any algorithm there are three fundamental building blocks, known as *control structures*. These are *sequence*, *selection*, and *iteration*. Using only these building blocks, complex algorithms can be defined.

[1] These algorithm description languages are used in the Computer Science Principles AP exam.

Figure 4.1: An assignment operation in both text language (left), and block language (right).

4.1.1 Variables, assignments, and the sequence control structure

Sequence is the most fundamental, and simplest, control structure in an algorithm. It means that the steps can be listed sequentially. When the algorithm is executed, the steps are executed one at a time in the order in which they are listed in the algorithm. [2]

In our examples, we must assume certain primitive operations and storage are available, in addition to the control structures. These are:

- A *variable* may store a number as its value. For example the variable x may store the value 23. We denote this as $x \leftarrow 23$. We would read this as 'x gets 23' or 'x is assigned 23'. We would *not* read this as 'x equals 23'. This operation is depicted in both the text language and the block language in Figure 4.1.

  A variable may consist of more than one letter, are case sensitive, and may even include numeric characters. Examples of variables are: `sum`, `result`, `result32`, `finalResult` `Result`.

- Move the value of a variable to another variable. This is called an assignment operation. We denote this as $x \leftarrow y$ and read this as 'x gets y'. Note that the former value of x, if it had one is replaced by the value of y. We say that this operation *clobbers* the variable x.[3]

- Addition, subtraction, multiplication, and division of numbers are denoted with operators: `+, -, *, /`, respectively. To the right of the arrow in an assignment one may have an arithmetic *expression*, composed of these operations, variables, constants, and parentheses. For example: $x \leftarrow (a + b) * 3$.

As an example of an algorithm which uses only the sequence control structure we define an algorithm to find the sum of 5 given integers, stored in the variables a, b, c, d, e. This algorithm is shown in Figure 4.2.[4]

[2]The Computer Science Principles AP course description refers to the sequence control structure as a *block*, not to be confused with the Block algorithm description language.

[3]Many students forget that $x \leftarrow y$ changes x but does not change y.

[4]The algorithm in Figure 4.2 could have been done with a single assignment statment, but we wish to expose the *sequence* control structure. Also this algorithm serves as a lead-in to the discussion of the iteration control structure, below.

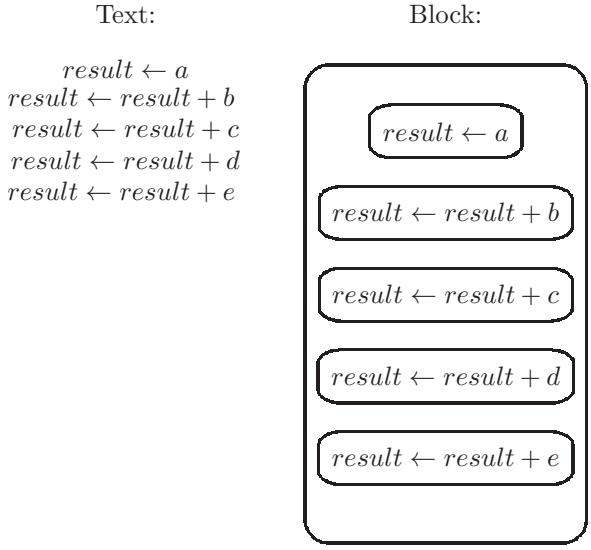


Figure 4.2: An algorithm to sum the values of the variables a,b,c,d,e using the sequence control structure, showing the Text version on the left and the Block version on the right.

4.1.2 Boolean expressions and selections

The *selection* control structure[5] allows us to choose one of two possible options in any step of an algorithm. But before describing the selection control structure we must introduce the concept of a boolean expression.

4.1.2.1 Boolean expressions

A *boolean expression*[6] is an expression which has a true/false value. More formally, if .be. represents a boolean expression, then a boolean expression may be defined as any one of the following:

1. A comparison of arithmetic (or boolean) expressions
2. .be. OR .be.
3. .be. AND .be.
4. NOT .be.

These boolean operations can be formally defined by a *truth table*, as shown in Figure 4.3.

An example of a boolean expression is shown in Figure 4.4.

[5]The Computer Science Principles AP course description refers to the selection control structure as a *conditional*.

[6]Named for the British logician George Boole

x	y	x OR y	x AND y	NOT x
false	false	false	false	true
false	true	true	false	true
true	false	true	false	false
true	true	true	true	false

Figure 4.3: Truth table showing three boolean operations

Text:

$$b < c \; AND \; (a = 3 \; OR \; a > b)$$

Block:

Figure 4.4: Example of a boolean expression, showing the Text version (top) and the Block version (bottom).

To form a comparison of arithmetic expressions we may use any of the six operators shown here: $= < \leq > \geq \neq$

A selection structure will evaluate a boolean expression, and based on the result, select a block of statements to be executed. There are two kinds of selection structures:

4.1.2.2 One-way selection

A *one-way selection* will select a given block to be executed only if the boolean expression is true. A flow diagram for a one-way selection is shown in Figure 4.5

A one-way selection structure is included in an algorithm using the word IF. An example of a one-way selection, in both text and block formats is shown in Figure 4.6. In that example, the assignments to the variables `b` and `x` are made only if the value of the variable `a` is 0.

4.1.2.3 Two-way selection

A *two-way selection* will select one of two given blocks; if the boolean expression is true, execute the first block; if the boolean expression is false, execute the second block. Figure 4.7 is a diagram which represents a two-way selection structure.

An example of a two-way selection is shown in Figure 4.8. In this example the variable `a` is assigned a value only if the current value of the variable `a` is greater than the value of the variable `b`, and the variables `b` and `x` are assigned values only if the value of the variable `a` is less than or equal to the current value of the variable `b`.

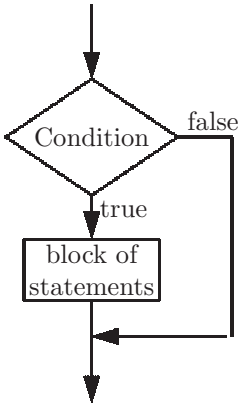


Figure 4.5: Flow diagram for a one-way selection structure

Text:

```
IF (a=0)
   {
        b = 3;
        x = b*17;
   }
```

Block:

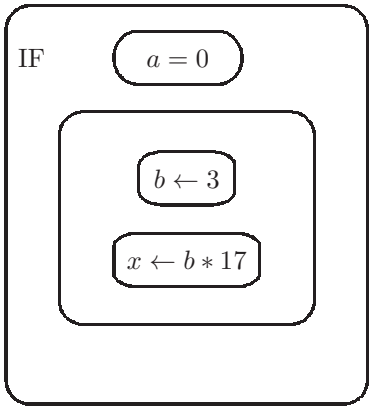


Figure 4.6: Example of a one-way selection structure, showing the Text version (upper left) and the Block version (lower right).

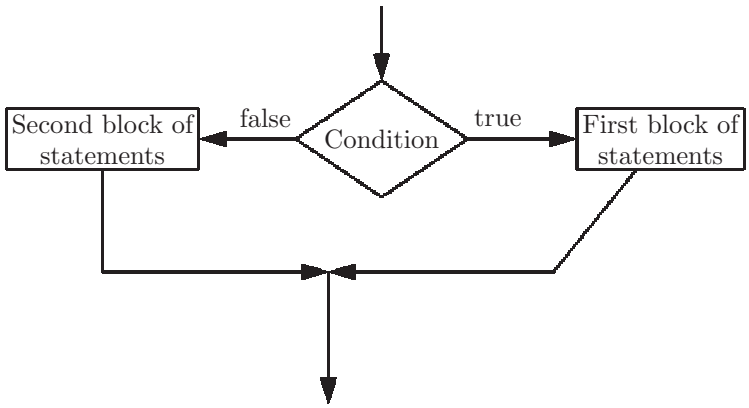


Figure 4.7: Flow diagram for a two-way selection structure

Text:

```
IF (a>b)
        a = b*3;
ELSE
   {
        b = 0;
        x = 0;
   }
```

Block:

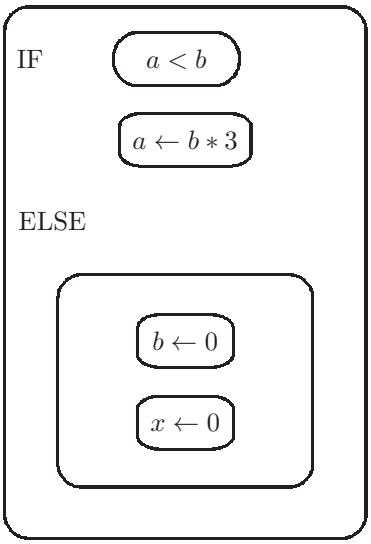


Figure 4.8: Example of a two-way selection structure, showing the Text version (upper left) and the Block version (lower right).

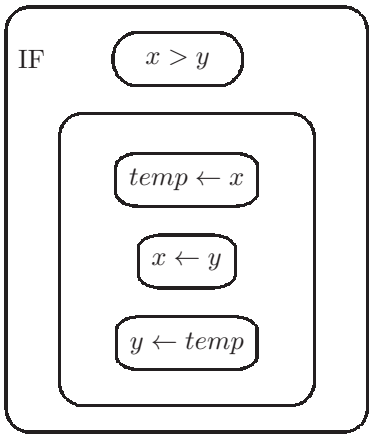


Figure 4.9: Example of an algorithm to order the variables x and y so that the smaller value is stored in x and the larger value is stored in y.

A one-way selection has no ELSE part; this is the only significant difference between a one-way selection and a two-way selection. [7]

As an example of an algorithm, we pose the following problem: Given two variables with values, x and y, terminate with the smaller value stored in x, and the larger value stored in y. This algorithm will need to use a third variable, we'll call it *temp*. An algorithm to solve this problem uses a one-way selection, and is shown in Text format below:

IF $(x > y)$
{
$temp \leftarrow x$
$x \leftarrow y$
$y \leftarrow temp$
}

This algorithm is shown in block format in Figure 4.9. In this example, nothing need be done in the case where $x \leq y$, thus there is no `ELSE` part, i.e. a one-way selection is sufficent.

4.1.2.4 Equivalent algorithms

When developing algorithms, it is often the case that algorithms which are very different in appearance provide solutions to the same problem. We say these algorithms are *equivalent* if they have the same result and the same side effects.[8]

[7] A block of statements in a selection may be a single statement (as in Figure 4.8), though many educators, and the Computer Science Principles course description, tend to use a block even if it consists of just one statement.

[8] Explicit results and side effects will be discussed below.

As an example consider the boolean expression:
`x > y AND b = 0`
We could use this in a conditional statement in two ways:

- ```
  IF (x > y AND b = 0)
      <Consequent Statement or Block>
  ```

- ```
  IF (x > y )
     IF (b = 0)
        <Consequent Statement or Block>
  ```

In either case the consequent statement will be executed only when both comparisons are true.

4.1.3 Iteration

An *iteration* control structure specifies the repetition of a block.[9] We must take care to ensure that the block is executed the correct number of times. There are a few ways of specifying a repetition in an algorithm. Here are some methods for controlling the number of repetitions:

4.1.3.1 Specify the number of iterations

In an algorithm we can specify the number of times that a block[10] is to be executed:
REPEAT n TIMES
where n may be any arithmetic expression. The expression is evaluated, and that is the number of times the block is executed. For example, an algorithm which calculates 2^{12} is shown in Figure 4.10.

4.1.3.2 Specify a termination condition

We can also control an iteration by specifying a termination condition. This will be a boolean expression, evaluating to either true or false. If the boolean expression is false, the block is repeated; if the boolean expression is true, the block is not repeated, i.e. the iteration terminates. The form of the iteration is:
REPEAT UNTIL $<$ *boolean expression* $>$

We can describe the same algorithm, to calculate 2^{12}, using a termination condition, as shown in Figure 4.11.

Care must be taken when using a termination condition; if the condition never becomes true, the body is repeated indefinitely, and never terminates.[11] This would not be an algorithm, since algorithms must terminate.

[9] An iteration is often called a *loop*.
[10] the block which is repeated is often called the *loop body*.
[11] This is known as an *infinite loop*.

Text:

$result \leftarrow 1$
REPEAT 12 TIMES
{
    $result \leftarrow result * 2$
}

Block:

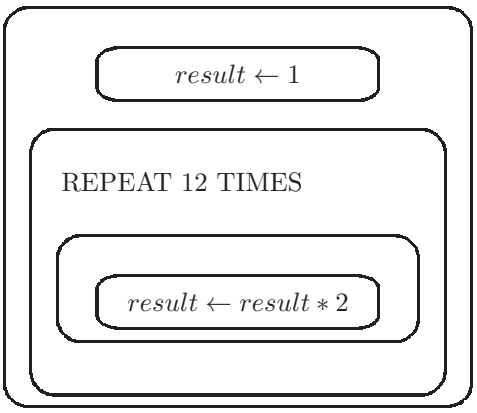


Figure 4.10: An algorithm to calculate 2^{12}

Text:

$result \leftarrow 1$
$counter \leftarrow 12$
REPEAT UNTIL counter = 0
{
    $result \leftarrow result * 2$
    $counter \leftarrow counter - 1$
}

Block:

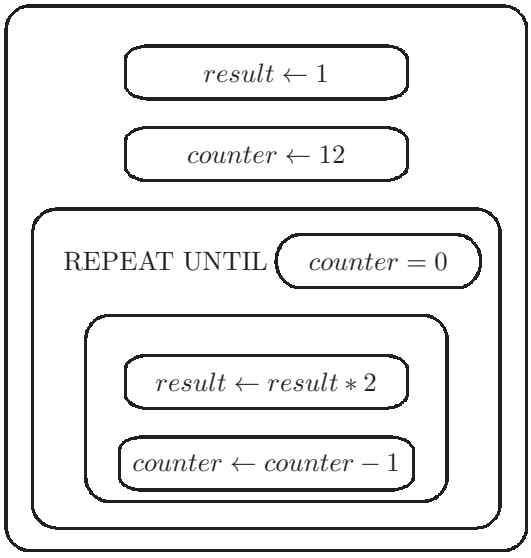


Figure 4.11: An algorithm to calculate 2^{12}, with a termination condition

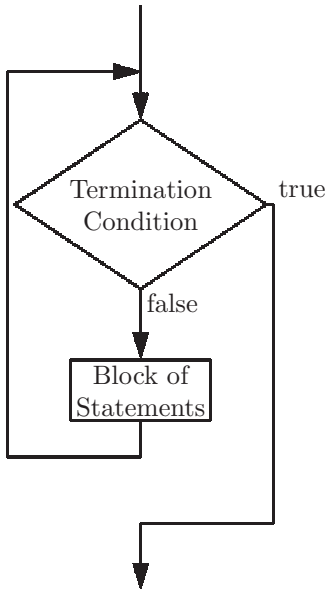


Figure 4.12: Flow diagram for an iteration structure

Text: Block:

$myList \leftarrow [-3, 12, 0, 12]$ $myList \leftarrow \boxed{-3, 12, 0, 12}$

Figure 4.13: A list of numbers; Text version on the left and Block version on the right.

Note that the algorithm given in Figure 4.10 is equivalent to the algorithm given in Figure 4.11; they have the same result.

Figure 4.12 is a diagram representing an iteration structure in which a logical condition is used to determine whether the steps are to be repeated.

4.1.4 Lists

A *list* is an aggregation of several data values into one component, which may be assigned to a variable. In most of our examples these data values will be numbers, but they may be alphabetic characters, or other kinds of data. An example of a list consisting of the data values -3,12,0,12 is shown in Figure 4.13.

Note that a list of numbers may contain negative numbers, and any list may contain duplicate values, unless otherwise stipulated.

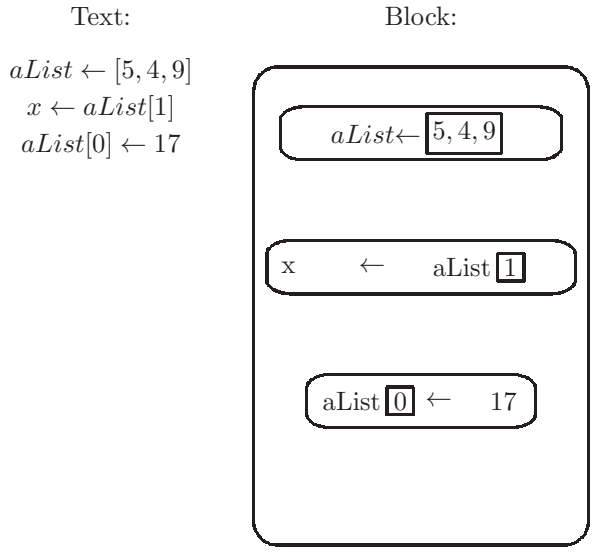


Figure 4.14: An algorithm which selects values from a List. At termination, x is 4, and $aList$ is [17,4,9].

4.1.4.1 List selection

We can *select* a single value from a List of values by giving its position, or *index*, in the List. The position of the first value is always 0; if the size of the List is n, the position of the last value is n-1. In our Text format for algorithms, List selection is specified by placing the index in square brackets. Thus, myList[3] refers to position 3 (actually the fourth position) in myList. In Block format, we show the index in a rectangle. To change a value in a List we can place the List selection to the left of the assignment arrow. Figure 4.14 shows a List of three numbers, the selection of the value at position 1, and a change to the value at position 0.

Note that the index can be any expression which evaluates to a valid position in the List.

4.1.4.2 Other List operations

Four operations are designed to be used specifically with Lists.

- The INSERT operation will insert a value into a List, increasing the size of the List by one. When applying this operation, one must give the name of the List, the position at which a value is to be inserted, and the value to be inserted, in that order, as shown in Figure 4.15.
- The APPEND operation will add a value at the end of a List.[12] One

[12] APPEND is equivalent to an INSERT after the last position in a List.

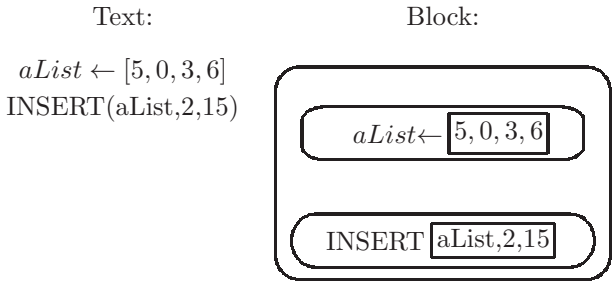


Figure 4.15: Inserting the value 15 at position 2 of the List [5,0,3,6]. After the insertion, $aList$ is [5,0,15,3,6].

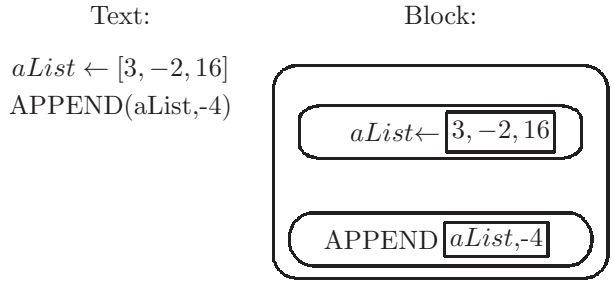


Figure 4.16: Appending the value -4 to the List [3,-2,16]. After the append operation, $aList$ is [3,-2,16,-4].

must give the name of the list and the value to be added, as shown in Figure 4.16.

- The REMOVE operation will remove the value at a given position from a given List. After the operation is complete, the size of the List will be reduced by one. It is an error to REMOVE from an empty List, i.e. a List which has no values. One must give the name of the list and the position of the value to be removed, as shown in Figure 4.17.

- The LENGTH operation returns the number of values in a List. The previous three operations change the List (this is a side effect), but have no explicit result. The LENGTH operation has no side effects, but does have an explicit result, which may then be assigned to a variable, as shown in Figure 4.18. Note that the result of the LENGTH operation may be used in an arithmetic expression. For example,
  $doubleLen \leftarrow 2 * Length(aList)$

4.1.4.3 Iterating through a List

Previously we showed two ways of creating an iteration control structure. We now show another iteration control structure, specifically designed for Lists. It

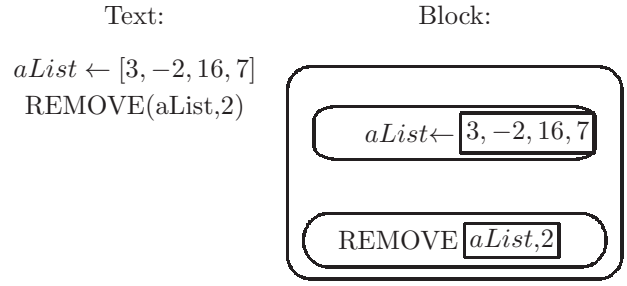


Figure 4.17: Removing the value at position 2 from the List [3,-2,16,7]. After the remove operation, $aList$ is [3,-2,7].

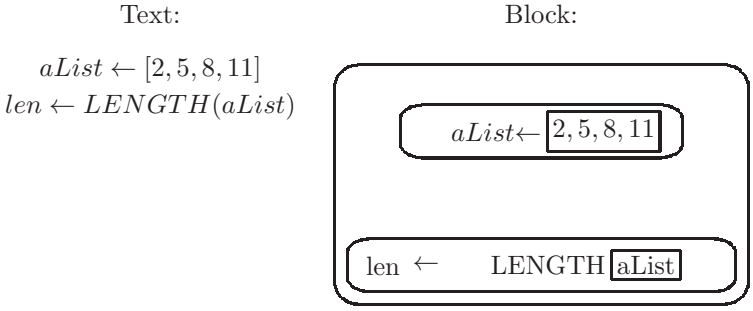


Figure 4.18: Finding the length of the List2,5,8,11. After the LENGTH operation, the value of len is 4.

Text:

```
FOR EACH variable IN list
{
     block of
    statements
}
```

Block:

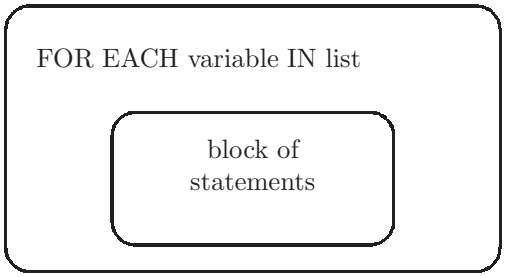


Figure 4.19: General format of a FOR EACH structure

is called a FOR EACH iteration. One specifies the name of a List, and the name of a variable. The iteration cycles through the values in the List, assigning each one to the specified variable on each iteration. Figure 4.19 shows the general format for a FOR EACH iteration.

As an example we present an algorithm to find the sum of the numbers in a List of numbers. It uses a FOR EACH ietration to cycle through the List, storing each number in the variable *value*, and accumulating the sum. It is shown in Figure 4.20.

4.1.5 Nested control structures

Any step within a block can itself initiate one of the three control structures - sequence, selection, and iteration. Thus, these control structures may be *nested*, one within another.[13] More formally, control structures are *recursive* structures. We will see examples of nested control structures below.

As an example of nested control structures we will present an algorithm which finds the smallest value in a given List.[14] The algorithm will choose the first value in the list as the presumed smallest, and store it in a variable, e.g. *smallest*. Then it will iterate through the remaining values, comparing each with *smallest*. If a value is smaller than *smallest*, *smallest* is changed to that value. The variable *smallest* is always storing the smallest value seen *thus far* as we iterate through the List. When the algorithm terminates, the variable *smallest* is storing the smallest value in the List. The algorithm is shown in Figure 4.21.

Note that we can select position 0 of the List at the beginning of the algorithm because we know the List is not empty. In this algorithm we have nested a conditional control structure within an iteration.

The problem which this algorithm solves is an example of an *extremum* problem, which is a more general class of problems involving a search for the largest or smallest in some collection.

[13] We see no apparent need to nest a sequence block within a sequence block.

[14] In this algorithm we assume the List is not empty.

Text:

$numbers \leftarrow [5, 3, 99, 3, -3]$
$sum \leftarrow 0$
FOR EACH num IN numbers
{
  $sum \leftarrow sum + num$
}

Block:

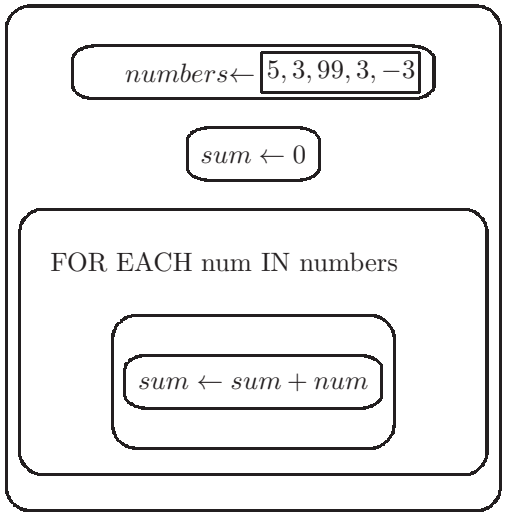


Figure 4.20: Algorithm to find the sum of the numbers in the List [5,3,99,3,-3]. At termination the value of the variable *sum* should be 107.

4.1.6 Abstraction of Algorithms: Procedures

Recalling from chapter 2 that *abstraction* is the process of hiding the details of an artifact so that only the essential parts are visible to a user of the artifact, we now apply abstraction to algorithms. We will do this by introducing the concept of a *procedure*. A procedure encapsulates an algorithm for use by other algorithms. There are two aspects to procedures:

- The procedure *definition* consists of:
  - A name for the procedure
  - Zero or more inputs to the procedure, known as *parameters*[15]
  - A block of statments known as the procedure *body*
  - An explicit result, or *return* value, which is optional

  The format of a procedure definition is shown in Figure 4.22.[16]
- The procedure can be *called* or *invoked* from another procedure by providing:
  - The name of the procedure being called

[15] What we are calling parameters are sometimes called *formal* parameters

[16] This notation is derived from the mathematical notation for a function, though an algorithm need not have an explicit result, and may have 'side effects'.

Text:

$numbers \leftarrow [5, -3, -99, 2, 3]$
$smallest \leftarrow numbers[0]$
FOR EACH num IN numbers
{
  IF $(num < smallest)$
  {
    $smallest \leftarrow num$
  }
}

Block:

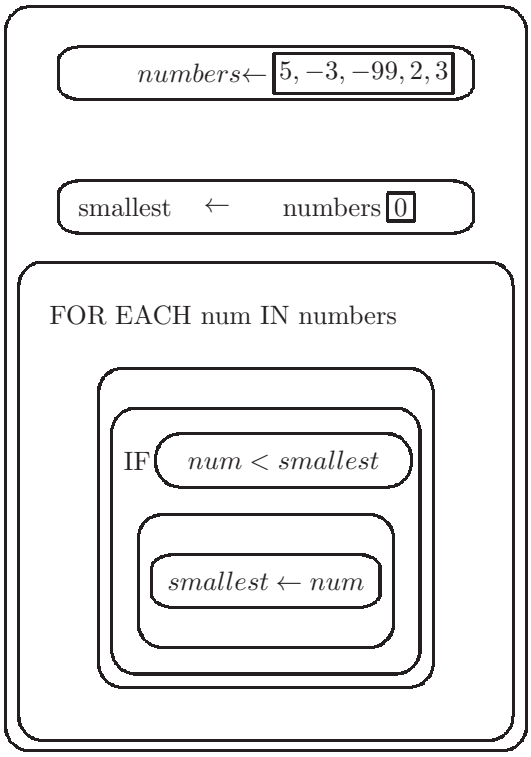


Figure 4.21: Algorithm to find the smallest number in the List [5,-3,-99,2,3]; when the algorithm terminates the variable *smallest* is storing the smallest value, -99.

Text:

```
PROCEDURE procedure name (parm1,parm2,...)
{
      block of
     statements
}
```

Block:

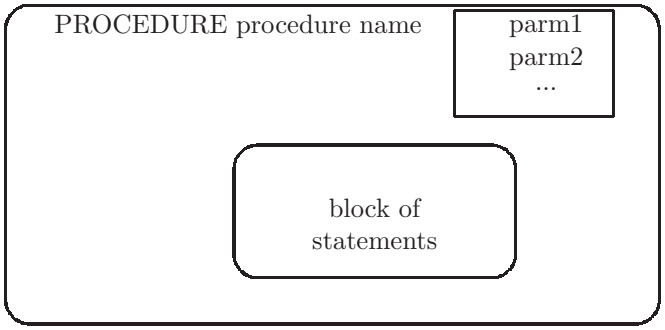


Figure 4.22: General format of a Procedure definition; parm1, parm2, etc. represent the procedure's parameters

Text: RETURN (expr)

Block: RETURN 

Figure 4.23: Format for a RETURN statement in a procedure; the expression (expr) is optional.

– The values for the inputs to the procedure, known as *arguments*[17] The arguments must agree with the parameters in the procedure definition, both in number and type

When a procedure is called, execution of the calling procedure is temporarily paused, the values of the arguments are assigned to the parameters, and the called procedure is executed. When the called procedure terminates, the calling procedure resumes execution at the point where it was paused.

4.1.6.1 Return value

A procedure may have an explicit result, or *return* value, in which case that result can be used as part of an expression. The format of the RETURN statement is shown in Figure 4.23.

The RETURN statement always terminates the procedure, returning control to the calling procedure. A procedure which returns no explicit result can be terminated with a RETURN which specifies no value to be returned.[18] A

[17] What we are calling arguments, are sometimes called *actual* parameters, to distinguish from formal parameters

[18] Many educators advise using a RETURN only at the end of a procedure.

Text:

```
PROCEDURE sumList(someList)
{
 sum ← 0
 FOR EACH num IN someList
 {
     sum ← sum + num
 }
 RETURN sum
}
```

Block:

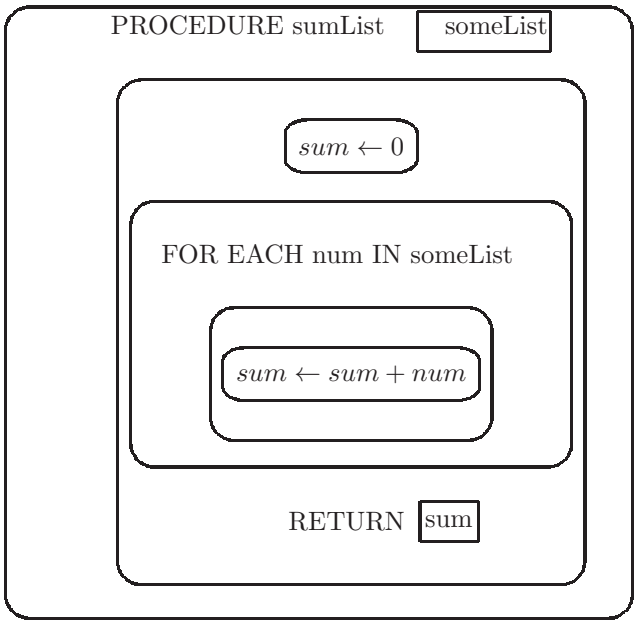


Figure 4.24: Definition of a procedure to return the sum of the numbers in a given List

procedure which does not return an explicit result may be useful for any *side effects* which it produces. Side effects include assignment of values to variables accessible to other procedures, producing output (see below), or any operation which changes the *state* of the computation.

As an example of a procedure which returns a value, we return to the algorithm which finds the sum of the numbers in a list (Figure 4.24). Here we show a procedure named sumList, which finds the sum of the numbers in *any* list; the list will be a *parameter* of the procedure. The procedure uses the same logic as our original algorithm, but it *returns* the resulting sum.

4.1.6.2 Output: DISPLAY

Algorithms are capable of producing *output*. We will not define exactly what this means, except to say that the value of an expression can be displayed in some form to a person who is using the algorithm. Think of it as a message to the user. The DISPLAY[19] statement can display the value of a given expression; the format of the DISPLAY statement is shown in Figure 4.25.

We can redefine our procedure to sum the numbers in a List, so that it will *display* the sum, instead of *returning* the sum. This algorithm is shown in

[19]The Computer Science Principles AP course description implies that the DISPLAY statement occurs only in procedures; however, it can conceivably be used anywhere in an algorithm.

Text: Block:

DISPLAY (expr) DISPLAY [expr]

Figure 4.25: Format for a DISPLAY statement

Text:

```
PROCEDURE sumList(someList)
{
 sum ← 0
 FOR EACH num IN someList
 {
     sum ← sum + num
 }
 DISPLAY sum
}
```

Block:

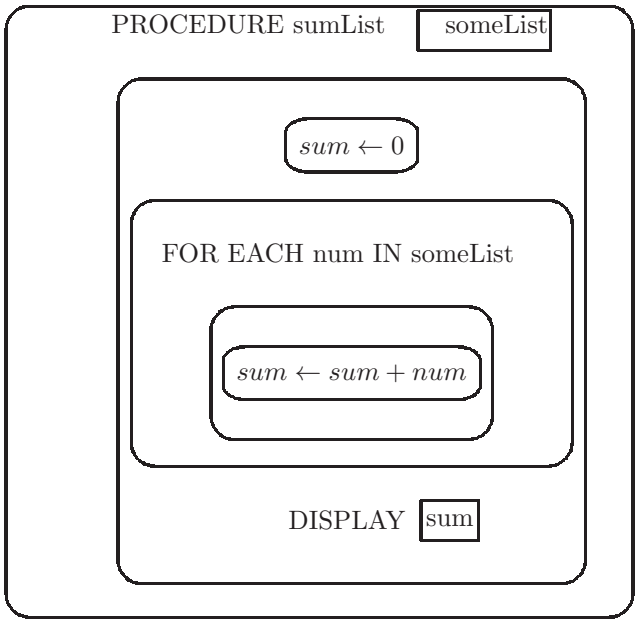


Figure 4.26: Definition of a procedure to display the sum of the numbers in a given List

Figure 4.26.

The student may be wondering which of these algorithms is preferable, Figure 4.24 or Figure 4.26. In situations where it is not clear whether the algorithm is to have side effects, either may be used. In cases where the result of a procedure is to be used by other procedures, it is best to use a return. On the AP exam, if you are asked to show a procedure, read the question carefully; it will probably tell you whether the result is to be returned or displayed.

4.1.6.3 Input: INPUT

A procedure has the capability of obtaining data values from the user of the procedure as it executes, a process known as providing *input*. In our Text and Block languages this is done with an INPUT statement, as shown in Figure 4.27.

Think of the INPUT statement as a call to a Procedure which returns a

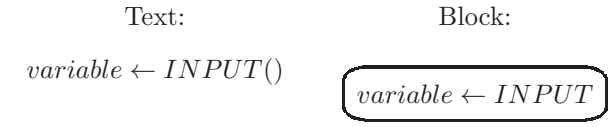


Figure 4.27: Format for an INPUT statement; the variable is assigned the value of the input provided by the user.

value; the value provided by the user.[20] An INPUT is typically preceded by an OUTPUT, to prompt the user to provide data, perhaps in a specific format.

4.1.6.4 The power of abstraction in procedures

We now present an example which we hope will expose the power of procedures. We define a procedure with three parameters, a List of (at least two) numbers and two indices (i.e. positions) in the List. The procedure's purpose will be to reorder the two given positions in the List, if necessary, so that the smaller number precedes the other number. For example, if the given List is [5,7,4,3,9], and we wish to order positions 1 and 3, the List would be changed to [5,3,4,7,9]. The procedure has no return value.[21] The name of the procedure is `order2`, and it uses a single conditional statement to determine whether the values at the given positions in the given List need to be exchanged; it is shown in Figure 4.28.

The `order2` procedure needs to *exchange* or *swap* the two values in the List. This can be done in a few ways. In Figure 4.28 the text version uses a variable, *temp*, to store one of the values. The Block version in that figure uses two variables, *val1* and *val2*, to store the values at positon p0 and position p1, respectively. Either of these strategies is acceptable.[22]

4.1.6.5 Combining Procedures

In an algorithm a procedure can be invoked from another procedure by providing the procedure name and the the arguments ,if any, in parentheses, as shown below:

```
procedureName(arg1,arg2,arg3...)
```

Each argument is an expression which is evaluated, and the value is copied to the corresponding parameter in the procedure definition. For example, to call the procedure `order2` on the List `fourNums`, to order positions 1 and 3:

```
order2(fourNums,1,3)
```

The procedure `order2` has no explicit return value. However, for procedures which do have a return value, the call may be part of an expression, using the procedure's returned value, just as it would a variable's value. For example, if

[20] We are not concerned here with *how* the user provides input, e.g. keyboard, microphone, storage device, etc.

[21] The reordering of the List is a side effect.

[22] It should be noted that this algorithm presumes that position p1 is smaller than position p2, i.e. position p1 precedes position p2 in the List.

Text:

PROCEDURE $order2(someList, p1, p2)$
{
IF $(someList[p1] > someList[p2])$
{
$temp \leftarrow someList[p1]$
$someList[p1] \leftarrow someList[p2]$
$someList[p2] \leftarrow temp$
}
}

Block:

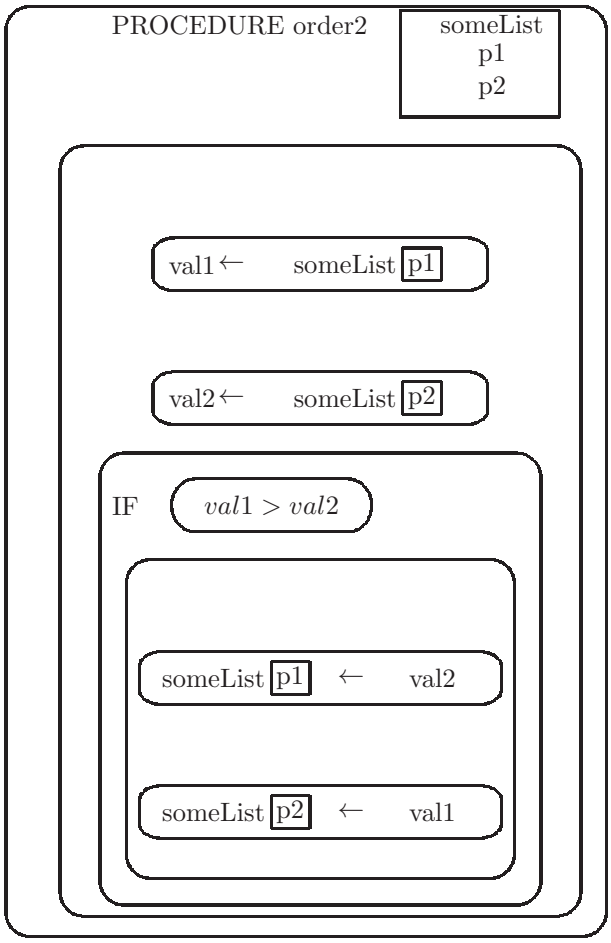


Figure 4.28: Definition of a procedure to order two of the the numbers in a List of (at least two) numbers, so that the smaller number precedes the other number

Text:

PROCEDURE $order3(someList, p1, p2, p3)$
{
  $order2(someList, p1, p2)$
  $order2(someList, p1, p3)$
  $order2(someList, p2, p3)$
}

Block:

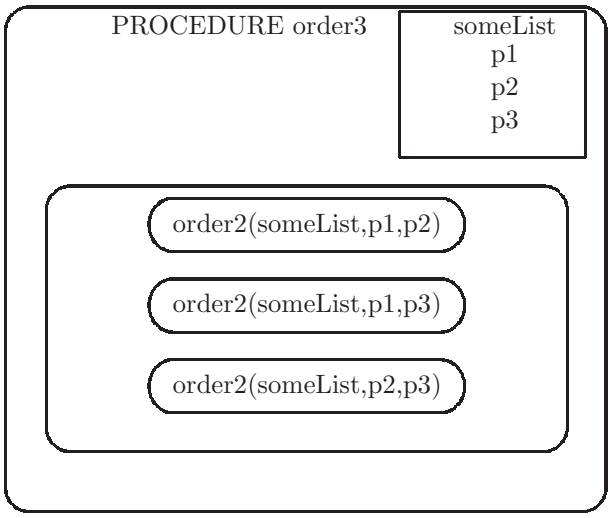


Figure 4.29: Definition of a procedure to order three of the numbers in a List of (at least three) numbers, so that they are ordered from smallest to largest.

the procedure `square` returns the value of its parameter multiplied by itself:
`area` ← `pi * square(radius)`

We now wish to develop a procedure to order three given positions in a given List. Rather than developing it from scratch, we will use the `order2` procedure. We call this new procedure `order3(list,p1,p2,p3)`; all that is needed are three steps, invoking the `order2` algorithm on each step:

1. `order2(list,p1,p2)`
2. `order2(list,p1,p3)`
3. `order2(list,p2,p3)`

The `order3` procedure is shown in Figure 4.29.

To see that this algorithm is correct, we provide a *trace* of its execution for the List [17,7,3] in Figure 4.30.

After the third invocation of the `order2` algorithm, the three values are ordered from smallest to largest.

In general there can be several different algorithms which solve the same problem. Our `order3` algorithm could have been defined as follows:

`order3(list,p1,p2,p3):`

1. `order2(list,p2,p3)`
2. `order2(list,p1,p3)`

Instruction	p1	p2	list
			[17,7,3]
order2(list,p1,p2)	0	1	[7,17,3]
order2(list,p1,p3)	0	2	[3,17,7]
order2(list,p2,p3)	1	2	[3,7,17]

Figure 4.30: Ordering the List [17,7,3] by calling order3(list,0,1,2); this diagram shows a trace of the algorithm.

3. `order2(list,p1,p2)`

Sometimes developing a new algorithm to solve a problem can yield insight into the problem itself. In the case of `order3`, it appears that three invocations to the `order2` algorithm are needed to solve the problem.

We stated above that the three fundamental building blocks, or control structures, for algorithms are sequence, selection, and iteration. We now include the combination of algorithms, or procedures, as the fourth building block.

4.1.6.6 Search

We often wish to search a List for a particular value; we'll call it the *target*. There are several algorithms which solve this problem; we'll examine two of them:

- One solution to this problem is to start at the beginning of the List and compare each value with the target, terminating when the target is found, or the end of the List is reached. This is called a *sequential search* algorithm. A sequential search procedure is shown in Figure 4.31; it returns the position of the target, or -1 if not found.

  The sequential search algorithm works fine for small lists, but for longer lists we would like to have an algorithm that doesn't take so much time. If we are to be searching, repeatedly, for many targets, it might be helpful to *sort* the list first, i.e. arrange the values in increasing (or decreasing) order. Once the list has been sorted, we can use a much faster search algorithm.

- The *binary search* algorithm applies only to lists which are sorted. In this case we can devise an algorithm which is much faster than the sequential search algorithm. In a binary search we begin by comparing the target with the value at the midpoint of the list.[23] There are four possible cases:

  – The value at the midpoint is equal to the target. The result is the midpoint.

  – The value at the midpoint is smaller than the target. Search the right half of the list for the target.

[23] In this algorithm we are assuming that the list has been sorted in *increasing* order.

Text:

```
PROCEDURE sequentialSearch(list, target)
{
   ndx ← 0
   FOR EACH num IN list
   {
     IF (num = target)
        RETURN ndx
     ndx ← ndx + 1
   }
  RETURN −1
}
```

Block:

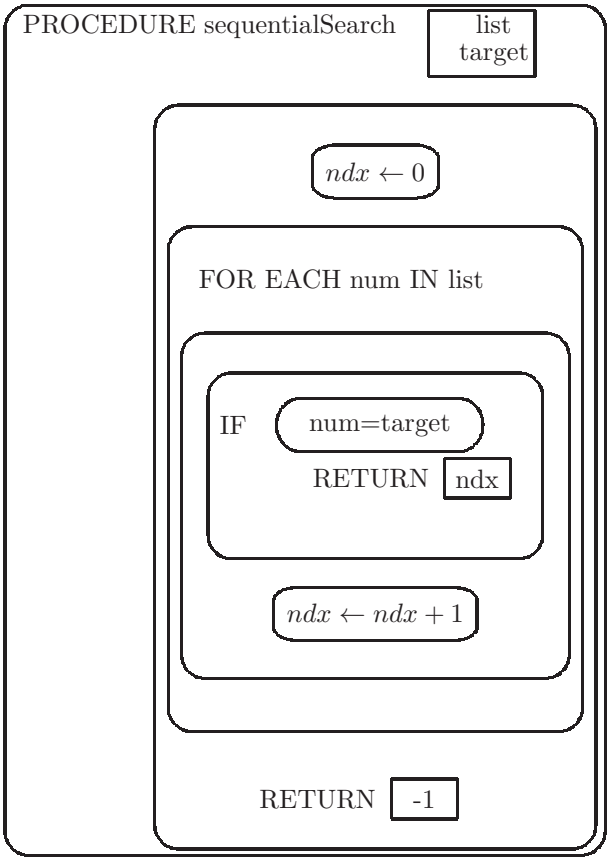


Figure 4.31: Definition of a procedure to search a List for a given target value.
It returns the position of the target, or -1 if not found.

- The value at the midpoint is greater than the target. Search the left half of the list for the target.
- The portion of the list we are searching is empty. The target is not found; the result is -1.

The important aspect of this algorithm is the fact that after each comparison, we eliminate half the values from consideration. The algorithm is shown in Figure 4.32.

Note that the termination condition for the REPEAT UNTIL loop in this algorithm is *start* > *end*. As the loop repeats, *start* and *end* get closer and closer to each other; on each iteration either *start* is increased, or *end* is decreased. If *start* becomes greater than *end*, the target is not found in the list, and the algorithm returns -1.

You may have used this algorithm often without realizing it. Examples:

- When searching for a word in a (paper) dictionary, encyclopedia, or telephone book,[24] a sequential search, looking at every word until you find the one for which you are searching, will take too much time. Instead you start in the middle (or somewhere close to where you think the word would be) and move forward or backward several pages.
- When searching for your flight on an electronic display of arrivals or departures at an airport. If there are many flights shown, it is best to find the *sort field* first, and do a binary search.
- When searching for an office name on the directory for a large building or hospital, a binary search is used.
- When searching for a course in your school registrar's list of courses offered, a binary search is used.

4.1.6.7 Simulation and random numbers

The importance of simulation was discussed in chapter 2. In order to implement a simulation which utilizes random behavior we use a *random number generator*. This would be software that produces a sequence of numbers with no apparent relationship. More formally this sequence of numbers should be called a *pseudo-random* sequence; anything produced by an algorithm cannot be truly random.

In our algorithmic languages we have a procedure named RANDOM(start,end) which returns a pseudo random number in the range [start..end], inclusive, each time it is called. The format of this procedure is shown in Figure 4.33.

As an example we define a procedure to roll dice, cubes with faces having 1,2,3,4,5, or 6 pips. Our procedure will have 1 parameter: the number of dice cubes to be rolled. It will return a list of numbers, each of which is in the

[24] In today's world many of these searches are done with personal digital devices, which probably expedite the search with hash tables; many students have no experience with binary search, and may have forgotten the sequence of letters in the alphabet.

Text:

```
PROCEDURE binarySearch(list, target)
{
    start ← 0
    end ← LENGTH(list) − 1
    REPEAT UNTIL start > end
    {
        mid ← (start + end)/2
        value ← list[mid]
        IF (value = target)
            RETURN mid
        IF  (value < target)
            start ← mid + 1
        IF (value > target)
            end ← mid − 1
    }
    RETURN  −1
}
```

Block:

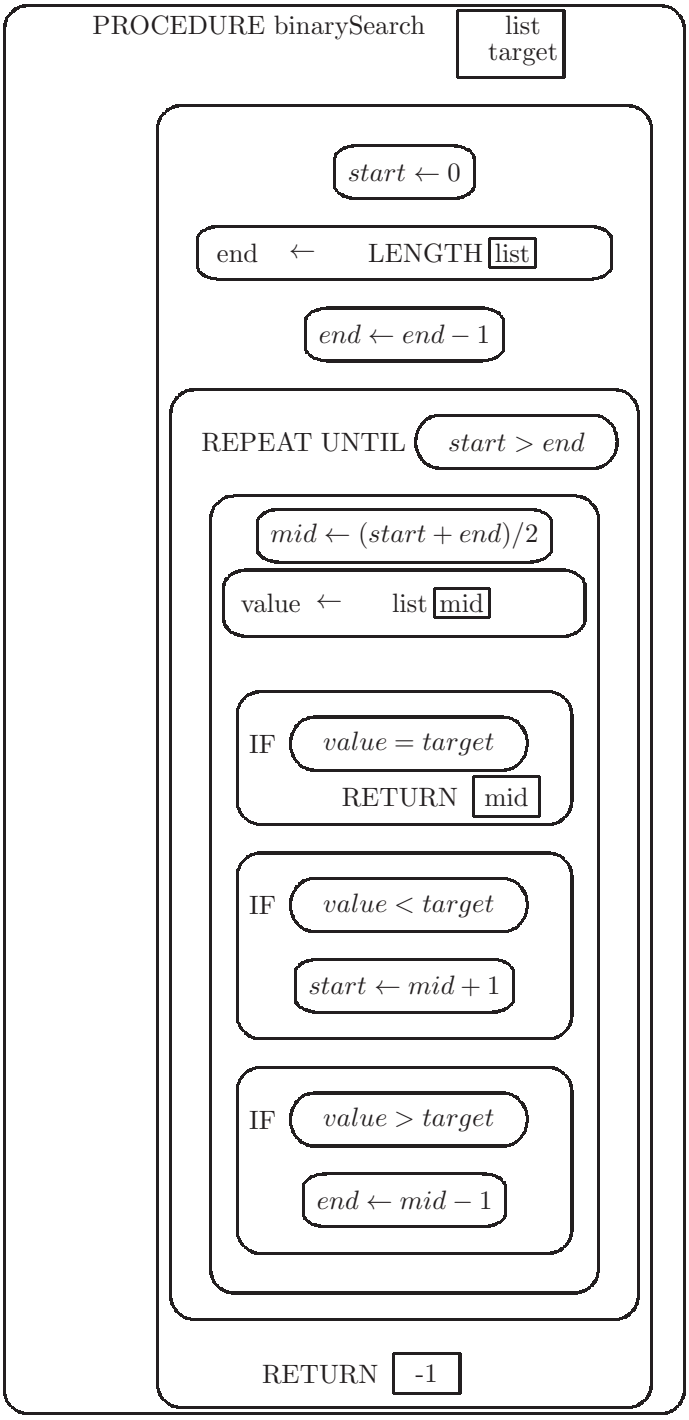


Figure 4.32: Definition of a procedure to search a sorted List for a given target value, using the binary search algorithm. It returns the position of the target, or -1 if not found.

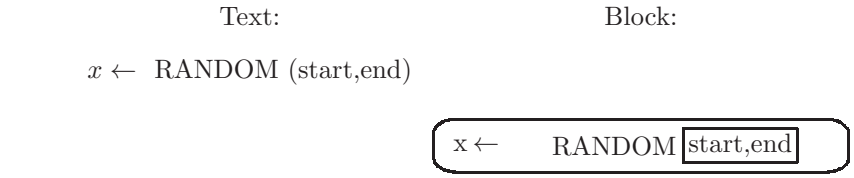


Figure 4.33: Format for a RANDOM(start,end) statement in a procedure; it returns a random integer in the range [start..end], inclusive, which is then assigned to the variable x

range [1..6], one number for each die that is rolled. The algorithm is shown in Figure 4.34.

4.1.7 Languages for Algorithms

The two languages that we have introduced to describe algorithms (Text and Block[25]) are somewhat formal and precise, yet we do not intend to execute either of them on a computer. They are used purely for the description of algorithms. This kind of language is often called *pseudo code*. It is precise enough to convey the behavior of the algorithm, yet is not intended for actual automatic execution on a computer.

Sometimes people also use *natural* language, such as English, to describe algorithms. We should be careful, however, to understand that natural languages are not precise and are susceptible to ambiguities and misinterpretation.

4.1.7.1 Programming languages

More precise languages for describing algorithms are known as *programming* languages; at least one of these will be examined in detail in chapter 5. Students often ask "Why is there more than one programming language? Why not simply use the best language for all programming tasks"? We will see that some programming languages are better suited for certain kinds of problems, known as problem *domains*. For example, some languages are better for business related problems, while other languages are better suited for scientific or engineering problems, and other languages are better suited for working with text. However, the control structures that we have introduced: sequence, selection, and iteration, together with the combination of algorithms, are sufficient to solve any problem that has a solution.[26] Thus the existence of several different programming languages is a result of the need for ease or convenience in programming, which turns out to be fairly difficult for many problems.

[25] The Block language described here was designed by the Computer Science Principles AP Committee and is not one that is favored by this author.

[26] As we will see later in this chapter, there are some problems which have no algorithmic solution.

Text:

```
PROCEDURE roll(n)
{
result ← { }
REPEAT n TIMES
 {
   die ←  RANDOM(1,6)
   APPEND(result, die)
 }
RETURN result
}
```

Block:

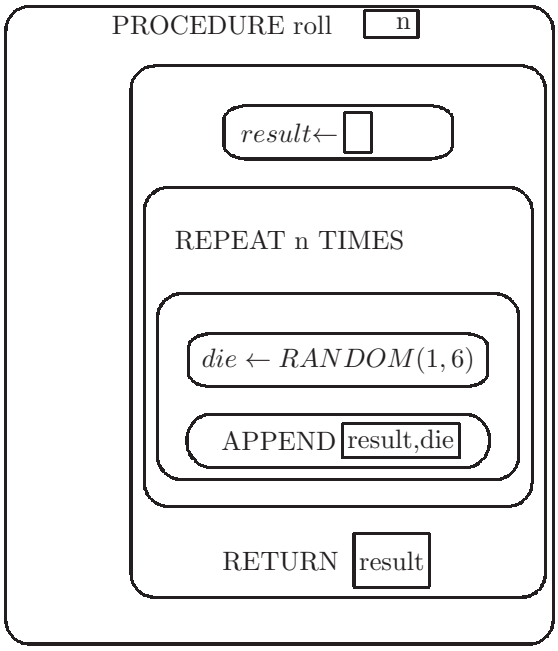


Figure 4.34: Procedure to roll a given number of dice. It returns a list of the values shown on the dice, each of which is in the range [1..6].

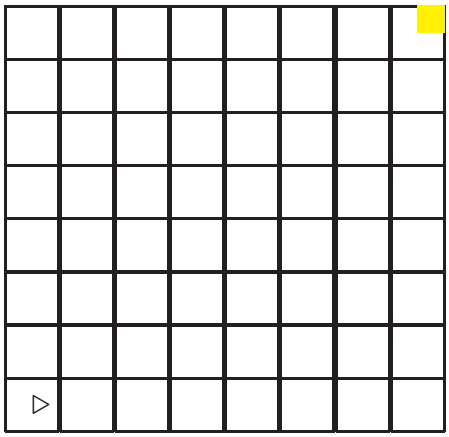


Figure 4.35: Diagram of a robot's environment, showing a start location (lower left) and a target destination (upper right)

4.1.7.2 Clarity and readability of a language

In addition to ease and convenience, we also note that clarity and readability of an algorithm can be affected by the choice of a language. Clarity and readability are important when collaborating with other people in the development process. Clarity and readability are also important for the developer who is struggling to understand his/her own programs.

4.1.8 Robot algorithmic language

As an example of an algorithmic language the AP course includes a fairly simple *robot* language which is used to guide an imaginary robot through a grid-like environment. The robot's environment is similar to a checker board, as shown in Figure 4.35, though not limited to an 8x8 grid.

The robot is capable of moving in a horizontal or vertical direction, one square at a time. The robot is also capable of sensing when it has reached the edge of the grid. At any point in time the robot is facing either north, south, east, or west, which is known as the robot's direction. The movement of the robot, and changes to its direction, are controlled by an algorithm using the following commands:

- `MOVE_FORWARD()`
  The robot will move one square in the direction that it is currently facing
- `ROTATE_LEFT ()`
  The robot will change its direction to the left of its current direction: (north-west-south-east-north)
- `ROTATE_RIGHT ()`
  The robot will change its direction to the right of its current direction:

Text:

```
PROCEDURE step()
{
 REPEAT UNTIL
        NOT CAN_MOVE(forward)
 {
    MOVE_FORWARD()
    ROTATE_LEFT()
    MOVE_FORWARD()
    ROTATE_RIGHT()
 }
}
```

Block:

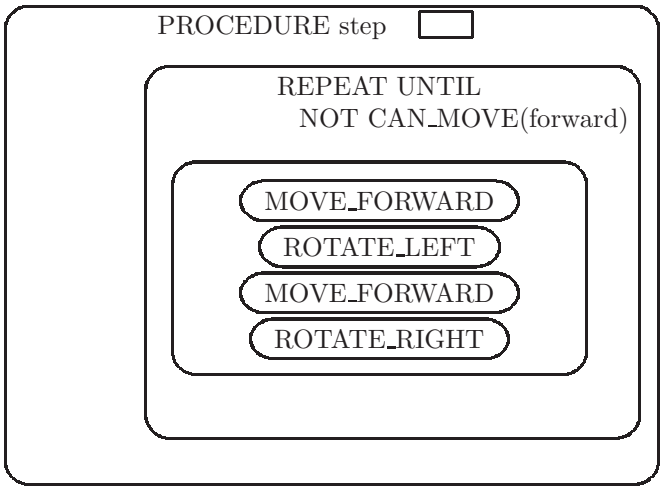


Figure 4.36: Definition of a procedure to move the robot in a step-like pattern from the lower left corner to the upper right corner

(north-east-south-west-north)

- `CAN_MOVE (direction)`
  This is a boolean operation which returns true if the robot is capable of moving in the given direction (i.e. it will not go off the edge of the grid). The direction can be `left`, `right`, `forward`, or `backward`.

An example of a procedure that controls the robot is shown in Figure 4.36. This procedure causes the robot to move in a 'step-like' manner from the lower left corner of the grid to the upper right corner of the grid, as shown in Figure 4.37.

4.1.9 Exercises

1. Define procedures to solve each of the following problems:

   (a) A procedure, named `isOrdered` with two parameters. It should determine whether the parameters are ordered, i.e. whether the first parameter is less than or equal to the second parameter, and return a true/false value.

   (b) A procedure, named `order3`, with four parameters. It should have the same effect as the order3 procedure defined in this section, but should be done without invoking the `order2` procedure.

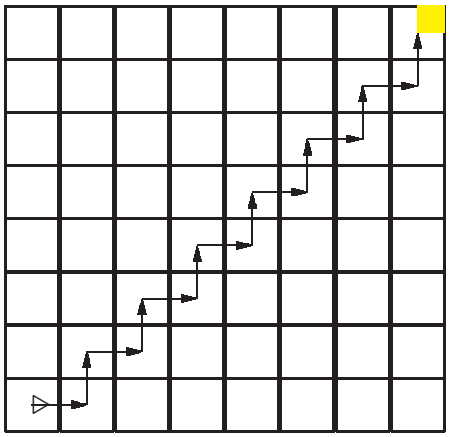


Figure 4.37: Robot moving in a step-like pattern on an 8x8 grid

(c) The Fibonacci sequence is a sequence of positive integers, beginning with 1,1, in which each number is the sum of the previous two numbers in the sequence:[27]
1,1,2,3,5,8,13,21,34,55,...
Define a procedure named `fib` with one parameter, n. It should return the value of the n^{th} number in the Fibonacci sequence (let the first position be 0).
fib(0) = 1
fib(1) = 1
fib(2) = 2
fib(3) = 3
fib(5) = 8

(d) The *modulus* operation produces the remainder upon integer division (see Figure 4.38). Define procedures (using two different algorithms) named `mod` with two parameters, x and y. The procedure should find the remainder when x is divided by y. Assume that x and y are both positive integers.

- Use an iteration, inolving repeated subtraction
- Use no iterations. Assume that the / operator produces the integer quotient.

(e) A procedure named `isDivisibleBy` with two parameters x and y. It should determine whether the value of x is divisible by the value of y. Assume that both parameters are positive integers.
Hint: Use the `mod` algorithm.

[27]The Fibonacci sequence appears in many natural biological formations, and has been studied extensively by mathematicians and scientists.

x	y	Quotient of $x \div y$	Remainder of $x \div y$
6	3	2	0
17	5	3	2
9	13	0	9
6234327	10	623432	7

Figure 4.38: Table showing the integer quotient and remainder (modulus) when x is divided by y.

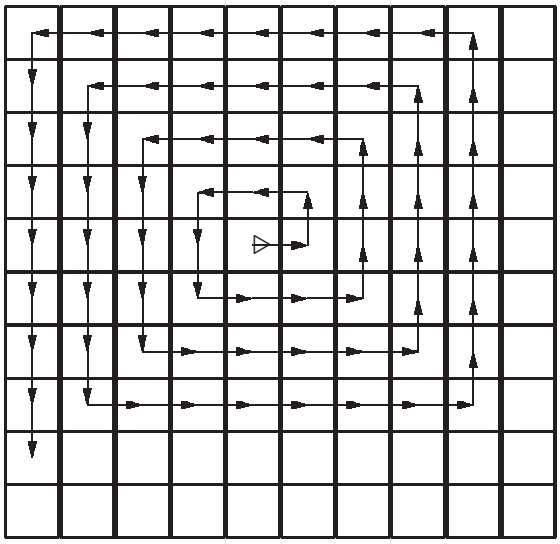


Figure 4.39: Robot moving in a spiral-like pattern on an 8x8 grid

(f) A procedure named `even`, with one parameter, x. The procedure should determine whether the value of x is an even number.
Hint: Use the `isDivisibleBy` algorithm.

2. In the previous problem you defined the `order3` procedure without invoking the `order2` procedure. Which definition of `order3` do you think is more clear and readable?

3. What are some of the programming languages that you may have heard of?

4. Define a procedure named `spiral` for a robot on the grid of a given size. This algorithm should cause the robot to move in a path that resembles a spiral, beginning at the center of the grid, as shown in Figure 4.39.

5. Show the path of the robot on a 12x12 grid, beginning at the lower left corner, which is generated by the following procedure:

```
Procedure exercise()
{
    n ← 1
    REPEAT 4 TIMES
    {
        REPEAT n TIMES
            MOVE_FORWARD();
        ROTATE_LEFT();
        REPEAT n TIMES
            MOVE_FORWARD();
        ROTATE_RIGHT();
    n ← n + 1
    }
}
```

4.2 Limitations of algorithms

In the previous section we have introduced the notion of *algorithm* and provided several examples of algorithms. In this section we raise some crucial questions regarding algorithms:

- Can all precisely stated problems be solved with an algorithm?
- If not, what are some examples of problems which have no algorithmic solution?
- For any given solvable problem, does it have equally usable algorithmic solutions?

4.2.1 Algorithm performance

We have seen that for any given problem, there can be more than one algorithm to solve that problem. It would be helpful to determine which algorithm is better from a standpoint of performance.[28] Here we attempt a somewhat formal effort to describe the quality of an algorithm's performance. We attempt to answer the questions, in a mathematical way:

- How fast is this algorithm?
  - This should be a measure of the running time, independent of the computing platform on which it may execute (i.e. the computer hardware and software used to execute the algorithm's implementation).
  - The running time will typically be a function of the size of the input to the algorithm.

[28] It might also be helpful to measure an algorithm's quality by assessing its readability and clarity, but that is beyond our scope.

Algorithm name	Run time formula	Run time description
order(x,y)	1	Constant
order3(x,y,z)	1	Constant
isOrdered(x,y)*	1	Constant
fib(n)*	n	Linear
mod(x,y)*	x / y	Linear
isDivisibleBy(x,y)*	x / y	Linear
even(x)*	x / 2	Linear

Figure 4.40: Table showing a running time formula for several algorithms defined in the previous section. * This algorithm was introduced as an exercise.

  - We would like to measure the running time as the size of the input becomes very large.[29]

- How much memory does this algorithm require?
  - Does this algorithm require a lot of temporary memory, in addition to the memory used for the input?
  - We give less attention to this question. With today's technology, most algorithm implementations will be limited by the time required to execute, rather than the space required for execution; most algorithms tend to 'run out of time' before they 'run out of space'.

4.2.1.1 Reasonable time

As we examine the time performance of algorithms, we will see that some algorithms are generally faster than others. We will pay particular attention to those algorithms which require so much time to complete execution, that they are essentially unusable for large input values; thus we distinguish between *reasonable* run time and *unreasonable* run time, for an algorithm's performance measure. All of the examples of algorithms that we have seen thus far have a reasonable run time. More mathematically, we can describe the run time as a function of the size of the input. In doing so, we we will provide a formula for the run time such that it will be *proportional to*[30] the actual run time for a particular implementation of that algorithm.

Figure 4.40 shows the run time formula for several algorithms defined in the previous section. All of these algorithms execute in reasonable time. The first three algorithms shown in that table do not involve any loops. Thus the run time does not depend on the size of the input. A formula which is proportional to a fixed run time is simply 1. We call this *constant run time*.

[29] The phrase 'becomes very large' is intentionally undefined.

[30] A formula, f, is proportional to a measurement, m, if there is some constant, c, such that $f = c \times m$.

The algorithm `fib(n)` however, does involve a loop which repeats n-2 times. Thus the formula n will be proportional to the actual run time, which increases as the the value of n increases. In this case we say that the run time is *linear* because the formula is a linear function of the input.

The algorithm `mod(x,y)` also involves a loop.[31] The number of times the loop repeats will be the quotient of x / y. For example, if x = 102, and y = 7, the loop will repeat 14 times, because 102 / 7 produces a quotient of 14. This algorithm also has a linear run time.[32]

The next algorithm in Figure 4.40 is `isDivisibleBy(x,y)`. It has no explicit loops; however, it invokes the *mod(x,y)* algorithm which does have a loop. We call this a *hidden loop*. Therefore, the run time of `isDivisibleBy(x,y)` will be the same as the run time for `mod(x,y)`.[33]

The run time for the `even(x)` algorithm follows a similar argument. There is a hidden loop in `even(x)` because it invokes `isDivisibleBy(x,y)`, which has a run time of x / y. In this case the value of y is always 2, thus `even(x)` has a run time of x /2.[34]

All of the algorithms that we have seen so far have a run time which is either constant or linear. We now examine a somewhat slower algorithm (greater run time). The problem we wish to solve is:
Find the sum of the first n numbers in the Fibonacci sequence.
To solve this problem we will find the values of the first n numbers in the sequence, and add them up. We do this with a variable known as an *accumulator*, which stores the sum of all the numbers seen so far in the loop. The algorithm is called `sumFib(n)` and is shown in Figure 4.41.[35]

In this algorithm we assume that decrementing the loop variable, *n*, does not affect the number of times that the loop repeats.[36]

In this algorithm `sumFib(n)` we are invoking the `fib(n)` procedure. We use the result of `fib(n-1)` in a calculation. The result of `fib(n-1)` is added to the value of *sum*, and then stored back into the variable *sum*.

The algorithm `sumFib(n)` is our first example of an algorithm where there is a loop within a loop. We call this a *nested loop*. The outer loop is expicit in `sumFib(n)`. When it invokes the `fib(n)` algorithm, this also contains a loop; this is the inner loop. Each time the outer loop in `sumFib(n)` repeats the inner loop in `fib(n)` executes several repetitions.[37] The run time performance of

[31] Here we are referring to the algorithm which uses repeated subtraction.

[32] It is true that for some inputs, such as x = 1000 and y = 1000, the run time will be very small, but here we are talking about *average case* performance, a concept which is beyond our scope.

[33] It is true that there are some extra steps in `isDivisibleBy(x,y)` which are not included in `mod(x,y)`, but the time required for these steps will be negligible, as the size of x / y becomes very large.

[34] It would also be correct to describe the runtime as simply x: If x/2 is proportional to the run time, then x is also proportional to the run time.

[35] This is not the most efficient solution to this problem.

[36] Many educators would discourage the practice of modifying loop control information in the body of the loop, since this could have different effects in various programming languages.

[37] The number of times the inner loop executes will decrease by 1 each time `fib(n)` is invoked, but including this in our analysis is beyond our present scope.

Text:

```
PROCEDURE sumFib(n)
{
 sum ← 0
 REPEAT n TIMES
 {
     sum ← sum + fib(n − 1)
     n ← n − 1
 }
 RETURN sum
}
```

Block:

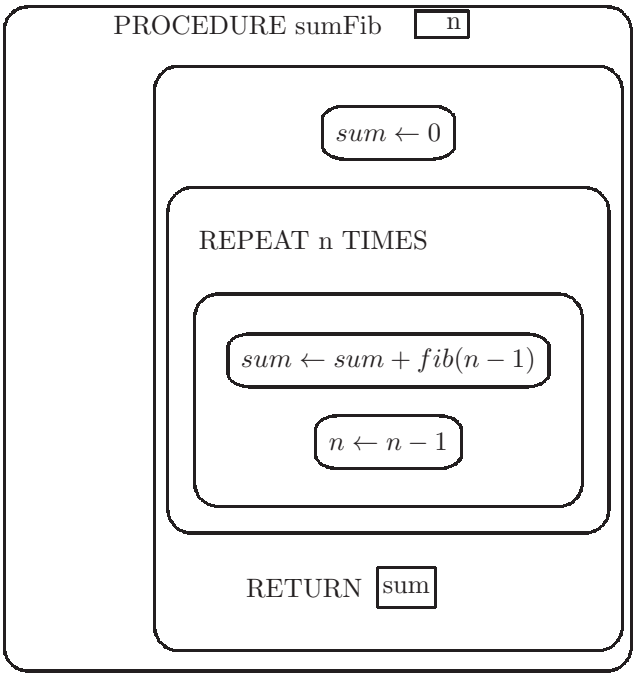


Figure 4.41: Definition of a procedure to return the sum of the first n numbers in the Fibonacci sequence

Text:

PROCEDURE $fibSlow(n)$
{
IF $(n < 2)$
RETURN 1
RETURN $fibSlow(n-1)+$
$fibSlow(n-2)$
}

Block:

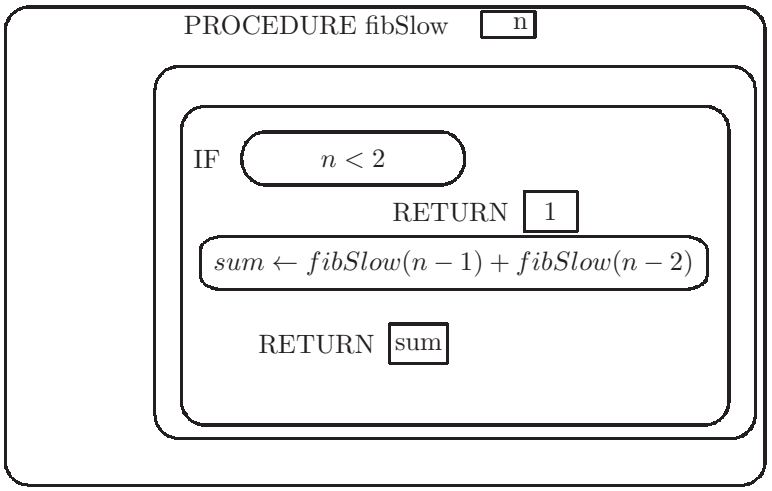


Figure 4.42: Definition of a procedure to return the nth number in the Fibonacci sequence, using a recursive call

`sumFib(n)` will be proportional to n^2. We call this *quadratic* performance. The algorithm `sumFib(n)` will be somewhat slower than the algorithms we have seen so far, and the difference in performance will be more evident when the value of the parameter, n, is larger. However, we still classify the run time of `sumFib(n)` as reasonable time.

4.2.1.2 Unreasonable time

There are some algorithms which take so long to execute that we say the run time is *unreasonable*. In such cases for a fairly large input value, we can wait while the computer executes the algorithm, and never see a solution even though we know the algorithm must eventually terminate with a solution.

As an example of an algorithm which has an unreasonable run time, we return to the problem in which we wish to find the value of the n^{th} number in the Fibonacci sequence. In this case we use a different algorithm, one which is much slower. We call it `fibSlow(n)`; it is shown in Figure 4.42.

Note that the `fibSlow(n)` algorithm invokes the `fibSlow(n)` algorithm in the case where n is not less than 2. It invokes itself! We call this a *recursive* algorithm, and the usage of recursive algorithms is called *recursion*. In this case each call to `fibSlow(n)` results in two more calls to `fibSlow(n)`. Figure 4.43 depicts a diagram showing these recursive calls for an initial input value of 4.

To analyze the run time performance of `fibSlow(n)` we note that if we were to call `fibSlow(5)` there would be almost twice as many calls to `fibSlow(n)` in our diagram. Increasing the input by only 1 will cause the run time to

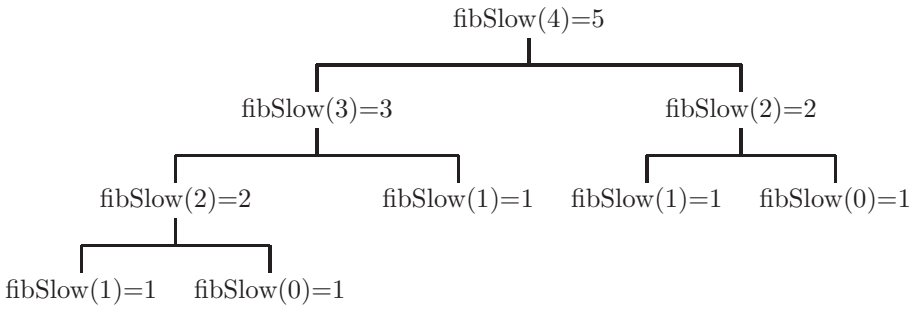


Figure 4.43: A diagram showing recursive calls to the fibSlow(n) algorithm

double! Thus the formula for the run time of `fibSlow(n)` is 2^n. [38] We call this *exponential* run time. Note there is a very significant difference between 2^n and n^2 (quadratic run time). As n becomes large, 2^n increases much more sharply than does n^2. Moreover, as n becomes large, 2^n increases much more sharply than does n^p, for any value of p.

An implementation of an algorithm with exponential run time on any computer will take so long to run that we will not see a result in our lifetime (for a fairly large input value). Such algorithms are essentially useless. In chapter 5 we will implement this algorithm and experiment with its run time.

Fortunately, if we wish to find the value of the n^{th} number in the Fibonacci sequence, we can use our original `fib(n)` algorithm, which has linear performance. However, there are some problems for which we are not so fortunate:

- There are problems for which we are unable to find an algorithm with reasonable run time, though in theory there might be such an algorithm.
- There are problems for which we are sure that there are no algorithms with reasonable run time.

In either case we often resort to a strategy in which we use a *heuristic*. A heuristic is similar to an algorithm, in that it consists of a finite number of well-defined steps. However, a heuristic is not guaranteed to terminate, nor is it guaranteed to produce a correct solution to the problem if it does terminate. It will often produce an approximation to a correct solution in a reasonable time. Despite these limitations of heuristics, we use them as a last resort, when we are unable to find an algorithm that runs in reasonable time. The bottom line: an approximation to a correct solution is better than no solution at all.

Problems which have no reasonable algorithmic solution are often *optimization* problems, i.e. find the best way of doing something, or find the easiest way to get somewhere.

An example of such a problem is known as the *traveling salesman* problem: Given a map with several cities, and the distances between the cities, what path

[38] We do not mean to imply that recursion causes unreasonable run time; many recursive algorithms are quite efficient.

should a salesman take to visit all cities, such that the total distance covered is minimized? Several algorithms to solve this problem have been proposed, but none of them have a reasonable run time; thus a heuristic is usually used if the number of cities is not very small. People have searched for an algorithm with reasonable run time for this problem, to no avail. Yet no one has been able to prove that no such algorithm exists.

There are many problems which can be shown to be equivalent to the traveling salesman problem in this respect. They are called *NP-complete* (or sometimes *NP-hard*) problems.[39] Two important results are:

- If we could find a reasonable algorithm for any one of these problems, we would have reasonable algorithms for all of them!
- If we could prove there is no reasonable algorithm for any one of these problems, we would have a proof that there is no reasonable algorithm for any of them!

When we say that a problem is NP-hard we don't mean that it is difficult to find an algorithm which solves the problem; we mean that it is hard to find an algorithm that has a reasonable run time.

4.2.2 Solvable problems

In this section we are interested in determining whether there are problems for which no algorithmic solution exists. We are less concerned with run time performance here. For those problems which have no algorithmic solution we have no choice but to use a heuristic, which will not find a perfect solution to the problem.

There are many problems in today's world which may not have an algorithmic solution:

- How can we persuade people of different cultures, religions, and nationalities to live together peacefully?
- Is Bach's Toccata and Fugue the most enjoyable music ever written?
- Is N'Sync better than FiftyCent?
- Is there a God?

These are all significant problems, and we may be very interested in finding algorithmic solutions to them. However, these problems are not stated precisely enough for our purposes; they are not mathematically formulated. In our usage of the word 'problem', they really are not problems. In order for a problem to have an algorithmic solution, it must be stated precisely in somewhat mathematical terms.

[39] The origin and meaning of these terms is beyond the scope of this course. Instructors are referred to Donald Knuth's January 1974 letter to *ACM SIGACT News*.

4.2.3 Undecidable problems

A *decision* problem is a problem for which the result is a simple yes or no (true or false). In the exercises of the previous section, the algorithms `isOrdered(x,y)` and `even(x)` are solutions to decision problems.

A decision problem which has no algorithmic solution (not even an unreasonably slow one) is called *undecidable*. Here is an example of an undecidable problem:

> Given a computer program,[40] determine whether that program, when executed on a computer will ever terminate.

Recall in the section on algorithm design and implementation, that the fundamental building blocks, or control structures of an algorithm are sequence, selection, and iteration, along with combining algorithms. An iteration can be specified using a complex logical expression, making it impossible to determine whether the iteration will ever terminate.

4.2.4 Evaluation of algorithms

Here we try to answer the question: How good is a given algorithm? There are several different aspects of algorithms which can be examined to answer this question:

4.2.4.1 Run time

Does this algorithm run faster, for a given input, than other algorithms which solve the same problem?

The time required for an algorithm to run to completion generally depends on the size of the input to the algorithm. We have seen that:

- An algorithm with constant run time will execute faster than an algorithm with linear runtime, for large inputs.
- An algorithm with linear run time will execute faster than an algorithm with quadratic runtime, for large inputs.
- An algorithm with polynomial run time will execute faster than an algorithm with exponential runtime, for large inputs.

4.2.4.2 Memory required

Does this algorithm require less temporary memory, for a given input, than other algorithms which solve the same problem?

We can use the same kind of measure on memory that we used above for time:

[40] A computer program is just plain text, written with a particular syntax, that can be translated to executable code on a computer.

```
constant < linear < quadratic < non-quadratic polynomial < exponential
```

where we are now measuring the space required, rather than the run time. With today's technology, an algorithm will usually run out of time before it runs out of space; consequently, most algorithms are assessed for time efficiency, and are assessed for space efficiency only when memory requirement is a limiting factor or when algorithms with the same time efficiency need to be compared.

4.2.4.3 Efficiency by input

Is this algorithm more efficient in time and/or space for some inputs than it is for other inputs?

This question is somewhat subjective. There are many problems where knowledge of the data involved can lead to more efficient algorithms. For example, if you are searching a list of names for a particular name (a 'target'), you could start at the beginning of the list and examine every name until you found the target, or reached the end of the list. If however, you knew that the names were in alphabetic order, the algorithm could terminate when reaching a name that comes after the target name alphabetically.[41]

4.2.4.4 Size of the algorithm

How many steps are there in this algorithm? Shorter algorithms are generally preferable to longer algorithms; short algorithms are generally easier to read, understand, and implement. We saw evidence of this in the exercises for the first section of this chapter (the `order3(list,p1,p2,p3)` algorithm which does not make use of the `order2(list,p1,p2)` algorithm; it is much longer, making it more difficult to read and understand). In general we will seek short, simple algorithms to solve complex problems, and we will not seek long complex algorithms to solve simple problems. Simple problems are often best solved without the aid of an algorithm.

4.2.4.5 Clarity and readability

How clear and readable is this algorithm? Algorithms are most useful when they can be shared and used by many people in a variety of contexts. If an algorithm is neither clear nor readable, it is difficult for others to use and/or customize it for a particular problem or data set.

The clarity and readability of algorithms relates directly to the benefits of collaboration when implementing the algorithms and when developing new algorithms.

Exact measurements of clarity and readability are not generally used; this is a more subjective measure of an algorithm's quality.

[41] This is known as a *sequential search* algorithm; a *binary search* algorithm (of a sorted list), which looks at the name in the middle of the list first, and eliminates half the names from consideration, is a different algorithm.

4.2.5 Exercises

1. What is the run time efficiency of each of the following algorithms? In each case your response should be one of the following:
   `Constant, Linear, Quadratic, Exponential`
   Also indicate whether the algorithm runs in reasonable time.

   (a)
   ```
   PROCEDURE gauss(n)
   {
   sum ← 0
   ctr ← n
   REPEAT n TIMES
       {
           sum ← sum + ctr
           ctr ← ctr − 1
       }
   RETURN sum
   }
   ```

   (b)
   ```
   PROCEDURE gaussFast(n)
   {
   sum ← (n × (n + 1))/2
   RETURN sum
   }
   ```

   (c)
   ```
   PROCEDURE looper(n)
   {
   sum ← 0
   REPEAT n TIMES
   {
       i ← n
       REPEAT n TIMES
       {
           sum ← sum + i
           i ← i − 1
       }
   }
   ```

   (d)
   ```
   PROCEDURE slow(n)
   {
   ctr ← 0
   i ← 1
   REPEAT n TIMES
   {
       REPEAT i TIMES
       {
           ctr ← ctr + 1
   ```

```
    }
    i ← i * 2
}
}
RETURN ctr
}
```

2. Refer to the previous exercise:
   (a) What is the result of gauss(5)?
   (b) What is the result of gaussFast(5)?
   (c) What is the result of looper(5)?
   (d) What is the result of slow(5)?
3. (a) What is the difference between a heuristic and an algorithm?
   (b) Give an example of a real problem which we may wish to solve with a heuristic rather than an algorithm.
4. Search the internet to find a decision problem which is undecidable.
5. Referring to the first exercise, the algorithms `gaussFast(n)` and `gauss(n)` solve the same problem.
   (a) Describe the problem which those two algorithms solve.
   (b) In what respect(s), if any, is the algorithm `gaussFast(n)` better than the algorithm `gauss(n)` ?
   (c) In what respect(s), if any, is the algorithm `gauss(n)` better than the algorithm `gaussFast(n)` ?

Chapter 5

Programming (with C++)

As we mentioned in chapter 4, an algorithm can be *implemented* with a computer *program*. Each step of an algorithm will correspond to a statement in a computer program. The program is the realization of the algorithm.

5.1 Program development

5.1.1 Why program?

Advantages of a program over an algorithm:

- With a program (and a computer) we can test an algorithm to verify that it is producting the correct result(s).
- Since programs run on real machines, they can interact with the outside world (drive a car, control household security systems, perform retail and financial transactions, etc.)
- Programs are often more precise than algorithms; sometimes algorithms are expressed with pseudo-code. A program must conform to the proper syntax and semantics of the programming language in which it is written.

Programming results in the creation of software, and software (which can be exectuted on a computer or other digital device) has changed our lives. All digital devices: computers, tablets, phones, automobile components, music players, tv sets, electronic games, etc. must be programmed with the appropriate software to provide services. Thus, programming has enabled the creation of digital artifacts.

This chapter contains an introduction to the programming process and principles which apply to most programming languages. Examples of these principles are provided for a particular programming language - C++. The concepts

explained here generally apply to other languages, with minor differences in syntax and approach.[1]

There are many possible motivations for software development. We may wish to:

- Solve a problem
- Build a tool for others to use
- Express something creatively
- Create new knowledge

Programs are often used in conjunction with visual displays, such as the one on your phone. Programs also produce sound, listen for sound, sense your touch, determine your location, and interact with the world in many ways.

People sometimes develop programs to satisfy their own personal curiosity. This can lead to advancements in their own field of study or in other fields of study.

Programs which you develop for your own personal use may often be simple, short, and relatively easy to develop. Programs developed for widespread use by many people often need to take extensive precautions for safety and security, particularly if they provide access to the internet.

5.1.2 Problem solution

When tackling the solution to a problem by implementing an algorithm, it is rarely the case that the first attempt will be successful. Programming, and software development for all but the simplist of problems, is a rather tedious and error-prone process. Some sort of iterative process is generally used:

1. A precise statement of the problem to be solved is needed. This phase of development is called *analysis*. For large software development projects the analysis is done by a *systems analyst*, who is knowledgable in the problem domain, as well as in programming.

   A data set, called *test data*[2] needs to be created or derived from real data.

2. For large projects the software needs to be divided into smaller, more manageable pieces, called *modules*. This is the *design* phase and is usually done by a *software engineer*. Each module has a clearly defined purpose, with clear expectations of its inputs and outputs, known as an *interface*.

3. Programmmers, working as a team, are each assigned one or more modules. They generally work independently and combine the modules into the final program when all modules are complete.

[1] Open source textbooks for this course based on Java and Python are also available.

[2] The creation of adequate test data is often more time consuming or complex than the programming.

4. The final system is tested using the test data. If the result is not correct, there are three possible courses of action:

   (a) An error in programming implies that the module needs to be fixed.

   (b) An error in design implies that the design needs to be corrected, and all affected modules need to be made consistent with the new design.

   (c) An error in the specifications means that the specifications must be corrected. This implies that the design of the affected modules need to be corrected, and the affected modules all need to be made consistent with the new design.

This process is repeated until the testing indicates that the output or result is correct.[3]

After testing is finished, it is often the case that there are changes to the original specifications. Customers are often unable to predict all of the potential uses and problems with the software. As specifications change, more modules can be added to the system, often without changing existing modules.

5.1.2.1 Program documentation

Large programs can be complicated, unwieldy, and difficult to work with. No matter how carefully we may modularize our software, more effort is needed to clarify and simplify our understanding of a program. Clarification is definitely necessary to understand programs which have been written by other people, but also for programs written by ourselves. *Program documentation* is one way of providing this clarification. Documentation consists of natural language (e.g. English) narrative description(s) of modules and individual statements in a module, to explain their purpose as well as any nuances which are essential to understand and make changes to the program. There are two fundamental kinds of documentation:

- *Internal documentation* consists of narrative descriptions in the code itself. These program lines are often called *comments*, and they serve to elucidate various aspects of the program. Internal documentation has no effect on the actual workings of the program; the computer ignores all comments. However, internal documentation greatly facilitates program maintentance and enhancement.

  In the C++ programming language there are two kinds of comments:

  – A *single-line comment* begins with `//` and continues to the end of the line, as shown in these examples:

```
x = 3 + b      // Add 3 to b, store result in x
//  This is the solution to exercise 3 in chapter 4
```

[3] For large projects, an additional testing step is used: *acceptance testing* involves testing the program at the customer's site with real test data.

– A *multi-line comment* begins with `/*` and ends with `*/`, as shown in this example:

```
x = 3;       /*  x is now 3.
                 x represents the number of
 clients.
*/
```

External documentation consists of narrative descriptions stored separately from the program. This documentation typically describes module-level or system-level components, and does not generally describe individual statements in a program.

Substantial documentation is essential when working collaboratively, as programmers will need to understand the intent of other programmers when writing software to interact with other programmers' software.

5.1.3 Collaboration

When developing large software systems, large teams of programmers are often used to reduce development time. However, even for smaller projects teamwork can be beneficial. When working with teams, the complexity and extent of a problem can be reduced for each team member.

5.1.3.1 Separation of responsibilities

As described above, when the design of a system specifies modules, each member of a team can be assigned responsibility for one (or a few) module(s). With a well designed system, this greatly simplifies the task of each team member. Each team member can work independently and test their work, up to a point. Eventually, modules will have to be merged into a complete system for testing.

5.1.3.2 Working together

Often team members will work together on a particular module. For cases where the implementation of a single module is very difficult, it can be helpful to have two people working on it together. This is often called *pair programming*. In this mode, one member is in the 'driver' seat, i.e. typing code on the keyboard, and the other team member is watching and carefully understanding everything that the first member is typing. They are in constant verbal communication, ensuring that everything is sensible and correct. It might seem that this is an expensive way to develop software, since both team members are paid to develop one module. However, it is often the case that this investment reduces the time spent on debugging and maintenance by more than a factor of 2.

5.1.4 Exercises

1. What is the difference between a program and an algorithm?

2. What are the names of the four steps in the software development process?

3. Describe the difference between internal and external documentation for a software system.

4. How can collaboration reduce the complexity of the software development process?

5.2 Algorithm implementation

Once we have developed an algorithm to solve a particular problem, we generally will wish to *implement* that algorithm by writing a program. The statements in the program will correspond directly to steps of the algorithm. Most programming languages have control structures that correspond directly with the control structures inroduced in chapter 4: sequence, selection, and iteration.

5.2.1 C++ history

The C++ programming language is derived from an older language named C which was developed at AT&T Bell Laboratories in the 1960's. In the late 1980's a new programming paradigm, called *object-oriented* programming was introduced, and the C language was extended, and named C++, to include object-oriented features.[4]

5.2.2 Sequence

As with algorithms, the statements in a program are execured sequentially, in the order in which they appear in the program. An example of a step in an algorithm could be
1. $x \leftarrow 17$
which means that the value 17 should be stored in the variable x. The corresponding statement in the C++ language would be:
`x = 17;`
We read this as "x gets 17" or "x is assigned 17". It should NOT be read as "x equals 17". Though it is true that x equals 17 after this statement has executed, consider the statement:
$x = x + 1;$
which means add one to the value of x, and store the result back into x. Those who read this statement as "x equals x+1" will usually end up in confusion.

In C++, there are several *types* of data. Some examples of data types are shown in Figure 5.1. C++ requires the programmer to declare the type of a variable before using it. So a sequence of C++ statements could be:

```
int x;
x = 73;
```

[4] C++ was developed principally by Bjarne Stroustrup, also at Bell Laboratories.

Name	Meaning	Examples		
int	whole number	17	0	-500
float	number	17.0	-3.1415	6.02e23
char	A single character	'w'	'7'	'$'
bool	A true/false value	true	false	

Figure 5.1: Table showing some of the primitive data types in C++

Operation	Meaning	Example	result
+	addition	3 + 25	28
-	subtraction	3 - 25	-23
*	multiplication	3 * 25	75
/	division of floats	25 / 7.0	3.5714286
/	division of integers	25 / 7	3
%	modulus	25 % 7	4

Figure 5.2: Table showing arithmetic operations in a programming language

```
x = x * 2;
```

The first statement is a *declaration*. It means that the variable x is to store whole numbers only. A variable should be declared just once in any program block; after it is declared it may be used and changed many times. After execution of these statements, the value of the variable x would be 146.

The arithmetic operations in C++ are shown in Fig 5.2. Note that:

- In addition to the four operations introduced in chapter 4, we have an operation to find the *remainder* following the division of integers. The / operator finds the quotient and the % operator finds the remainder, otherwise known as *modulus*. [5]
- In an ambiguous expression such as $30/3+2$ we note that multiplication and division take *precedence* over addition and subtraction. (The result is 12, not 6)
- In an ambiguous expression such as $30-3+2$ we note that when two operations have equal precedence, the leftmost operation is performed first. (The result is 29, not 25)

5.2.3 Selection

The *selection* control structure in C++ is very similar to its algorithmic counterpart. It consists of the following:

[5] We discourage the use of modulus with negative operand(s). The result is not consistent across programming languages and computing platforms.

Comparison operator	Math symbol	Meaning
`==`	$=$	equals
`<`	$<$	is less than
`<=`	$\leq$	is less than or equal to
`>`	$>$	is greater than
`>=`	$\geq$	is greater than or equal to
`!=`	$\neq$	is not equal to

Figure 5.3: Table showing the comparison operators for a programming language

- The keyword `if`. A keyword is a word in a programming language used for one specific purpose.
- A true/false condition inside parentheses. This is known as a *boolean* expression;[6] it is something that is either true or false. An example of a boolean expression is:
  `(x + y < z + 2)`
  The comparison operators which may be used in a boolean expression are shown in Figure 5.3.
- A C++ statement. This is the statement which is executed only if the boolean expression is true.
- The keyword `else`.
- A C++ statement. This is the statement which is executed only if the boolean expression is false.

As with algorithms, the `else` part of an `if` statement is optional.

Some examples of Java statements are:

```
if (x > 0)  y = 3;  else z = 0;

if (x+3 == (y+3)*2 - 1)
   x = z - 31;
```

In the first example the variable `y` is assigned the value 3, only if the value of `x` is greater than 0. There is an `else` part for the `if` statement; the variable `z` is assigned the value 0 only if the value of `x` is not greater than 0.

In the second example, the variable `x` is assigned the value of `z-31`, only if `x + 3 is equal to (y+3)*2 - 1`

Note that, unlike algorithms, a C++ program is *free format*. The line breaks generally have no effect on the program's intent. There may be several statements on one line, or a single statement may be split across several lines. It is the programmer's responsibility to format the program for maximum clarity and readability. We will generally indent nested control structures to enhance readability.

[6] Boolean expression is named after the English logician George Boole.

5.2.3.1 Statement blocks

Another example of an `if` statement is shown below:

```
if (x == 0)
   {  y = y + 3;
      x = 1;
   }               // if statement ends here
```

In this example the consequence of the `if` is a *block* of statements, enclosed in curly braces. If the value of `x` is 0, both statements in the block are executed; if the value of `x` is not 0, neither of the statements in the block is executed.[7] We also included a single line comment clarifying the end of the `if` statement.

Note that any statement in an `if` statement may itself be an `if` statement; thus, `if` statements may be nested, as they were with algorithms.

5.2.4 Iteration

An iteration in C++ may be implemented with the key word `while`. It corresponds to the `repeat` instruction of an algorithm. An iteration in C++, often called a *loop*, consists of:

- The key word `while`
- A true/false condition inside parentheses. This is the condition which determines whether the iteration is to continue. If the condition is false, the iteration terminates. The format of the condition is exactly the same as the format of the condition in an `if` statement.
- A C++ statement. This is the statement which is executed repeatedly as long the boolean expression is true. This statement may be a block of statements; it is known as the *body* of the loop.

Some examples of C++ iterations are:

```
while (x > 0)
   {  y = y * 2;
      x = x - 1;
   }

while (x <= y)
   {  x = x + 1;
      y = y - 2;
   }
```

We make a few points on iterations:

[7] Many authors recommend using the curly braces even when they are not needed, i.e. for a block consisting of one statement.

- There should be at least one statement in the body of the loop which has an impact on the condition. In the first example, x is decremented by 1 on each iteration, thus x must eventually reach 0, terminating the iteration. If the iteration never terminates, it known as an *infinite loop*, which is generally an error.

- If the condition is initially false, the loop repeats 0 times.[8] This is often a desirable feature.

Note that any statement in the body of the loop may be an `if` statement or a `while` statement. As with algorithms, the control structures may be nested to any depth.

We conclude this section with an example of a C++ program that determines whether a given positive integer is a prime number. An integer greater than 1 is prime if it is not divisible by any integers other than 1 and itself. The following is a list of some small prime nubmers:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, ...

To solve this problem, we will use the C++ modulo operator. It is denoted with a percent symbol: %[9] If x and y are integers, then x % y is the remainder produced when x is divided by y. For example, 17 % 3 is 2.

In the C++ code shown below,[10] it is assumed that the variable `x` has been assigned some positive integer value.

```
// Determine whether the value of x is prime.
int i;
i = 2;
while (i < x)
  if (x % i == 0)
        // terminate, x is not prime.
  else
        i = i + 1;
// terminate, x is prime.
```

Since we have not yet learned how to terminate a program, we indicate termination, and the result, with comments.

In this program there is only one statement in the body of the loop; it is an `if` statement (with an `else` part); consequently curly braces were not needed to enclose the body of the loop.

5.2.5 Exercises

1. Show C++ statements to perform each of the following:

   (a) Declare w and z to be variables, each of which stores an integer.

[8] The `while` statement is a pre-test loop. C++ also has a post-test loop.
[9] This operator has nothing to do with percentages.
[10] There are more efficient ways to test for primality.

(b) Store the value 18 in the variable z.

(c) Store the value -3 in the variable w.

(d) Store the sum of w and z into the variable z.

(e) Store the value 16 into w, only if w is positive.

(f) Store the value 6 into w, if the sum of w and z is negative, and store the product of w and z into z if the sum of w and z is not negative.

(g) Find the sum of all the integers from 1 to 100 and store into the variable `sum`.
Hint: Declare a variable which is increased by 1 in each iteration.

(h) Find the sum of all the integers from 555 to 1000, and store into the variable `sum`.

5.3 Program abstractions

In chapter 4 we described the process of combining algorithms, i.e. invoking an algorithm from another algorithm. This is an example of *program abstraction*. We can develop software tools, and use those tools to build more complex tools, which are in turn used to build even more complex and powerful tools. This is how technology evolves.

In various programming languages many different words are used for the notion of a portion of a program that can be invoked from other programs:
`subprogram, subroutine, procedure, function, method`

In C++, the appropriate term is *function*. A C++ function definition consists of:

- A return type, describing the type of the data which is to be the result of the function, if any. For a function which returns a whole number, the return type would be `int`. For a function which returns a true/false value, the return type would be `bool`. For a function which has no explicit result, the return type is the key word `void`.
- The function name[11]
- A list of parameter types and names, in parentheses, and separated by commas. The parentheses are required, even if the function has no parameters.
- The function body, consisting of a block of C++ statements with sequence, selection, and iteration control structures.[12]

[11] Conventionally, the function name usually begins with a lower case letter.

[12] The curly braces are required, even if there is only one statement in the body of the function.

As an example of a C++ function, we will define a function named `sumOfSquares` which has two parameters, both whole numbers. It should return the sum of the squares of those two numbers. For example, if the parameter values are 3 and 4, the value returned should be 25 because $3^2 + 4^2 = 9 + 16 = 25$. The function definition is shown below:

```
// Return the sum of the squares of the two parameters
int sumOfSquares(int x, int y)
{  return x*x + y*y;  }
```

Mathematicians know from the Pythagorean theorem that the length of the hypotenuse of a right triangle is equal to the square root of the sum of the squares of the two legs. Thus three whole numbers a, b, and c are called a *pythagorean triple* if $a^2 + b^2 = c^2$. An example of a pythagorean triple is {3,4,5}.

We now wish to define a function with three parameters which will determine whether three whole numbers form a pythagorean triple. We will use our previous function `sumOfSquares` in this function:

```
// Return true only if the values of a,b,c form
//   a pythagorean triple.
// Assume c is the largest value.
bool pythagoreanTriple (int a, int b, int c)
{  if (sumOfSquares(a,b) == c * c)
       return true;
   else
       return false;
}
```

5.3.1 Reducing complexity with abstractions

Program abstraction is a way of hiding details. We can develop a program segment (such as a C++ function), test it, and convince ourselves that it is correct. We can then make use of that program segment in other programs, simply by invoking it when necessary. As long as we understand the constraints on the parameters, known as *preconditions* and the expected result of the functions, we can use it without concerning ourselves with all the details of its inner workings. That is the beauty of program abstraction.

5.3.1.1 Data abstraction

In chapter 2 we learned that all data is binary: a sequence of 1's and 0's. We then can interpret that binary data in various ways. This provides a means of hiding the details (the 0's and 1's) and viewing the data in a way that is easier to understand. This is known as *data abstraction*.

In the C++ language there are several primitive data types provided, as shown in Figure 5.1. Each of these actually is represented by a binary sequence. For example, we use the number 17. Internally it is represented

Type	Value	Represented by
int	37	00000000000000000000000100101
char	'w'	01110111
boolean	false	0
boolean	true	1

Figure 5.4: Examples of bit sequences representing primitive data types in a programming language

by 00000000000000000000000000010001.[13] Other examples are shown in Figure 5.4.

This is the first level of data abstraction.

At the next level, we can use the primitive data types to define more useful data types, in which the primitive values are hidden to reduce complexity. Here are a few examples:

- Strings of characters. We often wish to deal with several characters as a self-contained data item, known as a *character string*, or simply *string*. In some programming languages this is provided as a primitive type; in C++ it is defined in a software library installed with the C++ compiler Strings are normally implemented as a sequence of characters. When scanning this sequence of characters, it is necessary to determine when the end of the string is reached. In some languages the length of the string is stored with the characters. In the C language (predecessor to C++) there is a terminating character, known as a *null* character.[14]

- Rational numbers. When working with numbers that are not necessarily whole numbers, we can use a *floating point* data type, in which we allow for values like 1.5. However, for the result of the division 1.0 / 3.0, for example, we will never get a perfect result. There are many numbers that cannot be represented perfectly. This justifies the need for a defined type, which could be called `Rational` or `Fraction`, in which each value is represented by two integers: a numerator and a denominator. Thus the value 1/3 would be represented by the pair (1,3). Arithmetic operations on Rational numbers can provide perfect representations, where floating point representations can only provide approximations.

At the next level, we are working in a particular problem domain, and wish to perform further data abstractions. If we are working with an application at an educational institution where we need to store information about the students, this is an ideal opportunity for data abstraction.

[13] In C++ an int could consist of 32 bits, but on a different platform a C++ int could be 64 bits. Consequently C++ programs are not necessarily portable from one computer or operating system to another.

[14] This design for strings has made the C languages vulnerable to a security breach known as a "buffer over-run".

We could define a new data type called `Student`. Each Student would be composed of several items, called *fields* or *member data*.[15] Some of the fields defining a Student could be:

- The Student's name: A string
- The number of credits completed: an integer
- The Student's grade point average: A floating point number, storing an approximation, or a Rational (if the programmer had defined a Rational type) storing an exact value.
- The Student's address: A defined type, Address
- Courses: A vector[16] of the Courses in which the Student is currently enrolled, where Course would be another defined type

In order for this to work we would need defined types for vector, Address and Course, each of which would be composed of several fields.

In this manner we continue building new data types, using existing data types. This is data abstraction.

5.3.1.2 String processing

Here we elaborate on character strings. With this data type there are several operations which could be defined:

- Compare two strings to see which comes first, alphabetically. In this operation the ascii codes of the characters are used, because they are in alphabetic order.
- Make a copy of a given string.
- Search a string for a given substring. For example, searching the string `"Princetonian"` for the substring `"ton"` would produce the value 6, because `"ton"` begins at position 6[17] in `"Princetonian"`
- Extract a substring from a given string. This operation usually requires a string, a start position in the string, and a length for the substring to be extracted. For example, `substring("Princetonian", 2,3)` would produce the string `"inc"`.

[15] In purely object-oriented languages fields are known as *instance variables*.
[16] We discuss vectors below
[17] Position numbers begin at 0, in most programming languages.

Number	Can be coded as		
37	37.0	3.7E1	0.37E2
-43.06	-43.06	-4.306E1	-4306E-2
6.02×10^{23}	6.02E23	602E21	0.602E24
1×10^{-12}	1.0E-12	0.1E-13	1000E-9

Figure 5.5: Table showing some examples of floating point constants in a programming language

5.3.1.3 Numbers

We have seen that Rational numbers can be implemented using only integers. The same is true of floating point numbers. These are numbers which need not be whole numbers. We also include here numbers that are very large, or very close to 0. Most programming languages allow these numbers to be specified with decimal points and/or scientific notation, in which we include an exponent of 10. Because most programming languages do not permit superscripts in the program code, the letter E (or e) is used to indicate an exponent of 10. Figure 5.5 shows some examples of floating point values in C++.

Much can be learned by examining the construction of floating point numbers. These numbers an be constructed using only integers; a floating point number consists of only two integers:

- A signed mantissa: The digits in the number, excluding the exponent
- A signed exponent: An exponent of 10[18]

Storing only these two values, any floating point number can be constructed.

5.3.1.4 Vectors (and arrays)

A vector consists of a homogeneous sequence of values. When defining a vector you need to decide on the type of values it will contain (specified in angle brackets), and you should specify an intial size for the vector. To create a vector of integers named 'grades':
`vector<` *int* `> grades;`
To create a vector of integers named 'grades' with an initial size of 100:[19]
`vector<` *int* `> grades(100);`

As with strings, the positions in a vector begin with 0, so the vector of grades shown above would have positions 0..99. To store a value into a particular position of the vector, or to access the value at a particular position, use square brackets for the position:[20]

```
grades[0] =  99;         // position 0 now stores 99
```

[18] In actual floating point implementations the exponent is usually an exponent of 2 or of 16, because the mantissa is binary.
[19] Vectors are more efficient when created with an initial size
[20] The position is also called an 'index' or 'subscript'.

```
/* Function to return the average value of a vector
 * of double precision floating point numbers.
 * Pre: The vector is not empty
 */
double average(vector<double> v)
{
   double result = 0; // sum the values
   int i = 0;                    // loop counter
   while (i < v.size())
     { result = result+v[i];
       i = i + 1;
     }
   return result / v.size();    // return the average
}
```

Figure 5.6: A C++ function to find the average of a vector of double precision floating point numbers.

```
cout << grades[0];       // print the value at position 0
```

Vectors can grow and shrink. To add a value at the end of the vector, use the function `push_back`:

```
grades.push_back(88);     // append the value 88 at the end
```

This will increase the size of the vector, which can be accessed via the `size()` function:

```
cout << grades.size();   // print the size of the vector
```

Vectors can be passed to functions as parameters. Figure 5.6 shows an example of a function which will return the average value from a vector of double precision floating point values.

Arrays are similar to vectors, but the size of an array cannot be changed as the program executes.[21] Usage of arrays is not recommended; anything that can be done with an array can be done just as well, and more safely, with a vector.

5.3.1.5 Application Program Interface

Suppose we write a program named P1, which can perform several useful calculations. Another person writes a program named P2 which makes use of program P1 to perform those calculations. Program P1 is known as a *server* because it

[21] Arrays predate vectors and are used to implement vectors.

is providing a service. Program P2 is known as a *client* because it needs the services offered by a server. This is known as *client/server* terminology.

An *Application Program Interface* (API) is another form of program abstraction. An API generally accompanies a server, and provides all the information that a client would need (and no more) in order to make use of the services. The API would specify not only all the services that are available, but would also provide a description of what information they would need (parameters), what restrictions there may be on that information, and in what circumstances the server might be unable to perform the desired service. The details of how the services are performed are *not* part of the API. These details are hidden from the client, thus this is an example of abstraction.

5.3.1.6 Software libraries

Once we see the need for certain functionality, to be used in many different applications, we can develop a repository, or *library* of software modules to be used when needed. In C++ the API of various libraries are defined as *header* files in what is known as the Standard Template Library (STL). Components of this library include:

- C library - Utility functions inherited from the original C language
- Containers - Data structures such as queue, map, vector, set, stack, etc.
- Input/output - Software superceding the iostream header files from the C library
- Threads - Software allowing the programmer to define parallel processes
- Iterators - Software used to visit every member of a data structure, such as a vector or a set

Libraries are a fundamental example of program abstraction. The libraries have been well tested. They have clearly defined interfaces, i.e. APIs which explain exactly how they are to be used, what they do, and what, if any, side effects they may have. Once the APIs are produced, the libraries may be used without looking inside the modules at the details of their implementation. Libraries are helpul in preventing a developer from 'reinventing the wheel'.[22]

5.3.2 Exercises

1. Show a C++ function which evaluates the function:
   $f(x) = x^3 + 3x^2 - 5x + 2$
   `int f(int x)`

[22] To build its Android operating system for smartphones, Google made extensive use of the API for Java's class library, for which the copyright was held by the Oracle Corp. Oracle filed a lawsuit against Google for about $1B, claiming copyright infringement. After several appeals, the U.S. Supreme Court recently ruled in favor of Google, since Google was using the API, and not the *implementation* of that API, and this constituted *fair use* of copyrighted material.

2. Show a C++ function which evaluates the absolute value function.
   $abs(25) = 25$
   $abs(-3) = 3$
   $abs(0) = 0$
   `int abs(int x)`

3. Show a C++ function which finds the sum of all the integers from 1 to a given max value.
   `int sum(int max)`

4. Show an algorithm which can be used to multiply two numbers represented as Rational. Each number has a numerator and a denominator, both of which are integers.
   `Rational mult(Rational x, Rational y)`
   Some tips:

   - If `r` is a Rational number, you can refer to its numerator as `r.num`
   - If `r` is a Rational number, you can refer to its denominator as `r.denom`
   - To create a new Rational number, representing 5, in a variable named `r`:
     `Rational r(5,1);`

5. In this section we defined a data type, `Student`, with 5 fields. Define the following data types, which are used by the `Student` data type:

   (a) `Address`

   (b) `Course`

6. Show an algorithm which can compare two strings of characters to determine which comes first alphabetically.
   `strCmp(s1,s2)`
   Some tips:

   - The i^{th} character of a string s is s_i, where the first character is s_0.
   - Characters can be added and subtracted (It is actually done with ther ASCII codes.) `d - b = 2`
   - The length of a string, `s`, can be obtained with `s.length()`.

7. Show an algorithm which can extract a substring from a given string.
   `substring(s,start,length)`
   Some tips:

   - A string of length 0 is "".
   - Two strings can be concatenated with a raised dot:
     $john \cdot son = johnson$

8. Show an algorithm which can be used to multiply two numbers which are in floating point representation. Each number has a mantissa, and an exponent of 10, both of which are integers.
   `mult(x,y)`
   Some tips:
   - The mantissa of a floating point number, x, is x_{mant}.
   - The exponent of a floating point number, x, is x_{exp}.
9. Show a C++ function to find the position of the largest value in a given vector of numbers. The position of the first value is 0. The position of the last value is one less than the size of the vector. Your method should return -1 if the vector is empty.
   `int getLargest(vector<int> numbers)`
   Some tips:
   - The declaration `vector<int> numbers` means that the vector may contain nothing other than integers.
   - The size of `myVector` is `myVector.size()`
   - Check for an empty list first (size = 0)
   - Use the first value as the largest, search the list for larger values, replacing the largest as larger values are found.

5.4 Program Development and Maintenance

The software development process has been studied and refined over the years. For large projects, a sequence of steps can be used, as shown in the section on Problem solution (in section 1 of this chapter). Referring to those steps:

- If an error is detected in step 3 (unit testing), the error is corrected in that unit and testing continues.
- If an error is detected in step 4 (system testing), the mistake may be in the design, in which case the design error is corrected, affected modules are modified, and the relevant unit tests are repeated.
- If an error is detected in acceptance testing, the mistake may be in the analysis phase, in which case the system requirements need to be corrected, affected modules need to be redesigned, and affected units need to be modified and retested.

5.4.1 Program correctness

How can we verify that a program is correct and will always produce correct output, according to the original specifications? There is no simple answer to this question. Here we provide some guidelines that will often lead to program correctness.

5.4.1.1 Programming style

Programs should be written in a style which is readable, and easy to understand. Most programming languages are free format, meaning that white space (spaces, tabs, newlines) in the code is ignored; white space has no impact on the program's functionality. This can lead to code with poor style, such as:

```
   // Determine whether the value of x is prime.
   int
i; i =
2;
   while (    i <
x)
     if (x % i == 0) { // terminate, x is not prime.
           i = i + 1; }
   // terminate, x is prime.
```

As far as the compiler is concerned, this is a perfectly good program to determine whether a given integer is prime. However, it is extremely difficult to read. It can be formatted to read as shown below for clarity:

```
   // Determine whether the value of x is prime.
   int i;
   i = 2;
   while (i < x)
     if (x % i == 0)
       {
           // terminate, x is not prime.
           i = i + 1;
       }
   // terminate, x is prime.
```

Also notice that control structures are clearly indented, as we did in chapter 4. Some programming languages (Python) *require* this indentation.

Appropriate comments in the code also help a reader to understand how a program is supposed to work.

5.4.1.2 Meaningful names

The programmer must choose names for the variables, procedures, etc. in a program. Choosing meaningful names greatly improves the readability of a program. For example, the name of the C++ function shown above could be `foo`. It is much better to give it a meaningful name:
`bool isPrime(int x)`
That function uses a variable `i`, and determines whether the given integer is divisible by `i`. It would have been better to name that variable `factor`, because we are really trying to determine whether it is a factor of the given value, `x`.

5.4.1.3 Duplicated code

It is often the case that some sequence of statements in a program needs to be inserted at several different places in a program. This is what is known as *duplicated code*, which is not advisable. If the same sequence of statements occurs at 10 different places in a program, what happens if there is an error in the duplicated code? What happens if the program specifications change (as is often the case)? The code would have to be fixed in all 10 places that it was inserted. In large programs this can be a huge job.

Rather than duplicating the code, it is better to isolate it in one place, such as a function. Then whenever it is needed the function is called. Now if a fix is needed, there is only one place that needs fixing. Sometimes this is called *factoring* duplicated code.

For example, if we needed to test numbers for primality at several places in a program, all we need to do is call the `isPrime` function shown above, rather than duplicating the code everywhere it is needed.

5.4.1.4 Strive for short program segments

As we write larger programs, it becomes clear that the complexity of these programs increases to the point that verifying correctness becomes extremely difficult. For that reason we will divide a program into shorter segments, oftem called *code segments*. This can be done with abstraction, as we write procedures (C++ functions) to perform various services, and then use those services as needed. Each function can be tested individually. If this is done properly, each function will not be excessively complicated, and the entire program will be manageable.

5.4.1.5 Debugging

It is rare that a program of reasonable size will perform correctly the first time it is executed. The flaws, or errors in a program, are often called *bugs*.[23] The process of locating, identifying, and correcting these flaws is called *debugging*. There are a few different kinds of bugs:

- Compile-time bugs: These are errors which are often called *syntax* errors. The compiler[24] can provide diagnostic information on these bugs, such as the exact location in your program, and a description of the error.[25].

  An example of a syntax error would be:

  ```
  x = (2+3)*y);
  ```

  The parentheses are not correctly balanced.

[23] A moth that had flown into the circuitry of an early computer in the 1950's caused the machine to fail, hence the origin of the term *bug*.

[24] A *compiler* is the program translates your Java program to a form that can be directly executed by the computer's CPU.

[25] Some software can even offer suggestions on how to correct the error

```
int x,y,z;
y = 17;              // Error is here; y should be 27.

. . .                // much intervening code here,
                     // not affecting y.

x = 12;
z = x + y;
z = (z-x) / (y-17); // Exception occurs here,
                     // division by 0.
```

Figure 5.7: The location of an Exception and the location of the program flaw which caused the Exception can be far apart

- Run-time bugs: These are errors which occur after the program has been compiled, and it has started executing. These bugs will usually result in one of the following:
  - An Exception: Something happened which caused the program to terminate abnormally. For example, the program attempted to divide a number by 0.
  - The output which was produced is incorrect.
  - The program continues to execute, never terminating. This is called an *infinite loop*.

  All three of these errors are the result of incorrect statements in the program, known as *logic errors*. These errors are very difficult to fix because the logic error could cause the program to fail after many more statements are executed, i.e. the logic error and the location at which evidence of an error appears can be far apart from each other, as shown in Figure 5.7. In Figure 5.7. the value of `y` was incorrectly set to 17 (it should have been 27). When execution reached `z = (z-x) / (y-17)`, the value of `y` was still 17, and the divisor, `y-17` was 0, causing an Exception known as an Arithmetic Exception. The program then terminates before reaching a normal termination.

  The debugging process involves a diagnostic process during which it is necessary to:

  1. Understand the Exception or incorrect output.
  2. Find the location of the error.
  3. Understand the effect of the error on the progam's execution.
  4. Correct the error and test the program again.

  The debugging process is so important, and so difficult, that software called a *debugger* has been developed to aid in the process of debugging

run-time errors. The debugger can help determine the location of the actual error, and can help the programmer understand flaws in the program, but the debugger cannot fix the error; that task is left for the programmer. Most debuggers are capable of:

- Setting breakpoints in a program. When execution reaches a breakpoint, there is a pause until the debugger is given a command
- Showing the values of variables as the program executes
- Stepping through the statements in the program one at a time
- Stepping into a called function, versus running the called function at full speed
- Continue executing at full speed until the next breakpoint is encountered

In summary, the debugger provides a visual display of the program's *state* during execution, which is very helpful to the programmer.

5.4.1.6 Overall purpose of the programm

In order to test the program, and ensure its correctness the programmer must be aware of the program's purpose. A program has an *input/output relation*. For every input there is an expected output, and if the programmer does know what the expected output should be, the programmer will not be able to debug the program.

5.4.1.7 Use cases

Examples of certain inputs, and their corresponding desired outputs are called *use cases*. These use cases are provided to the programmer to help reach an understanding of the overal purpose of the program, and to aid in the testing process.

5.4.1.8 Is the program correct?

Complex programs undergo extensive testing to determine correctness. However, the set of possible inputs is so huge, that it is usually not possible to test all possible inputs. In particular, software which interacts with the outside world will be very difficult to test for correctness.[26]

Much research has been conducted in the area of program correctness. There are algorithms which are designed to prove that a program is correct. However, in order for this to be effective, the input/output relation must be specified precisely, and in a form that the proving algorithm can utilize. Programs which interact with the world, such as the software in an airplane's cockpit cannot

[26] Consider, for example, the software which aids an airplane pilot. It utilizes information such as air speed, altitude, the plane's orientation, etc. to help the pilot fly the plane. This software is difficult to test for all possible situations, and a software error can be catastrophic.

be proven correct. Another criticism of correctness algorithms: After you have proven a program to be correct, how do you know that your proof is correct?

Many programming shops require the programmers to provide logical explanations and justification for the correctness of their programs, in addition to extensive testing.

When attempting to verify program correctness, it is helpful if the program has been divided into smaller modules, or functions. If each of these smaller functions is correct, and they are used correctly, it is more likely that the entire program is correct. If any one unit is not correct, the entire program is considered incorrect.

5.4.1.9 Code review

In determining correctness testing is important. But there is another technique, called *code review* which is also used. Code review involves reading through the statements in a program and providing an explanation or justification for each statement. Often code review is done in pairs, or groups, of people. The original programmer provides the explanations and the others listen critically, attempting to point out fallacies in the explanations or code which does not agree with the explanations.

5.4.1.10 Program functionality

The function, or purpose, of a program can be described as an input/output relation. Often this is not possible or not sufficient. In such cases the functionality is described by how a user interacts with the program, or by how the program interacts with the world outside.. The functionality is generally described from the user's perspective; it would not be likely to be described by internal details, such as program variables and statements.

5.4.2 Exercises

1. Rewrite the C++ code segment shown below so that it is clear and readable. Be sure to indent control structures properly.

```
        if (x >
        17
) if (y < x) { while (x > 0) x = x   -1
        ; y = y * 3; } b =
        0
        ;
```

2. The code segment shown below is supposed to find the average of a vector of floating point numbers (`myVector`). Rewrite this code segment using meaningful variable names. Assume the vector is not empty. Be sure to

indent control structures properly. Also include comments to describe the internal intention of various variables and/or statements. [27]

```
float x1 = 0.0;
float x3;
int x2 = 0;
while (x2 < myVector.size())
   {
x1 = x1 + myVector[x2];
      x2 = x2 + 1;
    }
x3 = x1 / myVector.size();
```

3. Eliminate the duplicated code from the following code segment, without changing the intent of the code segment.

```
if (x > 0)
   { y = 17;
     x = x + 1;
   }
else
   { y = 17;
     x = 0;
   }
```

4. Eliminate the duplicated code from the following code segment, without changing the intent of the code segment.

```
while (x > 0)
   { y = 17 * a - 4;
     z = x*3;
     x = x + 1;
   }
```

5. Reduce the size of the code segment shown below without changing its intent. It is supposed to find the average values of three vectors of floating point numbers: `numbers`, `grades`, and `scores`.

```
float averageNumber, averageGrade, averageScore;
float sum = 0.0;
int i = 0;
while (i < numbers.size())
  {
```

[27] If you were to run this code on a computer, you would change all the declarations of `float` (single precision floating point) to `double` (double precision floating point) to avoid error messages from the compiler.

```
            sum = sum + numbers[i];
            i  = i + 1;
          }
        averageNumber = sum / numbers.size();

        float add = 0.0;
        int ctr=0;
        while (ctr < grades.size())
          {
add = add + grades[ctr];
            ctr = ctr + 1;
          }
        averageGrade  = add / grades.size();

        float total = 0.0;
        ctr = 0;
        while (ctr < scores.size())
          { total = total + scores[ctr];
            ctr = ctr + 1;
          }
        averageScore  = total / scores.size();
```

Hint: Use abstraction. To pass a vector to a function, provide the name of the vector as the actual parameter. In the function declare a parameter as a vector.

6. True or False:

   (a) Debuggers can help you fix syntax errors.

   (b) The compiler will find run-time errors.

   (c) After locating the source of a run-time error, the Debugger will correct the error for you.

   (d) Code review means the same thing as testing.

   (e) Use cases are valuable when testing a program.

   (f) All software can be verified for correctness, thus eliminating the possibility of failures when the software has been released for general use.

5.5 Programming with Mathematics and Logic

5.5.1 Using mathematics and logic

The computer's memory consists of a linear sequence of 1's and 0's. Though we may place various interpretations on this sequence of bits, the concept of a number is fundamental in computer science.

5.5.1.1 Working with numbers

We have seen in chapter 2 how to represent positive or negative whole numbers as binary values. We also saw that whole numbers can be used to represent non-whole numbers, very large numbers, and numbers very close to zero using a floating point representation, with a mantissa and an exponent.

We repeat that there are inevitable approximations and inaccuracies with floating point numbers.[28]

Once we have a representation scheme for various kinds of numbers, fundamental operations such as addition, subtraction, multiplication, and division can be defined (usually done with hardware). The numbers and operations can be used to define more interesting numbers, such as rational numbers, complex numbers, or numbers with unlimited precision.[29]

Most programming languages permit the programmer to specify calculations algebraically. I.e. an expression such as `(a+b)-(c*d)` written in a program is actually translated by the compiler to a series of three operations by the computer:

1. Add `a+b`, leaving the result in a temporary memory location, `T1`
2. Multiply `c*d`, leaving the result in another temporary memory location, `T2`[30]
3. Subtract `T1-T2`, leaving the result in yet another temporary memory location, `T3`

Numbers and numerical computations are fundamental to everything we do with computers. Even applications such as character string processing, graphics images, sound clips, and video formats require extensive numerical calculations.

5.5.1.2 Working with logic

Logical concepts are everywhere in hardware design[31], software, and computer science. We often work with an algebra similar to ordinary algebra, called *boolean* algebra. With boolean algebra the only values are true (i.e. 1) and false (i.e. 0). In chapter 2 we studied the boolean operators which are used in boolean algebra (`AND`, `OR`, `NOT`, and `XOR`).

A good understanding of boolean algebra is essential when programming the computer. For example, a C++ programmer might code:

```
if (a != b && (c == 0 || a != b))
```

and realize that it can be simplified to:

```
if (a != b)
```

[28] In many programming languages $0.1 + 0.1 + 0.1 \neq 0.3$ because of these inherent inaccuracies.

[29] Data types in many programming languages have limited precision; there is a maximum value and a minimum value for a Java int, for example. In C++ the maximum and minimum values depend on the hardware/operating system platform.

[30] The first two steps could be executed in either order, or in parallel.

[31] Hardware design is often called *logic* design.

according to a well-known boolean identity:

```
x && (y || x)  =  x
```

where `x` and `y` are boolean variables (i.e. they can store true/false values).

Logic concepts are also useful when reasoning about a program's correctness. We often make assertions, either in comments or in the code, about the state of a program, values of variables, etc. For example, if we had the statement:

```
x = (y+z)/w
```

we might want to be sure that w is not 0, which could cause the program to terminate with an error. With a series of steps in this reasoning process, we could assert: $w \neq 0$. Many programming languages allow for this kind of `assertion`, which would alert the programmer when the assertion is false (early detection of errors is always desirable).

5.5.1.3 Working with abstraction

As we saw in chapter 2, abstractions are everywhere in computer science. A fundamental property common to all computers is the notion of an *array* of values, which are stored in contiguous memory locations. Arrays provide for quick and direct access to any of the values.

Using the fundamental properties of arrays, binary integers, and addresses,[32], we can employ a series of abstractions to develop more powerful and efficient structures:

1. Using integers, arrays, and addresses, we can develop lists (or vectors), which can grow and shrink as a program executes.

2. Using lists, we can develop *stacks* (last-in, first-out lists).

3. Using lists, we can develop *queues* (first-in, first-out lists).

4. Using lists, we can develop *hashtables* providing quick access to large quantities of data.

5. Using addresses, we can develop *trees* providing quick access to large quantities of data.

6. Using either hashtables or trees, we can develop *sets*, in which there are no duplicate values, and which provide quick access to data.

7. Using either hashtables or trees, we can develop *maps*, also known as *dictionaries*, which allow for quick access to data.

And the list goes on... We are constantly building new tools using existing tools. With all the tools described above we develop efficient ways of extracting what we want, and possibly *iterating* over the data, i.e. extracting all of the values, sequentially.

[32]The location of a value in memory is known as its *address*.

5.5.2 Exercises

1. If you were to write a C++ program to average a vector of integers, you would need to find the sum of the integers, then divide the sum by the size of the vector (i.e. the number of integers in the vector).

   (a) Should the variable storing the sum of the integers be declared as an `int`, or as a `double` (i.e. floating point)?

   (b) Should the variable storing the average of the integers be declared as an `int`, or as a `double` (i.e. floating point)?

2. Explain briefly how a program designed to compare strings of characters might make use of integer arithmetic.

3. If `x` and `y` are boolean variables, they can store true/false values. Prove that the boolean expression
   ```
   x && (y || x)
   ```
   simplifies to the equivalent expression
   ```
   x
   ```
   Hint: Show the value of the given expression for every possible assignment of values to `x` and `y`. This is called a *truth table*.

4. When abstraction is used to build new tools from existing tools, the inner workings and details of the tools being used should be hidden from the client. Explain briefly why they should be hidden.

5.6 Hands-on programming: C++ from the command line$^{\otimes}$

This section will help you get started with hands-on programming in C++. For a more complete description there are several textbooks available for purchase.[33]

5.6.1 Starting up: A main method

In C++ execution begins with a main function:
```
int main()
```
This function may call other functions. When it terminates, the program execution is finished.

To execute a program from your computer's command line,[34] it must first be compiled.[35] The resulting output of the compiler, a machine language version of the program, which is named `a.out` by default, can then be executed.

For example, if the name of your source program file is `example.cpp`, use this two-step process:

[33] See for example *Big C++* by Horstmann and Budd.

[34] A command line session can be started on MacOS with a `terminal` application. On a Windows PC use the `CMD` application.

[35] Many compilers for C++ are available. I recommend the gnu compiler, which is g++.

1. g++ example.cpp
2. a.out

The first step compiles the program, and the second step executes the program. If there are error messages from the first step, correct the errors before attempting to execute the program.

5.6.2 A complete C++ program

In this section we show an example of a complete C++ program. The purpose of the program is to prompt the user for a series of student names, and a test score for each student. After all the names and scores have been entered, the program will put out the name and score of the student with the best test score. The program is shown in Figure 5.8.

5.6.3 Exercises

1. What will be the name of the output file when compiling the bestStudent program?
2. True or False: If the compiler produces no error messages, your program must be correct.
3. Revise the bestStudent program shown in Figure 5.8 so that it also puts out the name and score of the student with the lowest score. If two or more students are tied for the best or worst score, any one of those students is an acceptable result.
4. Revise the bestStudent program shown in Figure 5.8 so that it also puts out the average score for the these students.
5. Revise the bestStudent program shown in Figure 5.8 so that it also puts out the names of all students who have above-average scores.[36]

[36] In the fictitious town of Lake Woebegone, Minn, all the students are above average.

```
#include <iostream>
using namespace std;

// This program prompts the user to enter the names
// of one or more students, along with a test score for each student.
// It will put out the name, and score of the best student.
int main()
{
    string name, bestName;
    int score, best;

    cout << "Enter a name " << endl;
    cin >> name;
    bestName = name;         // First student is best
    cout << "Enter a score for "<< bestName << endl;
    cin >> score;
    best = score;

    while (name != "$")
    {    if (score > best)    // Better than best seen thus far?
        {   best = score;
            bestName = name;
        }
        cout << "Enter a name, or $ to terminate " << endl;
        cin >> name;
        if (name != "$")
        {   cout << "Enter a score for "<< name << endl;
            cin >> score;
        }
    }
    cout << "Best student is " << bestName << ", score is " <<
        best << endl;
}
```

Figure 5.8: An example of a complete C++ program. It will put out the name and test score of the student with the best score, of one or more students entered at the prompt.

Chapter 6

The Internet

The internet pervades modern computing; the internet and systems built on it have had a profound impact on society. In this chapter we will take a look at some of the design principles used to establish the internet, and we will see how those principles have allowed the internet to perfrom well even as it has grown to accommodate a huge amount of traffic. We will also examine some issues of security, which enable us to use the internet for confidential and/or sensitive transactions. Not only consumer finance, but also government, military, transportaion, and scientific research applications rely on the internet for security.

6.1 Brief history

The internet's prototype was a network of mainframe computers at sites (mostly university research and DOD sites) funded by the Department of Defense Advanced Research Project Agency (ARPA). It was known as ARPANET and existed as early as 1969. Another network, known as CSNET, was a connection of the computers of various universities' computer science departments. There was also a consortium of universities connected with BITNET at about the same time. In 1982 with the specification of a standard communication protocol, Transmission Control Protocol/Internet Protocol (TCP/IP), these networks were able to communicate with each other, forming a network of networks, and the internet was born.

6.2 A Network of Autonomous Systems

Here we attempt answers to some questions about the internet:

- What is the internet?
- What are the components of the internet?

- How do the characteristics of the internet impact the systems which are hosted by the internet?

6.2.1 How the internet functions

6.2.1.1 World-wide connection

As a network of networks, the internet is able to connect devices and networks world-wide. Any device which has TCP/IP software can attach to the internet. Any foreign network of devices which may or may not have the TCP/IP software may connect to the internet as long as the foreign network's host computer has TCP/IP software; the foreign network's host computer can act as a gateway to the internet for non-compliant devices.

6.2.1.2 End-to-end architecture

Most devices connect to the internet by using TCP/IP software, rather than by connecting through a gateway network. The trend is toward usage of TCP/IP around the world. This allows for greater efficiency and compatibility. Because new devices are always being created, they are designed to be TCP/IP compatible from the start, and can communicate with all other devices through TCP/IP.

6.2.1.3 IP addresses

Each device which is directly connected to the internet is assigned a unique number, known as its *IP Address*. IP addresses were originally 32 bits, but because of the amazingly rapid expansion of the internet, IP addresses were extended to 128 bits in 1998 with version 6 of IP (IPV6). Thus, the internet architecture is evolving to accommodate growth.[1]

6.2.1.4 Domain names

Because internet users generally do not wish to communicate with numeric IP addresses, a naming system has been established, to make the specification of an internet device more convenient. This naming system is called the *Domain Name System* (DNS). Internet software is capable of translating between IP addresses and their associated domain names. For example, you can attempt to visit 'www.Amazon.com', which is then translated to the IP address of Amazon, to establish a connection.

Domain names form a *hierachy* with multiple levels. For example, the following domain names can be viewed as being composed of several levels, in a structure that we call a *tree*.[2]

[1] When we way that a technology *scales*, we mean that it is capable of performing well when loaded with increasing quantities of data or responsibilities.

[2] This diagram is called a tree, because the components branch out like the branches on a tree. Paradoxically the *root* of the tree is at the top, and the *leaves* are at the bottom.

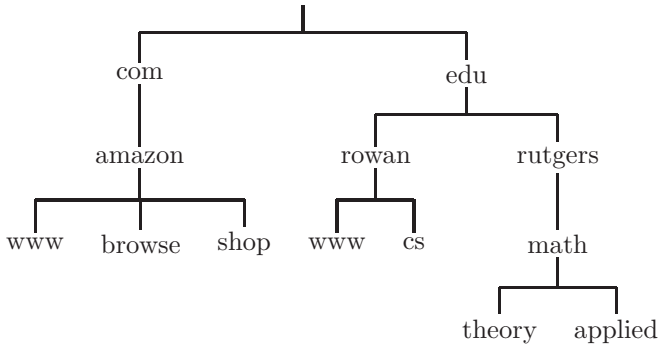


Figure 6.1: Tree diagram of several domain names, showing a hierarchical structure.

- www.amazon.com
- browse.amazon.com
- shop.amazon.com
- www.rowan.edu
- cs.rowan.edu
- www.rutgers.edu
- math.rutgers.edu
- theory.math.rutgers.edu
- applied.math.rutgers.edu

Figure 6.1 shows these domain names in the form of a tree.

At the top (i.e. root) of the tree we have `com` (company) and `edu` (education). Other common top-level labels are `gov` (government), `mil` (military), `org` (organization) and `net` (network). Nations other than the United States use a two-letter abbreviation for their country name at the top level, as shown in Figure 6.2.[3]

Domain names are assigned by the Internet Corporation for Assigned Names and Numbers (ICANN). Anyone can apply to ICANN for a new domain name. If it is a pre-existing name, it would have to be purchased from the owner for a negotiable price. In the early days of the internet it was not unusual for savvy individuals to apply for domain names such as `pepsi.com`, `ford.com`, or `CIGNA.com`. These domain names were correctly thought to be very valuable to the named organizations, and the original owners were able to sell these domain names at a huge profit.

[3]The small pacific island nation of Tuvalu sold its top level name `.tv` to the corporation `dotTV`; Tuvalu now owns 20% of that company.

Abbreviation	Country Name
at	Austria
au	Australia
ca	Canada
eu	European Union
mx	Mexico
nz	New Zealand
tv	Tuvalu
uk	United Kingdom
us	United States

Figure 6.2: Some common country abbreviations in internet domain names.

6.2.1.5 Internet standards

When digital devices need to communicate with each other it is important that they agree on how information is to be represented, how a connection is to be established, how to check for transmission errors, how to signal for termination, etc. These agreements are called *standards*. Some examples of internet standards are:

- HyperText Transfer Protocol (`http`): HyperText is derived from Apple's hypercard application (circa 1987), in which there may be links in a text document to other text documents. The transfer protocol is a standard way of sending these text documents (e.g. web sites) from one device on the internet to another.[4]

- Simple Mail Transfer Protocol (`SMTP`): This protocol, first defined in 1982, was used to standardize the transmission of email on the internet. It has evolved over the years to enable the inclusion of non-text in email messages, such as images and video clips.

- Transmission Control Protocol / Internet Protocol (`TCP/IP`): This is the protocol referred to above in the section on IP addresses. It standardizes the protocol used by any digital device to access the internet. `TCP` includeds checking for transmission errors.

- User Datagram Protocol (`UDP`): This is similar to `TCP`, but does not include error checking, and is more efficient.

6.2.2 Exercises

1. Spell out the full name of TCP/IP, and describe its purpose.

2. What is an IP address?

[4] `https` is a cryptographically secure version of `http` which ensures confidentiality and authenticity of the text which is transmitted.

3. What is a domain name? Give an example.

4. Spell out the full name of HTTP, and describe its purpose.

6.3 Some Characteristics of the Internet

6.3.1 Hierarchical design and redundancy

6.3.1.1 Hierarchical design

A *hierarchy* is an ordered list of items. Some examples of hierarchies on the internet are:

- The components of a domain name, such as `applied.math.rutgers.edu` may be thought of as an ordered list; in Figure 6.1 we view that list in a vertical format, showing that `edu` is the top-level component and `applied` is the bottom-level component.

- An IP address might also be considered a hierarchy. The original IP addresses were 32 bits and were usually shown as 4 integers in the range 0..255 separated by dots. For example:
  `150.250.20.112`
  Each number in the range 0..255 may be thought of as an 8-bit binary value, since $2^8 = 256$ the binary values range from 00000000 = 0 to 11111111 = 255.

  These IP addresses form a hierarchy with the first number considered the top level. At the author's institution, all of our IP addresses originally began with `150.250`.

- The architecture of the internet itself is a hierarchical structure.$^{\otimes}$ The Open Systems Interconnection (OSI) model is used to describe the design of the internet as a stack of 7 layers. Each layer is an abstraction of the more detailed layer beneath it. Figure 6.3 contains a brief description of each of the OSI layers. Further description of these layers is currently beyond the scope of this text.

6.3.1.2 Redundancy: Packet switching

In order to be resilient to transmission errors, outages, tampering, etc. there is much redundancy built into the internet. In general *redundancy* inolves duplicating a solution or implementation, so that if one solution fails, the system could still continue to function with the duplicated solution. One good example of redundancy is known as *packet switching*.

When a device on the internet needs to send a large message[5] to another device on the internet it is not sent as one long string of bits. This would

[5] Here the word *message* refers to any digital information, whether it be text, image, sound clip, video clip, or random bits.

	Layer	Protocol Data Unit	Function
7	Application	Data	High-level APIs
6	Presentation		Translation of data (e.g. encryption)
5	Session		Back-and-forth between pair of nodes
4	Transport	Segment, Datagram	Transmission of segments
3	Network	Packet	Routing, Addressing
2	Data Link	Frame	Transmission of a Frame from Node to Node
1	Physical	Symbol	Transmission of bit streams

Figure 6.3: The seven hierarchical layers of OSI, defining protocols for traffic on the internet

be susceptible to transmission errors and/or failures. Instead the message is broken into *packets* of variable size.[6] A typical message will then consist of several packets, in a specific sequence. Each packet consists of:

- A header, which contains origin and destination information, as well as a sequence number for the packet within the message.
- The data itself, i.e. bits of the actual message.

When the message is sent to another device on the internet (i.e. another IP address), the packets are sent individually to one or more internet nodes. Each node then attempts to forward packets as they are received until they eventually reach their intended destination. When the receiving device receives all the packets of a message, it reassembles those packets into a complete message using the sequence numbers in the packet headers. Figure 6.4 is a diagram showing how packets are sent from an origin to a destination.

In that figure the original message consists of 5 packets, labeled `P0`, `P1`, `P2`, `P3`, `P4`. The originating device may put out these packets to one or more internet nodes. It may duplicate the packets being sent out (e.g. the packet `P0` and `P4` are sent to the nodes `c1` and `c4`). This duplication is an example of redundancy. Some of the packets that have been sent out may take a long time to reach the destination, or fail to reach the destination at all, which is why redundancy is important.

At the destination (Dest) the device receives the packets `P4`, `P1`, `P3`, `P2`, `P1`, `P0`, `P4` in that order. Note that some of the packets are received twice due to the redundancy. The destination device then reassembles the packets in the correct sequence, discarding duplicate packets, to obtain the original message.[7]

Because of the redundancy inherit in the packet switching algorithm, the internet is said to be *fault tolerant*. A fault tolerant system can continue to function even after encountering unexpected errors.

[6] In IPv4 the packet size ranges from 20 bytes to 64K bytes.

[7] If, after a period of time, the destination device has not received all the packets of a message, it can signal to the originating device to resend the message.

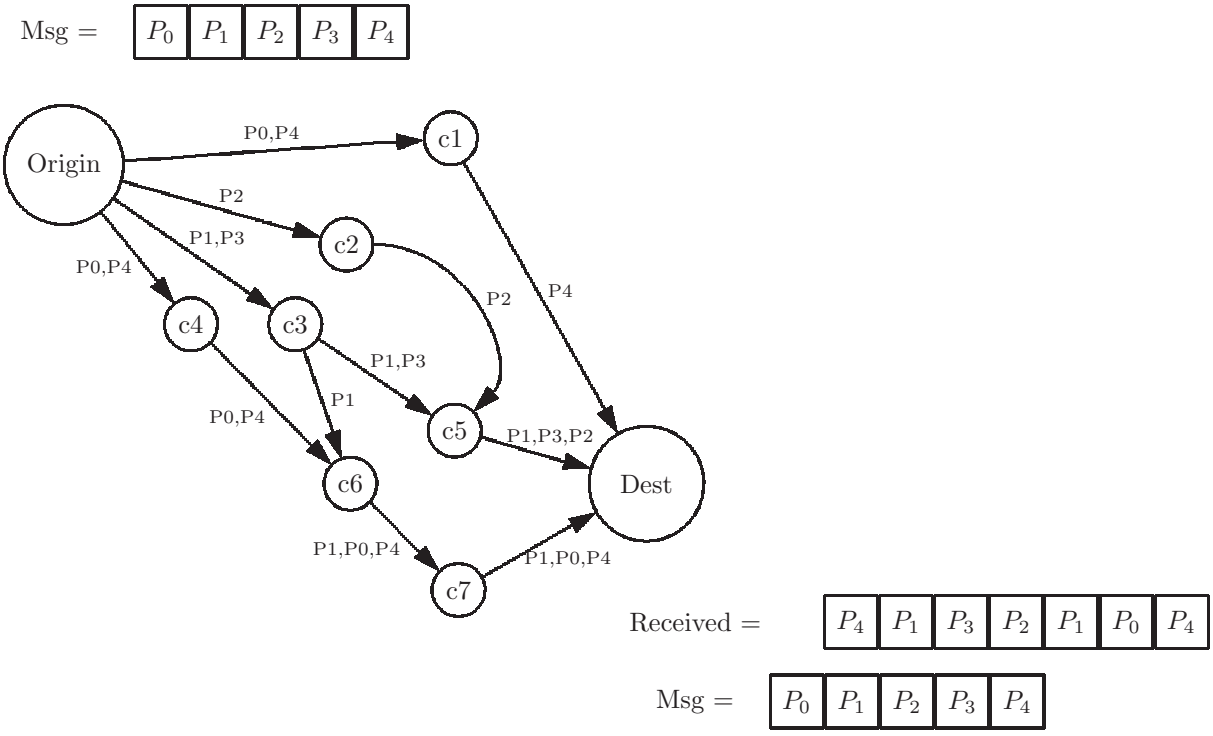


Figure 6.4: Diagram of packet switching on a network. The original message is divided into 5 packets which are put onto the network and reassembled at the destination (Dest).

6.3.2 Standards, growth, and scalability

When we say that a system *scales* well, or that it is *scalable*, we mean that as its load increases it continues to function well; i.e. it does not slow down excessively. The internet was designed with this in mind.

The hierarchical design and redundancy described in the preceding section are principally responsible for the remarkable scalability of the internet. In 1980 none of us dreamed that the internet would become ubiquitous and valuable all over the world, yet it continues to function with very little *latency* (the time between a request and a response to that request).

Aspects of the internet which have enabled this growth are:

- Redundancy of message routing: multiple routes to a destination
- Hierarchy of the domain name system
- Open standards (as opposed to proprietary standards) for interfaces and protocols, such as TCP/IP and http
- Packet switching
- TCP/IP software manages the effective routing of packets
- World-wide web standards: `http` and `https`
- World-wide web starndards for secure communication: SSL (Secure Socket Layer) and TLS (Transport Layer Security)

All these have an effect on the performance of the internet; if the internet did not scale well, people would resort to other communication strategies, and the internet as we know it would wither. One result of this amazing capability for scalabilty is what is known as the *Internet of Things* (IOT). Small devices such as digital cameras, weather sensors, advertising kiosks, etc. each have their own IP addresses and can communicate directly with other devices on the internet.

One way of measuring the performance of the internet is by *bandwidth*, which is the number of bits that can be sent from an origin to a destination per unit of time. Depending on the mode of connection to the internet, the bandwidth can range from 1 Mbit/second [8] to 100 Gbit/second. [9]

6.3.3 Exercises

1. How many hierachical levels are there in each of the following?
   (a) The domain name `gazette.ben.penn.edu`
   (b) The IP address `150.250.33.203`
   (c) The OSI model of the internet

[8] 1 Mbit = 1 Megabit = $2^{20} bits \approx 1$ *million* bits
[9] 1 Gbit = 1 Gigabit = $2^{30} bits \approx 1$ *billion* bits

2. Describe how redundancy is built in to the packet switching communication protocol on the internet.
3. Describe what is meant by *scalability*.
4. What is the popular name usually used to describe the huge number of devices currently connected to the internet?

6.4 Cybersecurity

Cybersecurity, otherwise known as computer security, or information technology security, involves the protection of computer networks, computers, and other digital devices which have access to the network. Software which is designed to cause harm is known as *malware*.[10] Cybersecurity systems are designed to protect from possible:

- damage to hardware, software, or data
- disruption of service
- unathorized access to private or confidential information, including transmitted information (email, etc.)
- unathorized intrusion to a component not intended for public access
- criminal or unethical uses of the internet

6.4.1 Addressing cybersecurity concerns

How are the concerns listed above being addressed to ensure security?

6.4.1.1 End-to-end encryption

Normally communication systems on the internet, such as email, have no built-in encryption. The content is available to whomever may intercept the transmission.[11] The solution is to encrypt messages, in such a way that only the intended recipient can decrypt the message to see the original plain text. There are many well designed software packages[12] many of which are freely available on the internet.

However, the typical user may not have the expertise required to download and use the encryption software properly; or it may be inconvenient. Moreover, the intended recipient(s) would need to download the same software, and would

[10] Malicious software

[11] An email is broken into packets, which are sent via various redundant routes to the receiver; however, some, or all, of the packets can be intercepted at an intermediate point and reassembled. Confidentiality is not preserved. Unlike letters sent via the US Postal Service, there is no federal law protecting against intrusion of email. An unencrypted email is like a postcard, the content is available to anyone who happens to see it.

[12] See Gnu Privacy Guard (GnuPg or GPG) which is free and available on the internet

face the same hurdles: expertise required, and inconvenience. To address these problems, *end-to-end encryption* packages have been developed. These packages provide encryption which, to various extents, is easy to install, seamless, and effective. After installation, the encryption is applied, by default, and the software is working 'behind the scenes' in such a way that the user does not even notice its effects.

One such package, Signal[13], is open source and free on the internet. It was created in 2014, but saw a substantial increase in downloads in 2019-2020. Signal not only provides end-to-end encryption (for email and telephone transmissions) services, but also provides authentication[14] and integrity[15] services.

6.4.1.2 Trust model

Much of the security built into the internet is based on the notion of *trust*. When we make a purchase online, we trust that our credit card information will remain confidential (to prevent financial fraud or theft). When we file our taxes online, we trust that our social security number remains confidential (to prevent *identity theft*). When a corporation's employees communicate *trade secrets* to other employees they trust that the trade secrets will remain confidentialial.

This is not implying that we should trust anyone or anything we encounter on the internet; to the contrary. However, all of the examples given above use *cryptographic algorithms*[16] to ensure that the internet can be trusted.

Digital certificates are like virtual id cards. They can be used to ensure authenticity of a person or entity on the internet. Once a group of people or entities who have shared their digital certificates have communicated safely they have formed a *web of trust*; others can then join the web of trust. A large web of trust ensures safe and secure communication. Also, there are *trusted authorities* or *certificate authorities*, large corporations with many clients, which can issue digital certificates, for a fee, to private individuals or corporations. An example of a digital certificate, issued to an author of this book, by Verisign is shown in Figure 6.5.

6.4.1.3 Caveat on domain names

There is no inherent trust built into the domain name system. Anyone can obtain a domain name (though they would have to identify themselves to ICANN). Users should be careful to check for correct domain names. For example, you may receive an email with a link, or visit a web page with a link, that appears as "Amazon". However, when you *mouse over* [17] the link, your browser reveals

[13] For an interesting interview with the founder of Signal, Moxie Marlinspike, see the *New Yorker* magazine, Oct 26,2020

[14] Authentication is the process of assuring the true identity of a person.

[15] Integrity is the process of ensuring that there has been no tampering, or altering, of communications.

[16] Cryptography is used not only for confidentiality, but also for authenticity and integrity in internet communication.

[17] Move your mouse pointer over a link, without clicking on the link

X.509 Certificate	
Issued To	
Common Name (CN)	Seth D. Bergmann
Organization	Rowan University
Organizational Unit (OU)	Computer Science
Serial Number	46:3c:2a:01:33:30:ba:91:22:4c:25:71
Issued By	
Common Name (CN)	Verisign, inc.
Organization	Verisign, inc.
Organizational Unit (OU)	
Serial Number	33:34:1a:9b:21:30:0a:31:42:4d:24:22
Period of Validity	
Begins on	19 June 2020
Expires on	18 June 2024
Fingerprints	
SHA-256	23:a3:c3:55:ac:8f:32:23:98:9a:02:23:34:5b:ff:9a:bc:b0:97:23:32:e3: 22:bb:32:90:02:32:49:a8:3c:d0:13:82:e1:cd:35:88:bc:21:34:ab:c9:23
SHA-1	23:a3:c3:55:ac:8f:32:23:98:9a:02:23:34:5b:ff:9a:bc:b0:97:23:32:e3:f3

Figure 6.5: An example showing some of the information in a digital certificate issued to Prof. Seth Bergmann; issued by Verisign

the actual URL[18] of the linked site. If it is not "Amazon.com", it is probably a hoax, possibly dangerous, and should be avoided.

6.4.1.4 Hardware, software, and human components of cybersecurity

Security can be built into hardware. A few examples:

- It is now possible to encrypt the entire contents of a computer's disk storage. If an intruder attempts to access data from the disk, it will have no useful meaning since, without the appropriate key information, it cannot be decrypted. Amazingly, this encryption/decryption process is fast and transparent to the owner of the data.
- Desktop telephones and other communication devices can be equipped with encryption technology for secure and confidential communication. These devices are used primarily in the military and in high-level government communications, but also in some business environments.
- Smartphones and other wireless phones must have security built in for confidentiality, even for the most casual users, as the transmissions are broadcast 'over the air'.

[18] Universal Resource Locator: This includes the domain name of the selected web site.

Security can be built into software. As we have seen, the *http* protocol which is used to navigate the world wide web has a secure version, *https*. It uses standard protocols to encrypt and authenticate transactions on the web (SSL and TLS).

In addition to hardware and software there are *human* components of security. People need to be aware that they are vulnerable to attack and/or cyber crime if they:

- Choose passwords which can be guessed easily
- Make their passwords available to casual acquaintances
- Leave a computer or phone unattended, especially if it is logged into a secure cite
- Respond to suspicious emails
- Click on strange links in emails or web sites
- Trust people who have been met on social media, with no knowledge of their true identity

6.4.1.5 Cyberwarfare

When a nation-state or adversary of a nation-state uses digital communications to infiltrate and/or attack the information systems of another nation-state, it is known as *cyberwarfare*.

- *Infiltration* may involve accessing classified data, snooping around, gathering as much information as possible without being detected. The nation-state being attacked would presumably be unaware that its classified information has been compromised.
- In *attack* situations the attacker will modify and/or delete critical classified information. In particular, information on the attacker could be altered or removed. If the nation-state being attacked did not notice the break-in, the modified information could be copied to back-up locations, and it would be difficult to restore the damaged files.
- Weapon systems are often controlled through digital communication channels. If not secure, those weapons could be disabled or otherwise taken over by an adversary.
- Defensive systems can be used to thwart an enemy's attack. Missile systems such as the Patriot Missile are designed to detect attacking missiles or aircraft and destroy them before they can deliver payloads. This kind of weapon is known as Anti-Ballistic Missile (ABM).
- Drone aircraft are increasingly being used for reconnaisance (when equipped only with cameras) and warfare (when equipped with weapons).

6.4.1.6 Cybercrime

Cybercrime is similar to Cyberwarfare, but the attacker and victim are typically not nation-states. Motives for cybercrime are generally financial (theft), but could also include revenge, illegal transactions, money laundering to avoid income taxes, and vice-related motives.

Some examples of cybercrimes are:

- Breaking into the information system of a financial institution such as a bank to obtain or modify financial records.
- Breaking into a company's data storage, encrypting the data with a secret key, and demanding a ransom (usually in bitcoin) for the key (see *ransomware* below).
- Intercepting credit card information on the internet.
- Accessing a retailer's database to obtain account numbers and credit card numbers.
- Accessing a government database to obtain social securiy numbers for the purpose of identity theft.
- Vice-related crimes such as child pornography, human trafficking, and illegal drug transactions.

6.4.1.7 Attacks on the Internet

Since its inception the internet itself has been the target of attacks, some malicious, and some less serious.

- In 1988 Robert Morris, a graduate student in Computer Science at Cornell University, thought he discovered a flaw in the internet's email protocols. He thought that if he sent an email to all his contacts, with an attached program to send to all of their contacts, recursively, that his original email would propagate through the internet. He tried it, and it worked. Soon many computers on the internet were doing nothing but forwarding Morris' original email, effectively shutting down those computers for other purposes. This also increased traffic on the internet to such an extent that everyone experienced serious delays. This kind of self-propagating software is known as a *Worm*, and this particular Worm was known as the Morris Worm. Robert Morris became the first person tried and convicted under the 1986 Federal Computer Fraud and Abuse Act.
- A more general kind of attack is known as a *Denial of Service* (DoS) attack. A DoS attack will typically flood a targeted computer or host with many superfluous requests for service, thus overloading the host with requests and limiting service for legitimate requests. Perpetrators often target sites

or services hosted on web servers, such as banks, or credit card payment gateways.

A *Distributed* DoS attack is an attack which is launched from several different IP addresses on the internet, all targeting the same service or host.

6.4.1.8 Attacks on Individuals

Malware which targets individuals rather than networks include viruses, phishing, cryptolocking, ransomware, and trojan horses.

- A software *virus* is capable of replicating itself and modifying other programs on the same computer so as to include the virus, which then infect still more programs. Anti-virus software is designed to protect your computer from a virus attack, but new viruses are always forthcoming. To be safe you need to update your anti-virus software frequently.
- A *phishing* attack is an attempt at identity theft, generally by an email, instant message, or website posing as a trusted entity. It will attempt to obtain sensitive information such as passwords, credit card numbers, and social security numbers. To thwart phishing attacks always check the actual email address of the sender (not the name) or check the actual URL of a website (not the link text). If they are unfamiliar, they are not to be trusted.
- *Cryptolocking* software is malware which gains access to a storage medium such as a fixed disk, and encrypts the data using a secret key. The owner of the data must then pay a ransom to obtain the key and decrypt the data.
- Any malware which poses the threat of permanent damage and demands a payment is known as *ransomware*. Often to avoid being identified, the payment is made with a cryptocurrency, such as Bitcoin.
- A trojan horse is named after the legendary Greek attack on the city of Troy, in which Greek soldiers hid in a large wooden horse and gained access to the fortified city. A computer *trojan horse* is any malware which is hidden in some vehicle, such as an email (with malware in an attachment), or a web site (with malware in a link).

6.4.1.9 Firewalls

One commong strategy used for cyber security is known as a *firewall*. This is network security softare on a local network which monitors and controls incoming and outgoing network traffic. It is capable of filtering out incoming malware, as well as vulnerable outgoing traffice. Often

- Incoming malware can be recognized and rejected, or suspicious emails can be automatically stored in a 'junk' email folder.

- Outgoing traffic, emails containing sensitive information such as social security numbers, financial account numbers, and credit card numbers, can be automatically encrypted. Instructions are then sent to the legitimate recipient on how to decrypt the information.

6.4.1.10 Personal cybersecurity

Cybersecurity can be improved by using smart personal habits:

- Use strong passwords on financial and academic sites.
- Change passwords often.
- Use different passwords on different sites.
- Do not leave sensitive information in public areas; do not send sensitive information in the 'clear' (i.e. unencrypted).
- Do not store passwords and other sensitive information on public computers.
- Do not open nor respoond to suspicious emails.
- Do not click on suspicious links.

6.4.1.11 Cryptographic cybersecurity

Other strategies for improving cybersecurity involve *cryptographic algorithms* which use mathematical computations to ensure security. These include:

- Confidentiality: Communication can be encrypted by the sender of a message, in such a way that it can be decrypted only by the intended recipient. There are two fundamental types of encryption algorithms:
  - Private key cryptography: The sender and the receiver share a common secret key. Only those who hold the key can decrypt messages which were encrypted with that key.[19]
  - Public key cryptography: Each user has a pair of keys - a public key and a private key. The public key is known to everyone. The private key is not shared with anyone, not even the intended recipient(s). The two keys are mathematically related. When encrypting a message to an intended recipient, the sender encrypts the message using the recipient's public key. The recipient decrypts the message with his/her private key. Noone else can decrypt the message.[20]

[19] How can the sender and recipient share a key safely? This is known as the *key distribution problem*.

[20] Public key cryptogaphy also provides a solution to the key distribution problem: A *session key* is encrypted using a public key cryptosystem, and sent to a recipient, who then decrypts it.

- Integrity: There is malware which is capable of intercepting messages being transmitted on the internet. The malware can make changes to the message and forward the altered message to the recipient, as though it was coming directly from the sender. A mathematical *hash function* can be used to thwart this attack, thus ensuring integrity. The input to a hash function is a message consisting of an unlimited number of bits. The output of the hash function is a bit sequence of fixed length, typically about 200 bits, which has no apparent relation to the message.

    1. The sender uses the hash function to produce a *hash value*:
    `hash(msg1) = h1`
    2. The sender sends the hash value, h1, to the recipient.
    3. The recipient receives the hash value, h1.
    4. The sender sends the message, msg1, to the recipient.
    5. The recipient receives a message, msg2.
    6. The recipient uses the message, msg2, as input to the hash function
    `hash(msg2) = h2`
    7. The result, h2, should be equal to h1. If not, the recipient does not trust the message; malware has tampered with either the message or the transmitted hash value.

- Authenticity: When communicating on the internet, how can we be sure that the person/entity with whom we are communicating is really who they claim to be? For example, when sending your credit card number to Amazon, how do you know that it is really going to Amazon, and not some other site controlled by malware? Public key cryptographic algorithms provide a solution to this problem, using *digital signatures*.

    1. A message can be signed by a sender using a public key algorithm, shown here as `Decr(msg)`:
    `sig = Decr(msg)`
    2. The message (either encrypted or sent in the clear) and the signature are both sent to the recipient.
    3. The recipient *verifies* the signature by encrypting the signature, shown here as `Encr(sig)`.
    `verify = Encr(sig) = Encr(Decr(msg)) = msg` If necessary, the recipient also decrypts the message.
    4. If verify is not equal to msg, verification fails and the recipient assumes that the message is not authentic, i.e. the sender is not who he/she/it claims to be.[21]

[21] Digital signatures are, in a sense, the *reverse* of usual practice. Instead of encrypting plain text, the sender *decrypts* the plain text. Instead of decrypting cipher text, the recipient *encrypts* cipher text.

6.4.1.12 Open standards for cryptography

In the early days of cryptography, people sought security by keeping algorithms secret. They believed that they were more secure if their enemies did not know their encryption algorithms. Eventually, however, whether by means of statistical methods, or other methods, our enemies will 'break' our codes.[22]

Today we generally believe in the efficacy of publicly known algorithms, also known as *standard* algorithms. Instead of hiding our algorithms, security is obtained by

- Using sufficiently large keys so that a brute force attack will take too much time, computationally
- Using public algorithms to ensure integrity
- Using public algorithms to ensure authenticity
- Generating a new session key for every communication session, and distributing those keys using a public key cryptosystem.

When algorithms are public, teams of researchers can collaborate to improve their strength.

6.4.1.13 Certificate authorities

Returning to the problem of authenticity: How can we prove that we are really who we claim to be? How can we prove our identity, cryptographically? In the description of public key cryptograpy, above, what is to prevent some malware from claiming to be Amazon, and providing a public key? A *digital certificate* serves the same purpose as a passport or driver's license. It can be used to authenticate identity. A digital certificate for a person or entity typically consists of:

- Identification information, such as full name
- Affiliated organization, such as the company for whom the person is working, or the educational institution which the person is attending
- Geographic location
- Expiration date
- A public key

The certificate itself does not ensure authenticity. However, there are organizations which are in the business of ensuring the authenticity or their clients. These organizations are known as *certificate authorities*. A certificate authority can issue a digital certificate for an individual person. That certificate is signed

[22] It has been said that every code can be broken.

by the certificate authority, which has its own public/private key pair. To verify a certificate, one merely verifies using the certificate authority's public key. Once the certificate has been verified, it is known that the public key it contains can be trusted. Some examples of certificate authorities are (with their market share, as of March 2020):[23]

- Comodo (41%)
- Symantec (30%)
- GoDaddy (13%)

6.4.2 Exercises

1. How can one check for a suspicious domain name in a web page link?
2. How can one check for an email with suspicious attachments?
3. What are some hardware devices that have built-in cybersecurity features?
4. What steps can an individual person take to improve their own cybersecurity?
5. Distinguish between *infiltration* and *attack* in cyberwarfare.
6. True or False: Robert Morris, a Cornell graduate student, was never tried and convicted of Computer Fraud and Abuse, for bringing down the internet with a worm that he had created.
7. What is a *phishing* attack?
8. What is the name given to software on a local network which serves to filter out incoming malware, and protect outgoing information?
9. Give some examples of *weak* passwords.
10. What kind of cryptographic algorithm is used for each of the following?
    (a) Confidentiality
    (b) Integrity
    (c) Authenticity
11. True or False: We would be more secure if we did not share our cryptographic algorithms with our enemies.
12. What information is usually included in a digital certificate?

[23] Large certificate authorities have a clear advantage over smaller certificate authorities in this business; as a result of of the large number of certificates issued, their level of trust is higher.

Chapter 7

Fault Tolerance

As described in chapter 6, the internet continues to work 24/7 despite occasional local outages or failures. This is known as *fault tolerance*. More generally, fault tolerance is the capability of a digital system, whether it be a single computer, a collection of connected devices, or a network, of recovering from the failure of some component. The form of the recovery can vary, and thus the extent to which a fault tolerant system is tolerant can vary:

- The system may avoid a complete termination, and provide information about the location, cause, and nature of the failure.
- The system may continue to function, with limited capabilities, or with a loss of the state and/or data from the component which failed.
- The system may make a complete recovery and continue to function as though there had been no failure at all.

7.1 Fault tolerance in a single device

In many applications or environments a system failure is unacceptable. Some examples are:

- A medical device used to control life-sustaining equipment in an intensive care unit (ICU) or coronary care unit (CCU).
- Navigational and/or control devices in aircraft and space missions. NASA's early Gemini and Apollo missions were equipped with two identical computers. At that time hardware faults were not very uncommon, and if one computer failed, the other would take over. This is known as *redundancy* and is still used today.
- Failure of a device in autonomous land vehicles can result in serious injury, or worse.

- Certain military operations need to be fault tolerant, such as drone attacks and missile detection systems for defense.
- Electric power grid control is done with computers which must be fault tolerant to provide service 24/7.
- Electronic voting systems, if ever implememnted, need to be fault tolerant to the extent that votes which have been cast are not lost if a component fails.

7.2 Fault tolerance in a network

In order for a network to *scale*, it must be fault tolerant. This means that as the size of the network increases, the probability of a fault somewhere in the network also increases; if a fault in any one node were to cause the network to fail, the network would not be fault tolerant.

The packet switching protocol of the internet is fault tolerant in the sense that there are multiple routes for a packet to reach its destination, as shown in Figure 6.4. In that diagram, if the node labeled `c1` were to fail, the packets `P0` and `P4` could still reach the destination by going through nodes `c4`, `c6`, and `c7`. Thus the reliability of the internet is improved by *redundancy*.

7.3 Redundancy

As described above fault tolerance is usually achieved by including some form of redundancy into the system. This entails the addition of extra, non-essential components, similar, or identical, to other components which can perform the same function if needed. In the case of a local system, redundancy can be achieved by including components which serve no other purpose than the replication of a critical component. Some examples of redundant systems are:

- The example of multiple computers on a mission to outer space, described above, is one example.
- An autonomous vehicle may have overlapping sensors which can view the surrounding terrain for pedestrians, other vehicles, or obstacles.
- In a network such as the internet, redundancy takes the form of multiple paths from an origin to a destination, as shown in Figure 6.4.
- In the 1980's a group of scientists at the University of California, Berkely, developed a fault tolerant data storage system known as RAID (Redundant Array of Inexpensive Disks).[1] These storage devices are capable of making automatic backup copies of all data in real time. Every time an

[1] Other systems which predated RAID, such as the 'Mirroring' system of Tandem NonStop Systems were similar to RAID.

item of data is written to a primary disk, the same data item is automatically written to a secondary disk. Thus the secondary disk contains an identical copy of all data on the primary disk, and serves no other purpose but to ensure that all the data survives if the primary disk should fail.

- All important transaction processing systems, such as airline reservations and financial transactions of banks, as well as medical devices incorporate some level of redunancy in order to tolerate a failure.

It is clear that redundancy increases the cost of a system, but in systems where a failure is not acceptable, the cost is justified. Also, as the cost of hardware continues to plummet, redundancy will be used more widely, and future systems will be more fault tolerant.

7.4 Fault tolerance in software

Redundancy is used to implement fault tolerance primarily in hardware. If two programs are identical, and the first program has a logic error (i.e. a bug), the identical copy of that program will contain the same bug. Redundancy does not appear to have any benefit for software.

However, there are features of C++, Java, Python, and other programming languages, which allow the programmer to build in a form of fault tolerance in software. A Java Exception is like a class, but it is designed to provide the programmer with an easy way to intercept the flow of control when a run-time error has occurred. The programmer can then handle, or recover from, the error and allow processing to continue. The Java *try* statement is used to specify a block of code that could conceivably fail, and the *catch* statement is used to handle the exception and continue processing, rather than coming to a crashing halt.

7.5 Exercises

1. List some fault tolerant systems other than the ones described here.
2. Does redundancy apply primarily to hardware, to software, or to both hardware and software, equally?
3. What were some applications which made use of Tandem NonStop Computers.

Chapter 8

Parallel and Distributed Computing

In this chapter we explore tactics that have been employed to speed up computations. In today's world we are either dealing with large amounts of data, or we are dealing with computational processes which require an exhorbitant amount of time to complete. In either case we are always looking for ways to make our computations more efficient, and terminate in a reasonable amount of time. This chapter explores two common approaches to this problem:

- Parallel computing - The process of executing more than one task, or more than part of a single task, simultaneously. This is typically done on a single computer or an array of local computers.
- Distributed computing - The assignment of subtasks to different, autonomous, users for mutual cooperation in the completion of a task. This is typically done by several computers on a network.

8.1 Parallel computing

Parallel computing was introduced by the Burroughs Corporation in the early 1960's. This term can relate to computations done at different levels:

- Instruction level parallelism attempts to execute different phases of instructions at the same time. A processor can do all of the following simultaniously:
  - Execute an instruction in the CPU
  - Fetch, from memory, the operands for the next instruction
  - Fetch, from memory, the instruction after that one

Figure 8.1: The supercomputer known as IBM Blue Gen/P "Intrepid", at Argonne National Laboratory

- Process level parallelism attempts to execute different parts of a program simultaneously.
- System level parallelism attempts to execute different programs, or tasks, simultaneously.[1]
- An array of local processors can be assigned different aspects of a problem, to arrive at a quick solution. An example of a local array of processors is shown in Figure 8.1.
- At a higher level, several independent computers on a network can be assigne different (or the same) aspects of a problem. This is normally called 'distributed computing', and will be discussed in the next section.

8.1.1 Run-time savings with parallelism

Parallel computing tasks can be broken up into 2 portions - a parallel portion and a sequential portion. The efficiency of a parallel computing task is limited by the sequential portion. Hence there is a limit to the number of parallel processing units which can be used to increase efficiency.

Parallel computing assumes that 2 tasks are not dependent on each other. If each task or step is dependent on the previous task or step then each step will

[1] This is probably the origin of the term *multitasking*.

P1	Time	Needs
T1	5	-
T2	2	T1
T3	7	T2
T4	9	-
T5	8	T2
T6	5	T3

Figure 8.2: Diagram of six tasks executed sequentially by a single processor, P1; total time required is 36. Some tasks need the results of prior tasks.

have to wait for the previous step to complete before starting itself. This would instead imply sequential computing. Thus, the efficiency of parallel computing is limited by the sequential portion of the task.

The speedup rate of a parallel solution is measured in the time it takes to complete the task sequentially divided by the time it takes to complete the task when done in parallel:

$$speedup\ rate\ =\ \frac{time\ to\ complete\ the\ task\ sequentially}{time\ to\ complete\ the\ task\ in\ parallel}$$

For example if the time it takes to complete a task in sequential mode is 70 seconds and the time it takes to complete the task in parallel is 14 seconds then the speed up rate is:

```
speedup rate = 70/14
             = 5
```

Note the speedup rate is a ratio.

At any of the levels mentioned above, the use of parallelism will generally result in smaller execution times. However, there are limitations on the use of parallelism. For example, at the task level, if a task, T1, requires data produced by another task, T2, then T1 and T2 cannot be executed simultaneously. T2 must complete before task T1 can begin.

Also, suppose the two tasks have side effects, such as producing output. If they are executed simultaneously, there is no guarantee that they will produce the same output each time they are invoked; they are not independent tasks.

P1	Time	Needs	P2	Time	Needs	P3	Time	Needs
T1	5	-	T4	9	-			
T2	2	T1						
T3	7	T2				T5	8	T2
T6	5	T3						

Figure 8.3: Diagram of the same six tasks executed in parallel by three processors, P1, P2, P3; total time required is 19. Some tasks need the results of prior tasks.

As an example, we examine six tasks executed sequentially on one processor[2] in Figure 8.2. Each of the tasks has a running time (in some fixed unit of time). Each task also may need results of computations from another task (shown in the `Needs` column of the diagram.[3] Tasks T1 and T4 do not need data produced by any of the other tasks. Because there is only one processor, only one task can execute at a time; i.e. the execution is strictly sequential, with no parallel execution. The total time required for all 6 tasks to run is 36 time units.

Now consider the same example, with 3 processors, each of which is capable of executing a task simultaneously with the other two processors. Figure 8.3 shows a diagram in which the same six tasks are executed. We see that tasks T1 and T4 can execute at the same time because they do not need to wait for another task to terminate. After task T2 terminates, task T5 can start executing on processor P3. After task T2 terminates, tasks T3 and T6 can execute. The six tasks can be completed in only 5+2+7+5 = 19 time units, a savings of almost 50% over the sequential model.

These diagrams are intended to be general in nature. The parallelism could be at the instruction level, the task level, or the system level.

8.1.2 Parallelism in personal computers

Today's personal computers have several *core* processors, each of which is capable of executing a sequence of machine language instructions. One of the most difficult aspects of designing compilers[4] and operating systems[5] involves making use of more than one core to achieve the run-time advantages of parallelism.

Many programming languages have features which allow the programmer to

[2] *Processor* is a general term for the hardware or software responsible for execution of a task.

[3] We assume that the needed data is not available until the task terminates.

[4] A *compiler* is the software which translates a program from a high level language to machine language.

[5] An *operating system* is the software which manages the computer's resources. `Windows`, `Linux`, and `MacOS` are examples of operating systems.

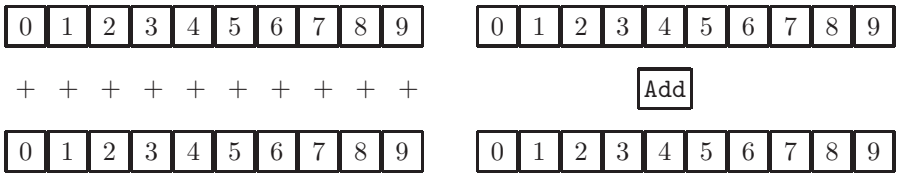


Figure 8.4: Left diagram: single data stream; a loop is needed to add two vectors. Right diagram: multiple data streams; two vectors are added with a single `Add` operation.

specify explicitly which sections of a program can run in parallel. For example, Java Threads allow the programmer to specify subprograms, or blocks of code, which can be executed independently, allowing the operating system to schedule them for simultaneous execution with other Threads.

8.1.3 Instruction stream and data stream parallelism

There are two kinds of parallelism which can be made available by a programming language compiler and/or operating system:

- *Instruction stream* parallelism corresponds to what was described in Figure 8.3. Separate sequences of instructions are executed simultaneously on separate processors.

- *Data stream* parallelism involves a *vector*[6] processing capability. In most programming languages, if we wish to add the values in two arrays, we need to code a loop, as described in chapter5. However, some languages, such as Fortran 90[7] introduced vector operations, in which arrays could be treated as primitive types. Thus, to add two arrays, no loop is needed, but a single operation such as `v1 + v2` would specify the addition of corresponding positions in the two arrays, v1 and v2. Fortan 90 compilers for supercomputers, such as those made by Cray would then generate code to specify that the the addition be done in parallel.

These two types of parallelism brought into use the terminology, *stream* to describe parallelism (or lack thereof) of either of the two types described above. If there is no parallelism, it has a *single* stream, and if there is parallelism, it has *multiple* streams.

To contrast multiple data streams with single data streams, we show a diagram in Figure 8.4 in which a single data stream architecture is shown on the left (a loop is used to add the elements of a vector), and a multiple data stream architecture is shown on the right (a single `Add` operation adds the two vectors).

[6] Vector is another term for array or list.

[7] Fortran was one of the first programming languages and is still used today, primarily for scientific or engineering applications.

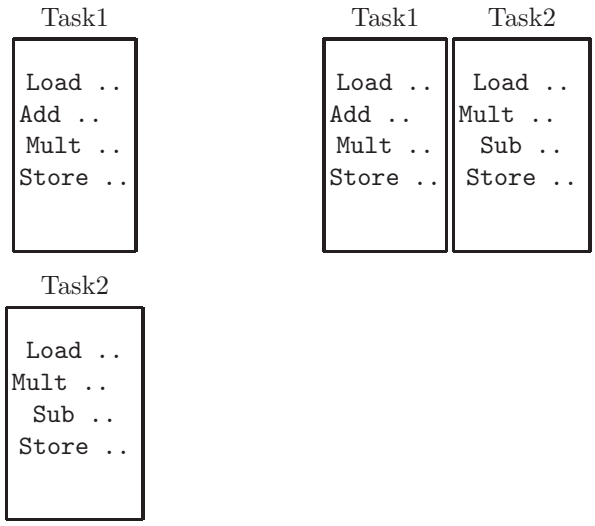


Figure 8.5: Left diagram: a single instruction stream; tasks are executed sequentially by a single processor. Right diagram: multiple instruction streams; two tasks are executed at the same time on different processors.

To contrast multiple instruction streams with single instruction streams, we show a diagram in Figure 8.5 in which a single instruction stream architecture is shown on the left (two tasks are executed sequentially by a single processor), and a multiple data stream architecture is shown on the right (two tasks are executed at the same time by two processors).

Thus we have four ways of specifying parallelism:

- A system which executes only one sequence of instructions at a time, and does not have a vector processor is classified as Single Instruction stream, Single Data stream (SISD). A diagram of the SISD architecture is shown in Figure 8.6.
- A system which executes only one sequence of instructions at a time, but does have a vector processor is classified as Single Instruction stream, Multiple Data stream (SIMD). A diagram of the SIMD architecture is shown in Figure 8.7.
- A system which executes several sequences of instructions simultaneously, but does not have a vector processor is classified as Multiple Instruction stream, Single Data stream (MISD). A diagram of the MISD architecture is shown in Figure 8.8.
- A system which executes several sequences of instructions simultaneously, and also has a vector processor is classified as Multiple Instruction stream,

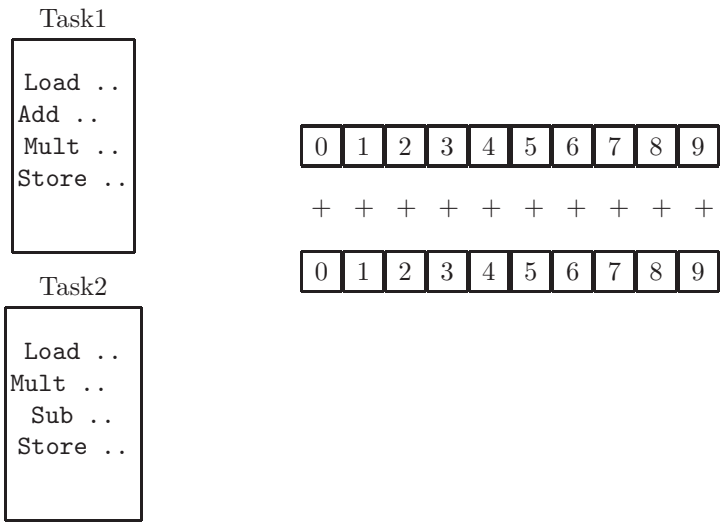


Figure 8.6: Diagram of an SISD architecture; single instruction stream, single data stream

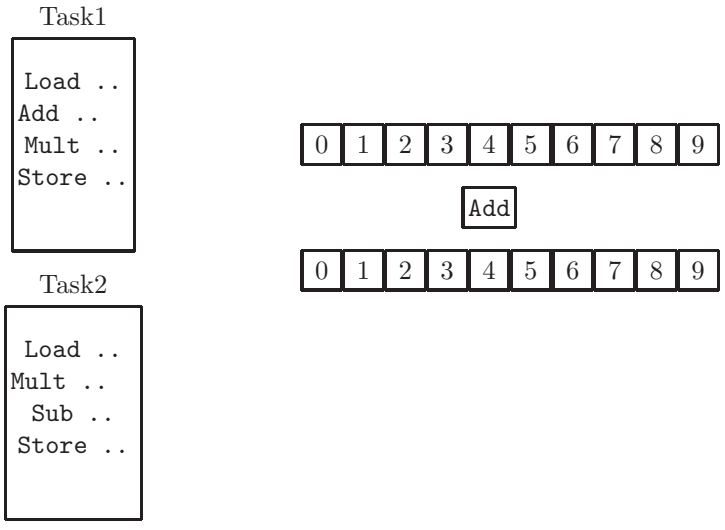


Figure 8.7: Diagram of an SIMD architecture; single instruction stream, multiple data streams

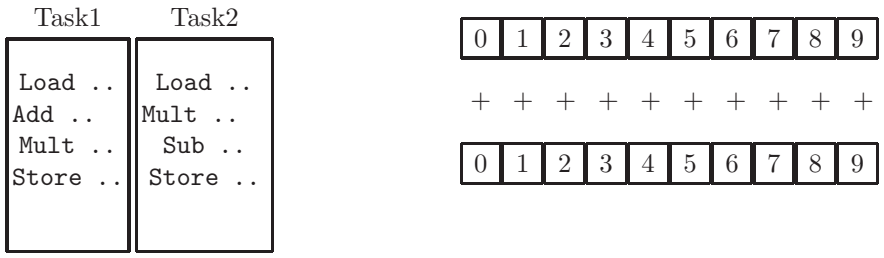


Figure 8.8: Diagram of an MISD architecture; multiple instruction streams, single data stream

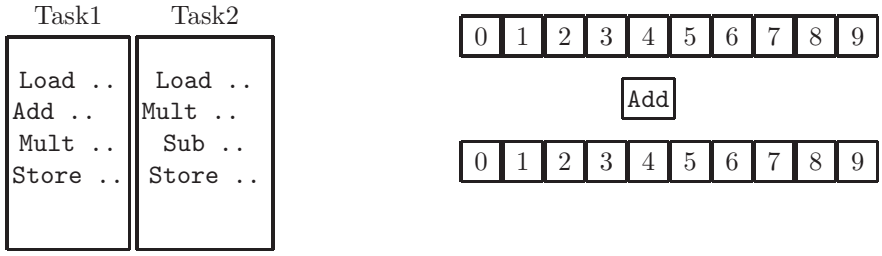


Figure 8.9: Diagram of an MIMD architecture; multiple instruction streams, multiple data stream

Multiple Data stream (MIMD). A diagram of the MIMD architecture is shown in Figure 8.9.

This classification scheme, known as Flynn's Taxonomy, defined in 1966 by Michael J. Flynn, is still used today.

8.1.4 Exercises

1. Which of the following tasks can be executed in parallel (i.e. simultaneously on different core processors)?

   (a) Task T1: Uses data provided by the user on the keyboard, to establish the current local ambient temperature, air pressure, wind velocity, and humidity.

   (b) Task T2: Uses data obtained from the internet to store the temperature, air pressure, wind velocity, and humidity at surrounding locations.

   (c) Task T3: Uses the data obtained from tasks T1 and T2 to predict the local ambient temperature, air pressure, wind velocity, and humidity at some time in the future.

2. Refer to Figure 8.2. How much time would be required to execute the six tasks shown, in parallel, if only two processors were available?

3. Use Flynn's taxonomy to identify each of the following as either SISD, SIMD, MISD, or MIMD.

   (a) A personal computer with one core, which is not a vector processor.

   (b) A personal computer with eight cores, none of which are vector processors.

   (c) A personal scientific workstation with one core that is a vector processor.

   (d) A supercomputer with many CPUs, each of which has a vector processor.

8.2 Distributed Computing

For a more detailed introduction to distributed computing, see *The Essence of Distributed Systems* by Joel Crichlow, Prentice-Hall.

8.2.1 Client-server terminology

In computing systems a module (hardware of software) which provides a service to other modules is known as a *server*. The modules which make use of that service are known as *clients*. For example, when you use your computer to access Amazon.com for a purchase, your computer (actually the browser on your computer) is the client, and Amazon's computer systems are the server.

8.2.2 Distributed computing and parallel computing

Closely related to parallel computing is the area of *distributed computing* which involves the parallel solution of problems with many clients or users. Distributed computing can be helpful in two primary ways:

- Your computer does not have enough memory to solve a problem. The solution is feasible only if several computers are used, each contributing to the total memory required to solve the problem.

- The solution to the problem takes so long that it can be solved only if several computers attempt the solution at the same time. This is feasible only if the problem lends itself to a separation of the search space for a solution, so that no two computers are working on the same aspect of the problem.

Some communication among the participating computer systems will be necessary; thus distributed computing normally implies the use of a network. However, if several clients are on a network, this does not necessarily mean they are

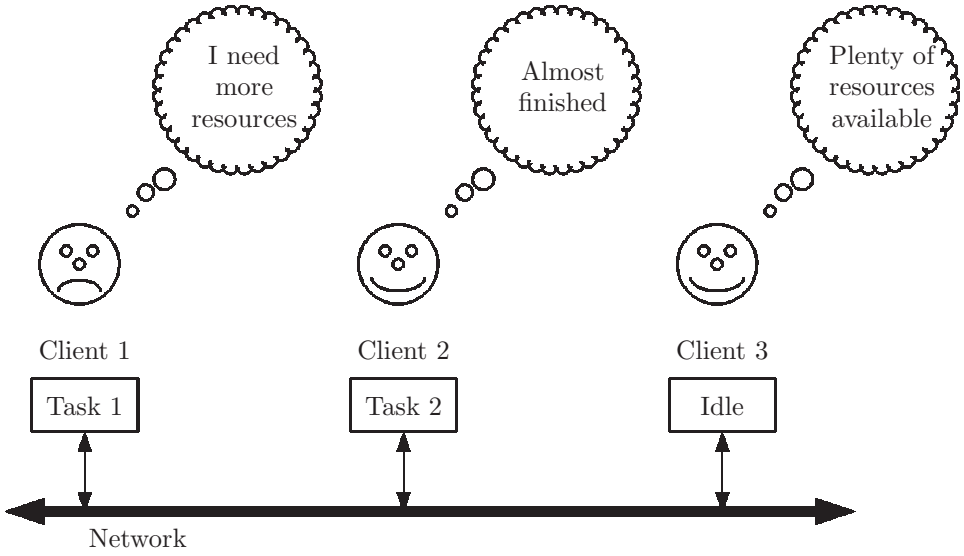


Figure 8.10: A non-distributed system, with three clients on the network

using distributed computing. In Figure 8.10 three clients are on the network. The first client is having trouble solving a problem; it lacks the necessary resources, such as memory, storage, CPU time, etc. The second client is working on a relatively easy problem and is almost finished. The third client is currently not working on any problems, and has lots of resources available. Figure 8.10 is NOT an example of distributed computing.

In Figure 8.11 all three clients are working on Task 1, in addition to whatever else they need to do. All three clients are sharing the burden by contributing resources to the solution of a difficult problem.

8.2.3 Types of distributed systems and examples

8.2.3.1 Distributed computation

When a problem solution involves extensive calculations, or searching for solutions, the problem can be solved faster when several computers work on it in parallel. In such cases the calculations, or search space, must be separated into several components, with one or more components assigned to each of the participating computers. These components should not overlap in their scope, and to be effective, the aggregate of all the components should be equal to the total search space of the problem to yield a solution. Some examples of distributed computation are:

- The search for extra-terristrial intelligence (SETI) involves searching the sky for radio signals which may have originated from intelligent beings on other planets. These signals can be received by large radio telescopes

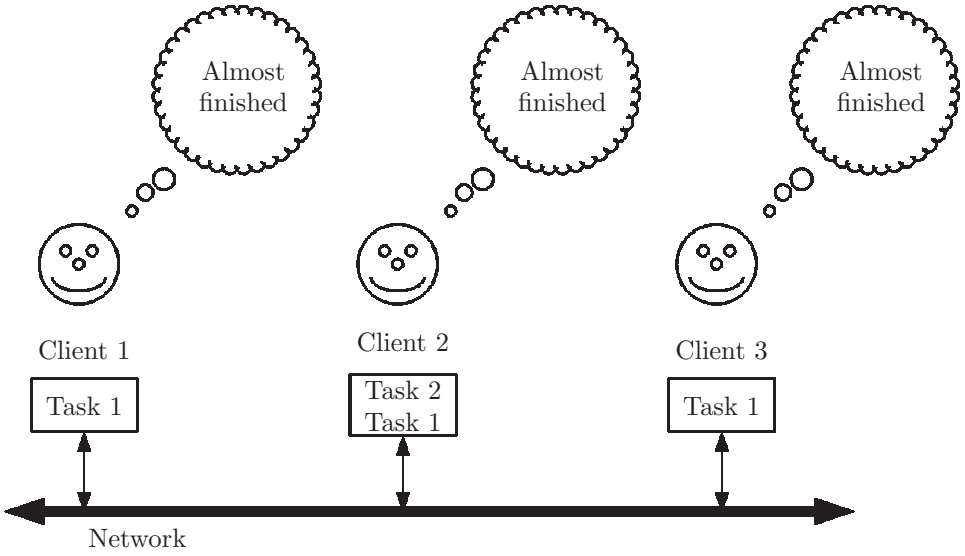


Figure 8.11: A distributed system, with three clients on the network, sharing the burden of Task 1

such as the one operated by NASA at Arecibo, Puerto Rico. The problem is that there is so much information (mostly random noise) that a single computer does not have the speed to search for patterns that could be from intelligent life. In 1999 a distributed computing project known as SETI@home was initiated at the University of California, Berkeley. Software was developed which enabled volunteers from all over the world to offer time on their personal computers to search the data collected at Arecibo. Some 200,000 home computers were ultimately used; each of these computers each searched a small section of the sky for 'intelligent' signals. They did this while otherwise idle, and also in parallel with other tasks that the owners were running. The project was temporarily put on hold in 2020 when the Arecibo radio telescope failed due to a structural collapse. SETI@home was one of the first large scale distributed computing systems to use the internet.

- Bitcoin is the first successful, and most popular, crypto-currency. It is a digital virtual currency (no physical coins, nor paper, are used) which relies on cryptographic algorithms for security and verification of transactions. Bitcoin is a peer-to-peer distributed system based on the internet; the software is free and open source. An important component of the Bitcoin sofware is known as *mining* software, which involves rewarding a user with a new Bitcoin for solving a computational problem. The difficulty of the problem is such that a miner succeeds in solving it about once every 10

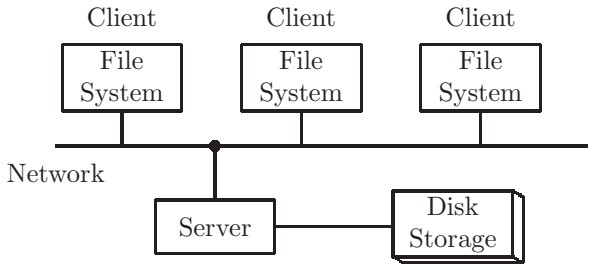


Figure 8.12: Diagram of a distributed file system

minutes. Since a Bitcoin is so valuable,[8] people interested in mining form *mining pools* to solve the problem in a cooperative way, and sharing the reward among those who contributed to the problem solution, if successful. This works because the search space can be easily partitioned into non-overlapping components.

8.2.3.2 Distributed file systems

Data stored in non-volatile devices such as disks or flash memory are critical to most computing applications. In cases where there is so much data that one computing system cannot store all of it, a distributed file system is useful. Also when an application needs to run at several different locations it may be more cost effective to store a subset of the data set at each of the local sites, and make all the data available at all sites through a network.

An example of a distributed file system, developed at Sun Microsystems (now Oracle) is Network File System (NFS), shown in Figure 8.12. Users (i.e. clients) of NFS can enter and retrieve an entire file, or a portion of a file at their local workstations. When the desired data resides on another client's workstation, the NFS software seamlessly provides the data through the network.

8.2.3.3 Distributed databases

A *database* is a collection of files with structure, or organization, imposed to facilitate entry, retrieval, updates, and relevant statistics of a data set. A *database management system* (DBMS) is software which includes a user interface and functionality to work with a database.

When working with large databases, or when many clients will be accessing the database, it will be advantageous to distribute the work among several clients. This is a *distributed database*. Each client would typically store some portion of the complete database. If a client needs to update or access its own portion of the database, the network is not involved. However, when a client attempts to access a portion stored at a different client, the distributed database

[8] According to Coinbase.com one Bitcoin is worth over $US 34,000, at the time of that this chapter was written. The value of a Bitcoin fluctuates greatly due to speculation.

management system will seamlessly provide this access to the client. The users at each client system should not need to see a distinction between local access and network access to the data.

Some distributed database systems maintain a complete version of the database at a central host site. When a client makes an update, it is stored on a client's system, and also on the central site.

Examples of distributed databases are:

- An airline reservation system. Changes are made to the database
  - by customers, on the airline's web site
  - by airline agents at multiple airport check-in or gate stations
  - by agents at a phone call center
  - by the airline's central flight scheduling office

  Each of these clients is sharing the burden of maintaining an accurate and consistent representation of reservations on the airline's flights.
- The account records of a bank or savings and loan association
- The inventory, order, accounting for a retailer with many outlet stores and inventory sites
- The inventory, order, accounting for a super retailer, such as Amazon. Amazon also uses search techniques to locate items not directly available in its own inventory.

8.2.3.4 Distributed real-time systems

When a bank updates interest holdings on all savings accounts, this is generally done overnight when transactions are not being processed. Whether the process takes 5 minutes or 5 hours is not a concern. This kind of process, sometimes called *batch processing* is NOT done in 'real' time. However, when a computational task produces a result, it is sometimes critical that the result be produced within certain time constraints. When events or components in the physical world need the results of the process within a very limited time constraint, we say the system providing the results is a *real-time* system. Examples of real-time systems are:

- A robot on an automobile assembly line needs to form an image of the component being assembled, process that image, and assemble the component before the next component to be assembled arrives on a conveyor belt.
- An autonomous automobile has sensors which have images of the surrounding roadway. Those images need to be processed to determine whether there are pedestriand, obstacles, or other vehicles nearby. This image processing must be done within time constraints, depending on the speed of the automobile.

- Many game-playing systems must perform real-time computations to simulate a human opponent.
- Flight simulators need to respond to the user's input in real-time, to simulate real conditions.

8.2.3.5 Distributed multimedia systems

In chapter 2 we discussed the digital representation of sound, images, and video.

- A sound consists of numbers representing changes in air pressure (i.e. pressure waves). A one minute, uncompressed sound clip requires about 10M bytes of storage.
- An image consists of pixels, many numbers each of which represents the color of a tiny section of the image. The Samsung Galaxy Note 5 smart phone displays 1440x2560 pixel values. Thus a single uncompressed image on that phone would consist of 3,686,400 numbers, requiring over 12M bytes of memory.
- A typical video clip consists of about 30 frames, or images, per second.

Using this information we estimate that a 10 minute uncompressed video clip requires 120M bytes for the sound, and 12M * 30 * 60 * 10 = 216G bytes for the images! Clearly compression of this data is necessary for anyone wishing to store video clips. Streaming services such as Netflix make extensive use of data compression.

Even with compression, multimedia content such as movies, video clips, and podcasts, requires a lot of storage. Multimedia servers, such as the ones operated by Netflix, YouTube, Hulu, and Peacock not only have a huge storage requirement, but must also respond to many simultaneous requests for video content. These servers are often called *streaming* servers because they send the information in separate sections. The client then decompresses a section while the previous section is being viewed. The entire file is never downloaded to the client.

Because of the huge storage requirements and concurrent service requests, multimedia systems are a prime candidate for distributed processing. A service such as Netflix would have multiple servers, each with its own storage (disk) media. When a server receives a request through the server network, it will respond directly if it is storing the requested item. If not, it will forward the request automatically to the appropriate server. A diagram of a multimedia server system is shown in Figure 8.13. In that diagram the distributed system consists of three servers, each of which has its own storage unit with several disks. These servers are networked separately from the internet.

8.2.3.6 Distributed operating systems

An operating system is software installed on every computer, without which the hardware is not capable of anything. The operating system is responsible for

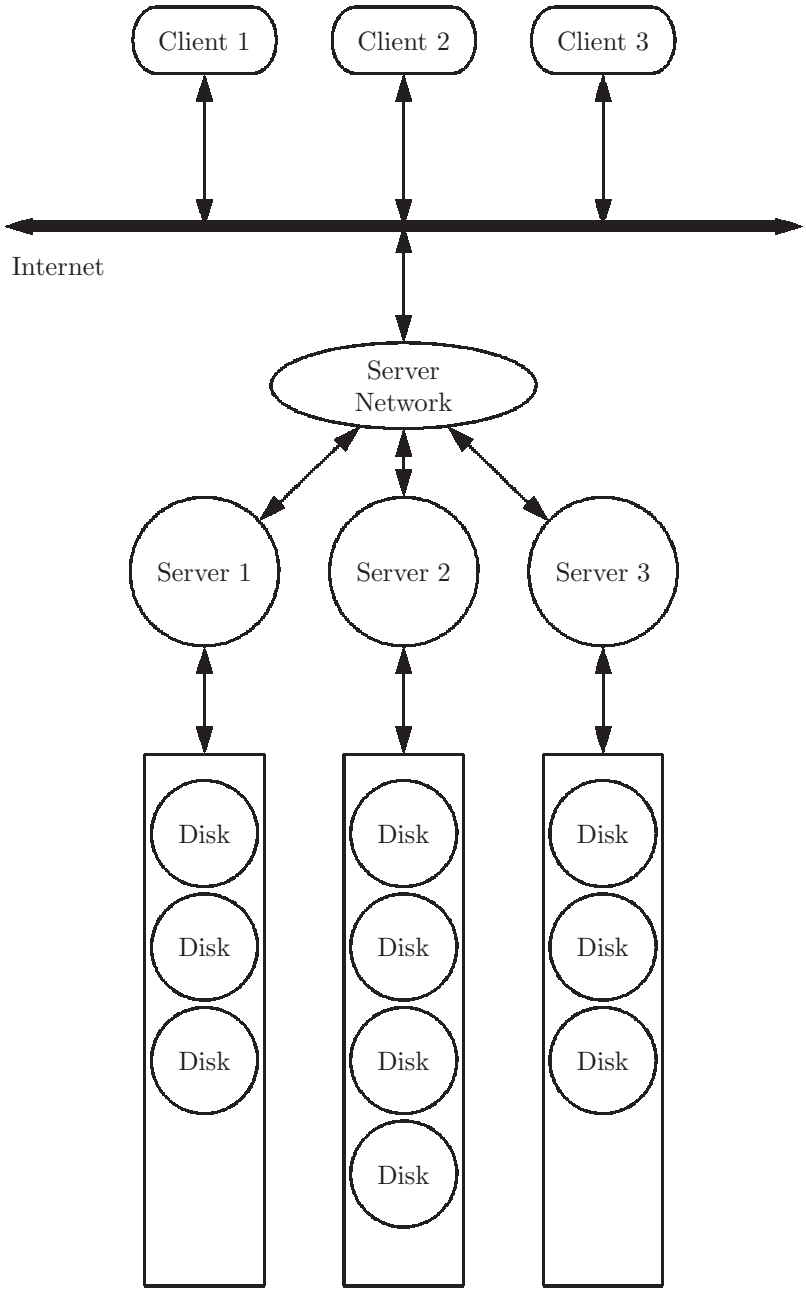


Figure 8.13: A distributed multimedia system with three servers and its own network, storing items such as movies, video clips, and podcasts stored in multiple storage units

managing the computer's resources:

- It loads programs to be executed from the disk to main memory.
- It schedules processes executing concurrently for execution by CPU cores.
- It manages memory usage which involves:
  - allocation and deallocation of memory to/from executing processes
  - making efficient use of memory, or expanding addressable memory using high-speed cache memory and virtual memory on secondary storage
- It provides access to files and directories (i.e. folders) on secondary storage (e.g. disk)
- It allows for input to and output from peripheral devices, such as keyboards, monitors, USB ports, etc.
- It provides an interface for interaction with a network.
- It provides an interface for interaction with a user (often known as a *shell*).

Some examples of operating systems are:

- Windows
- MacOS
- iOS (for Apple smart phones)
- Android (for smart phones other than Apple)
- Unix (or Linux)

Each operating system has its own user interface (also known as a *shell*). Also software written to run on a computer with a given operating system will not run under any other operating system.

A *distributed* operating system provides the user with the capabilites of several different operating systems over a network. A user with a PC running Windows would be able to run programs designed to run under MacOS if some other user on the distributed operating system was using MacOS on an Apple Macintosh computer. Moreover, execution of the program would be seamless, or transparent, to the user; the user would not be aware that the program is actually executing on another computer. A diagram of a distributed operating system is shown in Figure 8.14.

In Figure 8.14 there are three clients:

- The client at the top is running MacOS, with a bash shell for user commands.

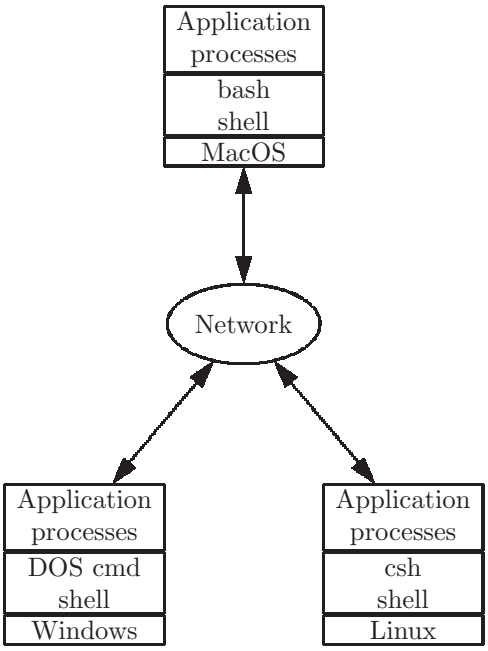


Figure 8.14: A distributed operating system with three clients, running different operating systems

- The client at the lower left is running Windows, with a DOS command shell for user commands.
- The client at the lower right is running Linux, with a shell known as csh for user commands.

Any one of the three clients can execute a program for any of these three operating systems. Moreover, this is transparent, meaning that the user is unaware that the program is executing on a different computer.

8.2.4 Exercises

1. I am using my desktop iMac at home to access my university's Unix system, named elvis, which allows many simultaneous users.
   (a) Is my iMac a client or a server?
   (b) Is elvis a client or a server?
2. What are the six kinds of distributed computing systems described in this section?
3. Describe how the Bitcoin mining problem can be attempted by several cooperating clients.

4. In what sense is a distributed computing system fault-tolerant?

5. In a multi-media system, when is a file decompressed?

   (a) When it is recorded

   (b) When it is stored on the server's storage system

   (c) When it is requested by a client

   (d) Prior to downloading or streaming it to a client

   (e) When the client has received at least the first section and wishes to convert it to sound and images.

Chapter 9

Global Impact

Computation has changed the way people think, work, live, and play. Computation in today's world is pervasive, and is becoming increasingly so - computing devices are everywhere; often we are not even aware of their presence. In this chapter we expose ways in which computation has fostered or improved communication, innovation, and personal cognition. We will also examine computational artifacts which can be harmful to individuals, or to our society. Finally, we examine the ways in which economics, society, and culture have driven developments in the field computation.

9.1 Communication, Interaction, and Cognition

Today it is easy for people to communicate with others all over the world; as recently as 1980, this was not the case. Long distance telephone calls were expensive, written communication took days in transit, and live video communication among individuals was unheard of. Today these are things which are taken for granted. As a result the way we think and solve problems has been altered.

9.1.1 Computing innovations

9.1.1.1 Written communication

Today we are using digital devices to send written communication more than ever, with the following artifacts:

- Email - was the application which was first to have an impact on written communications, in the late 1970's.[1] Today, with digitally encoded attachments, much more than text can be sent with email.

[1] The author sent his first email in 1973, albeit local email on a mainframe computer.

- SMS - Short Message Service, commonly known as "texting", was originally implemented to allow for cheaper communication via mobile phones. There are far fewer bits in 10-20 ASCII characters, than in compressed audio. This enabled the carriers to accommodate more usage per unit of time, and pass the savings on to customers in a very competitive business. As users became accustomed to this form of communication, usage eventually exceeded audio communication.[2]

- Chat - is a term usually used to describe a world-wide web application which allows two or more users to enter text in real time. Each user can see the other user(s)' text as those texts are entered. Chat is often used by corporations who need to respond to clients' requests for help or information; chat enables the corporation to respond to more requests per day than would a voice phone system.

9.1.1.2 Video communication

Bandwidth is a measure of the quantity of information which can be transmitted over a communications channel per unit of time. Bandwidth is usually expressed as bits per second. Because our bandwidth has increased in recent years,[3] large bit-sequences, such as those encompassing voice, images, and video, can now be sent in real time over the internet. Several applications which allow for *virtual* conferencing have been created:

- Skype - Audio and video only

- Facetime - Audio and video only

- Zoom - Audio, video, application sharing, screen sharing

- Cisco Webex - Audio, video, application sharing, breakout rooms for private sessions, and many other features

- Microsoft Team - Audio, video, application sharing, breakout rooms for private sessions, and many other features

- Learning Management Systems, such as Blackboard Collaborate - Audio, video, application sharing, breakout rooms for private sessions

9.1.1.3 Social media

So-called *social media* are internet-based services which allow users to interact with others in a way that was previously less convenient. Social media allow people to connect with other people with whom they may have common interests. Social media have evolved to include audio, images, and video, in addition

[2] SMS was the most widely used data application at the end of 2010, with an estimated 3.5 billion active users, or about 80% of all mobile subscribers (Source: Wikipedia.org, March 2020)

[3] Fiber-optic technology, and undersea cables are largely responsible for this.

to text. Because of their popularity, and because they deal with dense information, such as video clips, social media organizations have a need for extensive data storage capabilities, high bandwidth communication channels, and technical staff. These are generally funded by advertising. Some common social media sites, in order of popularity are:[4]

- Facebook - Founded by Mark Zuckerburg, a student at Harvard University in 2003, Facebook was based on a pamphlet distributed to all undergraduate students, showing pictures and some limited biographical information of all their classmates. Zuckerburg created a digital version of this pamphlet, which he eventually made available to other Ivy League universities, then other universities, and high school students. Finally in 2006 it was made publicly available. Registered users can post text, photographs and multimedia; with various levels of accessibility. Despite controversial issues (described below), Facebook was the most downloaded mobile app of the decade 2010-2019, globally.

- YouTube - A video-sharing platform created by former PayPal employees in 2005. Users are able to view, rate, share, and comment on video clips. YouTube was purchased by Google in 2006 for US$1.65 billion.

- WhatsApp - A text and voice messaging service. For voice it uses *Voice Over IP* (VoIP) technology to digitize audio, and transmit via the internet to another WhatsApp client thus bypassing the traditional telephone network system. WhatsApp is available on computers as well as smart phones. WhatsApp was acquired by Facebook in 2014 for US$19 billion.

- Facebook Messenger - A text messaging application. Facebook messaging is a descendant of Facebook Chat, which in turn is a descendant of America Online (AOL) Instant Message (IM). Messenger is now available on computers as well as tablets and smart phones. Users can also attach digital files containing photos, audio clips, etc.

- WeChat - A Chinese version of Messenger.

- Instagram - A photo and video sharing service. Users can upload photo and video files, to be shared publicly, or to restricted access lists. Users can rate other users posts (like) and follow other users. As of 2020 the most followed person is footballer Cristiano Ronaldo (210 million followers), and the most followed woman is singer Ariana Grande (179 million followers). Instagram was acquired by Facebook in 2012.

- Twitter - An internet-based service which allows users to post messages, known as "tweets". Others can view the tweets, and follow particular users, as with Instagram. Twitter was known as the 'SMS of the Internet' because it was designed to be used with short text messages for inexpensive and efficient transmissions. Twitter is banned in Iran, China, and North

[4]Source: Wikipedia.org, March 2020

Korea; it has been intermittenly blocked in numerous countries including Eqypt, Iraq, Turkey, and Venezuela.

- LinkedIn - A business and employment-oriented service. LinkedIn is primarily used for professional 'networking'[5] As of 2015, most of the company's revenue came from selling access to information about its members. In 2016 LinkedIn was purchased by Microsoft.

9.1.1.4 Cloud Computing

Since the early 1980's people have been using applications on their personal computers to perform tasks such as word processing, spreadsheets, graphic design, etc. These applications were programs, with a separate copy on each individual's computer[6]

As the internet developed, and bandwidth improved, it became clear that these applications could reside at one central computer. This enabled users to produce word processing documents and spreadsheets without installing the software on their own computers. The documents themselves could also be stored on the central computer. Because we visualize the software and data at some distant location, perhaps upward, this became known as *cloud computing*. Google has been offering cloud computing services, allowing for free usage of office applications (Google G suite), free storage of data (Google Docs), and free conferencing software (Google Hangout) in addition to the company's traditional search engine. Amazon is also a major player in cloud computing with its offering of Amazon Web Services (AWS).

Today the phrase 'cloud' refers to anything which is at a remote central location, which gives us a convenient way of remembering the meanings of two new words:

- Upload: Transfer of data or a document from a local computer to the central server, i.e. up to the cloud
- Download: Transfer of data or a document from a central server to a local computer, i.e. down from the cloud

9.1.1.5 Problem solution and dissemination of results

Prior to the advent of the internet research and collaboration among people working in the same area of research took place at a relatively slow pace. An individual would conduct research, submit a paper to a journal, wait for the review process (several months), wait for the paper to be published if it was accepted (several more months) only to discover that someone published similar or contradictory information in another journal.

[5]The author notes the common usage of 'networking' to mean the inter-communication of a group of people for the purpose of discovering or filling employment opportunities or establishing business relationships. It really has nothing to do with computer networks.

[6]The Microsoft corporation built an empire by selling this software, packaged with its Windows operating system.

Today, though the review process still takes time, the publication and dissemination of results is much faster. Digital versions of the journals are obtained by libraries and individuals all over the world.

In addition researches at geographically separated locations can collaborate on research, particularly if the research is computing-related.

For problems which require a lot of computing power, the problem can be developed as a *distributed* application, meaning that several computers can be working on the same problem at the same time, and coordinate their efforts. A few examples of this kind of problem solution are:

- SETI At Home: Search for Extraterrestrial Intelligence involves the processing of radio signals from distant galaxies to determine whether any patterns can be extracted from the usual noise emanating from the skies, as detected by radio telescopes. This requires much computing power. The SETI At Home project enabled anyone with a computer to download the software and take part in this project, without duplicating the efforts made by others.

- Bitcoin mining: Bitcoin is a cryptocurrency based on a peer-to-peer distributed network. There is no central repository for Bitcoin. To 'mine' a Bitcoin, one needs to download the software, and use it to solve a computing-intensive problem. The first miner to solve the problem gets a few newly 'minted' Bitcoins, in addition to other monetary rewards.[7] Because of the intense competition involved in mining, miners often collaborate in 'mining pools' where all miners in the pool work to solve the problem. If successful, all the miners in the pool are rewarded in proportion to the amount of computational work that they provided to the effort.

- DNA sequencing is the process of determining the nucleic acid sequence in a molecule of DNA for a particular species. There are four kinds of nucleotides in the sequence, denoted ad C,T,A,G. The number of nucleotides in one molecule can be in the millions, and the computation time involved in discovering the sequence is enormous. By assigning separate sections of the molecule, or separate genes, to various researchers, the DNA sequencing problem has been solved for many species, including humans.

9.1.1.6 Access to public data

Access to publicly available data has enabled the solution of many important problems. The ability to store large quantities of data, the ability to transmit these data efficiently, and the connectivity of the internet have all had a significant impact on our lives.

[7] As of April 2020, one Bitcoin was worth about US$6,750

- The National Weather Service maintains a database of atmospheric conditions throughout the US.[8] Using this data, the weather service is able to make fairly accurate weather forecasts, specific to local geographic locations. Weather forecasting has become extremely important in our economy, affecting transportation, commerce, agriculture, and military concerns. The data, as well as the forecasts, are publicly available.
- Census data, updated every 10 years, allows us to plan for infrastructure improvements: roads, bridges, housing, schools, etc.
- The coronavirus COVID-19 pandemic of 2020 would have been worse if governments had not made available information on the locations of the confirmed cases. Using this information, the virus was, to a small extent, contained to several areas, while the search for treatments and vaccinations took place.

  After vaccines were developed, the process of vaccine acquisition and delivery was developed using public information, such as web sites and databases.

9.1.1.7 Value of search trends

When people need to find information, they usually go to the internet, and use a *search engine*, such as Google to find what they need. Google has profited immensely by maintaining a database of what people search for. This is valuable information, for marketing purposes, and Google sells this information to interested companies.[9] For example, if you are in the business of selling real estate, it would be very helpful to know whether people are currently searching for single-family homes, duplexes, condominiums, or apartments, and in which geographic locations they have an interest.

9.1.1.8 Global Positioning System

Originated by the military, the Global Positioning System (GPS) consists of several satellites, some of which are in geosynchronous orbit[10] around the earth. Using signals from these satellites, a digital device can determine its own location, including lattitude, longitude, and altitude above sea level. This has had an enormous impact on navigation, not only for individual travelers, but also for air traffic, seafaring ships, and military units.

9.1.1.9 Networks of sensors

Inexpensive digital devices can sense local information, and make that information available on the internet. These sensors are part of the *internet of things*:

[8] By coordinating with other countries, and by using data from satellites, this database is now worldwide.

[9] Google also profits by selling advertising opportunities targeting specific searches.

[10] A geosynchronous orbit is one which has a period of one day; thus the satellite appears to be stationary to an observer on the earth.

- Networks of weather balloons and satellites sense atmospheric conditions to facilitate weather information and forecasting.
- Satellites and toll road sensors provide information on traffic patterns and problems in real time. In the near future, GPS devices in vehicles will enable the vehicles to communicate with each other for similar purposes.
- Vehicles can now sense the presence of hazards and other vehicles, warning the driver and/or averting danger automatically.
- Digital chips are now embedded in bridges to sense strain or movements of the bridge under the load of traffic. This information can be forwarded to a central site, alerting engineers of a forthcoming bridge failure.
- Agricultural products can be packaged with digital devices which track their location, and which coordinate with a central database. If a product is found to be tainted in some way, the source of the problem can be tracked to determine which products need to be removed from the shelves in stores.

9.1.1.10 Built-in smarts

With digital devices, certain intelligence can be incorporated in buildings, roads, and transportation networks. A smart building would know whether its foundation is sinking, whether its roof is defective, whether its elevator systems are operating efficiently, etc. Smart roads can sense traffic patterns and alert drivers of traffic jams. Traffic information is then stored in a centralized database, and is updated in real time.

9.1.1.11 Human assistive technologies

Digital devices can be used to augment human capabilities:

- Hand amputees can now have prosthetics with electrical nerve attachments, giving the patient sensory information similar to what they would receive from an actual hand. The prosthetics can also receive nerve signals from the patient's brain to effect the grasping of objects and other hand movements.
- Leg and foot amputees can now have prosthetics which are more than just stabilizers. They can receive and send nerve signals to facilitate walking.
- Hearing impaired people have been using digital hearing aids for several years. Digital technology allows them to hide noise and amplify important auditory signals. Newer technology for the totally deaf connects directly to auditory nerves, enabling them to learn to hear over time.
- Digital gloves permit those who cannot speak to communicate using hand movements.

- Virtual reality systems consist of a digital image viewer worn over the eyes and headphones on the ears. These systems permit an individual to experience a simulated environment for training or educational purposes.

9.1.1.12 Impact on society

The internet has impacted several aspects of our society and our daily lives:

- *E-commerce* is the name given to the capability of selling and purchasing items using the internet. E-commerce is the opposite of *bricks-and-mortar*, a more traditional business model in which the customer visits a store to select purchases. Most large retailers use a combination of the E-commerce and brick-and-mortar business models.

- Health care has been affected in many ways. Large health-care conglomerates now consist of a network of hospitals, doctors offices, rehabilitation centers, etc. The process of scheduling appointments, obtaining referrals, filling prescriptions, and communication among health care providers is now done on the internet. Health insurance, including Medicare and Medicaid, claims and transactions are now done on the internet.

- Today when people need information on almost any subject, they use an online search engine, such as Google. Where can I get a hoagie? How do I get to my cousin's house? How can I reset my digital watch (I lost the instruction paper years ago)? What can you tell me about Zachary Taylor (I'm writing a term paper)?[11] The answers to all these questions, and more, can be found on the internet.

- The internet has become a source of entertainment for many people. TV shows, movies, etc. are now *streamed*[12] on the internet. Companies such as Netflix have succeeded whereas companies such as Blockbuster have failed.[13] The internet has also been a source of entertainment in the form of game-playing with many players, and sports gambling[14]

- Online learning has gained popularity in recent years.

  – Most universities now offer a large segment of their curriculum as online courses, or hybrid courses.[15] A few universities are entirely on-line, with no classroom buildings.

[11] Many educators discourage the use of internet sources for purposes such as this; there is little guarantee that the information is complete and correct, and that the source is permanent. Most educators agree that sources should always be cited.

[12] Because of the huge size of a digital video file, it is more efficient to transmit the bits as needed, rather than the entire file at once. This is known as *streaming*.

[13] In his 1995 seminal work, *Being Digital*, Nicholas Negroponte predicted that the shipping of bits would eventually replace the shipping of atoms.

[14] Sports gambling, as with any organized gambling, is a long-term loss for the participants; we have to assume that people take part in sports gambling for its entertainment value.

[15] A *hybrid* course is one in which some components are done online and others are done face-to-face.

- In the early 2000's many universities, most notably Stanford and MIT, began offering Massive Open Online Courses (MOOC). These are essentially just video clips of lectures, with no interaction between the instructor and students. However, they are free, and the instructors are often well-known in their respective fields.
- Private companies now offer online courses with substantially more interaction between the instructor and the students. One such company is Coursera, which offers courses that generally involve exercises, quizzes, peer-graded assignments, and sometimes a final exam or project.
- During the Corona virus pandemic of 2020, all education would have had to terminate instruction if it had not been for the internet. Most institutions were able to continue using conferencing software such as Zoom, Cisco Webex, Blackboard Collaborate, Microsoft Team, and Google Hangout.

9.1.1.13 Impact on productivity

The internet has had an impact on productivity. Tasks which previously took days, can now be accomplished in a few seconds, given the appropriate tools. Employees can interact with colleagues even when traveling on business (or on vacation, which could be seen as a disadvantage). Also the internet has engendered distributed systems. A good example is an airline's reservation system. People can reserve seats on flights from any location.

For all the benefits accrued from the internet, there is one major disadvantage. We now rely on the internet for almost everything - commerce, transportation, education, weather, communication. If the internet were to fail, there would be global chaos, for which the recovery could take years. This is why network security is so important.

9.1.2 Scaling of the problem-solving process

Above we defined the notion of a problem solution which *scales*. As the size of the problem increases, the resources required to solve the problem increases at a manageable rate.[16]

The question of scaling is most appropriate in distributed systems, in which many separate computers are attempting to solve the problem in parallel. The best example is the SETI at Home project, in which people all over the world download software to search data obtained from radio telescopes. They are attempting to extract patterns, or meaningful signals, from the random noise emanating from distant galaxies (and from our own galaxy). The developers of SETI at Home were careful to ensure that the solution scaled.

[16] Mathematically, if the need for resources increases at an *exponential* rate as the size of the problem increases, the solution definitely does not scale.

Search engines are another example of a problem-solving process which scales. A search engine is faced with a mind-blowing task: search the entire world-wide web for some artifact. The amount of data presented to the search engine is enormous, and growing every day; yet, the time required for a search does not increase appreciably, for a successful search engine.

9.1.2.1 Problems which do not scale

There are, however, problems for which no one has been able to find a solution which scales. An example is known as the *traveling salesman problem*: Given a group of cities, and roads connecting the cities,[17] find a shortest path from city to city, in which each city is visited exactly once. It is easy to write a computer program that solves this problem, but if the number of cities is modestly large (100 or so), it will take too much time to execute. No one has found a way of solving the traveling salesman problem which scales, yet no one has been able to prove that no such solution exists.

There are many other problems which can be shown to be equivalent to the traveling salesman problem. If a solution which scales could be found for any one of those problems, we would have a solution which scales for all of them. These problems are known as *NP-Complete* problems.

9.1.2.2 Benefits from the contributions of many

Some online services use the contributions of many people to benefit both individuals and society. An example is the web service known as 'Meetup'. Meetup allows people to establish groups with a common interest. Once formed, members of the group, designated as leaders, can schedule and organize meetings of the group. Examples of Meetup groups are: Yoga enthusiasts, dog breeders, hiking and outdoor activity clubs, chess players, World War II history buffs, and many more.

The 'LinkedIn' site is primarily for people seeking employment, or companies seeking to hire employees. This is often called *networking*. The data collected by LinkedIn has some valuable features:

- LinkedIn can determine which fields have a glut of workers, and which are seeing a scarcity of workers.
- LinkedIn can determine which geographic locations have a glut of workers, and which have a scarcity of workers.
- LinkedIn can use trends in employment history to predict future employment statistics

[17] Mathematically, this is called a *graph*.

9.1.2.3 Crowdsourcing

When individuals or organizations obtain goods and services, including ideas, labor, and/or finances, from a large group of internet users, it is called *crowdsourcing*.[18]

The best example of crowdsourcing is `Wikipedia`, which is an online version of an encyclopedia. `Wikipedia` is built by people with some degree of expertise on a particular subject; they provide information on that subject, much like an encyclopedia entry. The contributors receive no financial payment, but since each contribution is relatively brief and many people may contribute to the same entry, Wikipedia has been able to produce a rather large collection of entries, many with well-drawn figures and diagrams. As people discover errors or poorly written sections, the Wikipedia articles are gradually improved over time. There are now Wikipedias in the languages of the world.[19]

9.1.2.4 Mobile computing

Recent years have seen an explosion in the usage of mobile digital communication devices. There are two general classifications of these devices: *smart phones* and *tablets* This new technology resulted from a combination of:

- Inexpensive, dense, and non-volatile semiconductor storage, known as *flash memory*
- Touch screen technology as a user interface
- High resolution digital photographic equipment
- Greatly improved battery technology

When coupled with the internet, phones can be used for much more than speaking with other people. Mobile phones are equipped with:

- GPS sensors for interactive maps and navigation
- Cameras, including video capability
- Sound recording and playback
- Intelligent software, natural language communication

9.1.3 Exercises

1. Today, written communication is primarily digital. In your own words, define each of the following forms of digital communication and briefly

[18] The term *crowdsourcing* is derived from *outsourcing* in which a specific source of the ideas or finances is named.

[19] There are 6.1 million articles in the English Wikipedia, which is the largest of more than 300 Wikipedia encyclopedias. Source: Wikipedia, April 2020

describe a situation in which the digital communication form may be preferred (i.e. when Chat would be the preferred method of digital communication over SMS and Email):

(a) Email
(b) SMS
(c) Chat

2. During the coronavirus COVID-19 pandemic of 2020, almost all learning became remote and relied on video communication:

   (a) Please list the types of video communication you have used for an academic purpose (e.g. Zoom).
   (b) Explain *bandwidth* and explain how did bandwidth has impacted your personal academic experience.

3. Social Media is part of the daily life of many people. However, there are some concerns about the amount of time spent using these platforms.

   (a) If you are a social media user: how many total hours per day, on average, are you on these platforms? Can you identify some advantages and disadvantages of your Social Media use?
   (b) If you are not a social media user: can you identify instances of others pressuring you or questioning you about being "off" social media? Do you think you will one day join a social media platform?

4. What is the difference between an *upload* and a *download*? Please give an example of something you would upload and something you would download.

5. Can you give some specific examples of ways access to public data helped many people?

6. What is your "go to" search engine and why?

7. Computing innovations have had a huge impact on E-Commerce. Please think broadly about the benefits and consequences of E-Commerce growth. Briefly outline and describe the benefits and consequences.

8. Today, we rely heavily on the internet. Let's say tomorrow, you no longer had access to the internet and needed to travel a far distance to a new place. What would you do to successfully make this trip? Remember, you have no access to the internet so you can't use GPS or GoogleMaps.

9. (a) What is meant by a problem solution which *scales*?
   (b) Give an example of a problem, and a solution to that problem which scales.
   (c) Give an example of a problem, and a solution to that problem which does not scale.

10. Explain how crowdsourcing works for Wikipedia.

9.2 Impact on Innovation

9.2.1 Impact on other fields

Advances in computer technology and intelligent software, coupled with the proliferation of the internet have had a huge impact on many other fields and endeavors.

9.2.1.1 Machine learning and data mining

When we devise an algorithm which automatically improves itself as it processes data, we say the algorithm is *learning*, The study and development of such algorithms is currently called *machine learning* (ML), and this is a branch of *artificial intelligence*[20] (AI). AI has been in existence for many years, but the problems which fall in this area has changed. When a problem has an efficient soution, it is no longer considered a problem in AI. For example, it was once thought that if a computer could outplay a human at chess, we would consider the computer intelligent; i.e. the game of chess is a measure of intelligence. However, as computers became faster, and as chess software discovered ways of trimming the search space of possible moves, chess-playing computers eventually outplayed the world's best chess players. At this point chess was no longer thought to be in the domain of AI.

However, chess is a good example of machine learning, when chess software improves automatically as a result of many chess games played against human opponents or other chess machines. The learning software is able to recognize patterns which are associated with winning or losing strategies.

There are examples of machine learning systems in banking, bioinformatics, computer vision, linguistics, robot locomotion, and many other areas.

Data mining is related to machine learning. When presented with a huge quantity of data, programs which search the data, and gradually accumulate knowledge of patterns or information in the data, we say the program is *mining* the data. As this is done, the program is able to impose a structure on the data, disregarding data values which are not relevant. The data mining software will extract previously unknown, but interesting patterns, such as groups of records (cluster analysis), unusual records (anomaly detection), and dependencies (implication of patterns). As opposed to information retrieval, data mining software is provided with little or no information on a search target; rather, it works independently to discover general features of the data.

9.2.1.2 Innovation in science and business

Advances in computing have had enormous impacts on developments in science and business. We have previously mentioned a few of the ways that computing has impacted science, some of which are mentioned here:

[20] A famous computer scientist, Saul Gorn, once remarked, "Aritificial intelligence is the opposite of natural stupidity".

- Computers have enabled us to decode the sequence of nucliotides in a DNA molecule for many species, including humans. This has led to many important applications. We can determine ancestral relationships among people; we can identify people accused of a crime; we can determine whether a person is predisposed to a genetic disease: breast/ovarian cancer, alzheimer's disease, Huntington's chorea, and many more.

- Pharmaceutical research often uses computationally intensive algorithms. Simulated models of large organic molecules can be used to develop new drugs.

- In the past astronomers painstakingly perused photographs of the night sky, searching for new galaxies, nebulae, comets, planets, and asteroids. Now the searching is done with pattern-matching software. Also, the discovery of a non-visible object such as a planet is often accomplished by noting the slight, but unexplained, motion of a larger nearby object, such as a star. Computers can detect these miniscule changes in position far better than human observers.

- Geographers and cartographers can make relief maps from satelite images. Political maps are easily drawn and edited with graphics software; maps can be enhanced with database information, such as population, clmate, altitude, etc.

- Archaeologists and anthropologists have a new tool known as *Lidar* (Light Detection and Ranging)[21] Most often used from aircraft, a Lidar machine uses a combination of laser and radar technology to see 'through' dense foliage to the ground surface below. The reflected laser beams are converted by a fast computer to a database of ground elevation levels. Using this technology, scientists have discovered lost cities of ancient civilizations, which had constructed pyramids, in the jungles of Central and South America.[22]

- Mathematicians spend their lives trying to find proofs of important theorems. Recently, the proof process has been formalized, so that a proof can be 'discovered' automatically, or a computer can be used to assist a mathematician in searching for a proof. *Automated Reasoning* software has been successful in finding difficult proofs. The most notable example of such a proof was a proof of the *four color theorem* in 1976 by Appel and Haken, with an improved version in 2005 by Gonthier. The four color theorem states that the regions of any map can be colored with four colors, with the constraint that adjacent regions must have different colors.

There have been numerous innovations in the busineess world resulting from advances in computation.

[21] Also known as LIDAR, LiDAR, and LADAR. Lidar was originally a portmanteau of *light* and *radar*. It is also known as *3D laser scanning*.

[22] See *The Lost City of the Monkey God* [2017] by Douglas Preston.

- In marketing research we attempt to discover the demographics of a market: What do people want? What are they willing to pay? Where are they located? How are their decisions influenced: by TV? by billboards? by the internet? by word-of-mouth? In the past this kind of information was obtained by either informal surveys or trial-and-error. Surveys took the form of mail, phone, or in-person surveys. Trial-and-error consisted of putting out a product or prototype to discover the market.

  Today this information is available in much larger quantities, and with much more detail. When you switch on your TV or change channels, the cable company maintains a record of what you are watching. This information can be used to sell advertising, or the record of what people are watching can be sold to individual companies or market research companies.

- When you use social media, such as Facebook, Twitter, or Instagram, your likes and preferences are maintained in a central database. This is valuable marketing information

- International corporations have probably gained the most from the internet. A company can have offices all over the world, with instant communications among offices. Teleconferencing facilitates meetings without the expense and down time of travel.

- There has been an expansion of business in the area of *data analytics*.[23] As a result of the huge amount of data available on the internet, the process of extracting useful information from the raw data is now being automated.

- There have been drastic changes in the stock markets resulting from the internet. In the past individual investors would contact an agent authorized to buy or sell shares of a stock. Today transactions can be done instantaneously on the internet. Some have exploited this capability by buying and selling quickly several times in one day, which is known as *high-frequency trading*.[24]

9.2.1.3 Accessing and sharing information

Modern computing has fostered innovation in many areas by providing the ability to access and share information.

Researchers in the sciences, social sciences, and other areas collect data before analyzing the data. This data can be shared with other researchers working on the same problems. It is not unusual for different researchers to draw different conclusions from the same data; this would encourage them to re-examine their research methods and agree on a conclusion which is more likely to be correct.

[23] Data analytics is the practical application of *data science*.

[24] See the book *Flash Boys: A Wall Street Revolt* by Michael Lewis, 2014

9.2.1.4 Open access, open source, and Creative Commons

Copyright laws in the United States are intended to encourage the creation of art and culture by giving the artists and authors exclusive rights to sell copies of their works.[25] Many people, including the authors of this book, have discovered that by foregoing the copyright, a work is often distributed much more widely.

Open source software and open source textbooks, such as this one, are created with this in mind. The authors receive no financial compensation (royalties). The textbook usually goes through no formal review process. However, as a result of widespread usage, and multiple authors, the work is gradually improved over time. The work may even *fork* several times, meaning that it splits into two different development projects with a separate team on each project. As users adopt a particular version of the project, one branch may wither and the other branch will thrive, forming a natural selection process. In this way quality is achieved with an open source work. Examples of open source software are Linux, Mozilla Firefox, and Apache internet server software.

The *Creative Commons* organization, founded in 2001 by Lawrence Lessig et. al., has codified the various ways in which copyrights can be modified. This has resulted in Creative Commons *licenses* which define the procedures and rights to copy for a creative work. Some examples of Creative Commons licenses are:

- Attribution (BY): The work may be copied and distributed only if they give the original author attibution (i.e. they cite the author).
- Share-Alike (SA): Derivative works[26] can be distributed, but only with a license that is no more restrictive than the original work's license.
- Non-commercial (NC): The original work may be copied, distributed, performed, etc., but only for non-commercial purposes. The copies may not be sold for profit.
- No Derivative Works (ND): The original work may be copied and distributed, but not modified.

These licenses are not mutually exclusive. A work may have more than one of these licenses, as long as they are not contradictory. This copyright license for this book is NC + BY.

9.2.1.5 Open and curated scientific databases

To foster collaboration among scientists, they often make their research data available to other scientists. With this kind of collaboration, all scientists can benefit from the work of other scientists. These databases include not only research data, but also references or citations to publications or other web sites. Some examples of *open data* are:

[25] Copyrights expire after 70 years.

[26] A derivative work is one that is derived from the original work. For example, web browser software with features added to the original work is a derivative work.

- Geographic maps
- Genome maps for various animal and plant species
- Connectomes - A 'wiring diagram' of the human brain
- Chemical compounds
- Mathematical formulae and identities
- Medical data - Diagnostic techniques, medication recommendations, success history
- Pharmaceutical data - Drugs, showing proprietary and generic names

Some examples of search engines for research publications are:

- CiteSeer: Database of research for Computer Science, Mathematics, and Statistics
- FSTA: Food Science and Technoloty Abstracts
- GeoRef: Geographical Sciences sources
- Google Scholar: Multidisciplinary
- IEEE Xplore: Computer Sience, Engineering, Electronics
- INSPIRE-HEP: High Energy Physics
- JStor: A multidisciplinary collection of journals, books, and research findings, with search capabilities
- MEDLINE: Medicine, Healthcare
- Science Acclerator: Department of Energy research and development

9.2.1.6 Impact of Moore's Law

In 1965 Gordon Moore, of the Fairchild Semiconductor Corporation, predicted that the number of components in an integrated circuit would double every two years. This statement implies exponential growth in the density of an integrated circuit. Amazingly, this prediction held true through 2012.[27] One consequence of this growth in density is that the speed of a computer's CPU would double every 18 months.

Because of this prediction, from the 1960's through the end of the millenium, software engineers developing new products could often assume that programs which take too long to execute during development and testing would run quickly

[27] One of the main obstacles that prevented further improvements is the heat dissipation problem. As integrated circuits become increasingly dense, they require more electric power, and generate more heat, which must be dissipated to avoid failure.

after the software is released, simply due to the faster CPUs which would be available at that time.

It has been said that if the automotive industry had made engineering advances similar to those made in the computer hardware industry, a Rolls-Royce would get 5000 miles per gallon of fuel, have a top speed of 2000 miles per hour, and would cost $1.75.

9.2.1.7 Impact on creativity in other fields

Computing has had an impact in virtually every path of life.

- Music composers do not need to copy scores freehand, but instead use software packages which speed up the process immensely. They can also get immediate feedback by listening to a digital performance of their composition.

  There are also packages which aid a composer in suggesting melodies and/or harmonies. Research in the automatic compostion of music continues to have an impact.

- Writers no longer sit at a typewriter, with white-out, carbon paper, ink ribbons, etc. The computer also helps to correct spelling and grammatical errors.

  Researchers have experimented with automatic literature generation, thus far with little success, but poets searching for a rhyme have undoubtedly used a computer.

- Architects and civil engineers can produce aesthetic and complex designs in a fraction of the time required before the computer age. Think of all that goes into the design of a modern hospital, including the electrical and data wiring.

- Computer Aided Design (CAD) software has assisted engineers in producing blueprints, circuit diagrams, industrial plant designs, etc.

- Until the year 2010 people with type 1 diabetes injected themselves with insulin to avoid high blood sugar. Too little insulin resulted in high blood sugar causing complications such as infection, kidney disease, eye disease, etc. Too much insulin caused low blood sugar; patients sometimes died while sleeping when their sugar levels dropped too low. Injection of just the right amount of insulin, even while sleeping, is expected to be possible and in widespread use by the time you read this. A digital device is used to combine data from a continuous blood sugar level monitor, with an insulin reservoir and pump attached to the patient's skin.

9.2.2 Exercises

1. What is meant when we describe a computer as *learning*? What is meant by *machine learning*?

2. Review the Marketing Research information located in this section (Innovation in Science and Business, look specifically under innovations in business). Consider all data gathered about you which may be used to market to you from places like your cable provider, mobile provider, and social media platforms. Were you previously aware this is how your data could be used? Now that you know, what are your thoughts?

3. What are copyright laws and how do they impact open source software and open source textbooks (like this one)?

4. Define the types of Creative Common licenses and identify the license(s) associated with this textbook.

   (a) Abbritution (BY)
   (b) Share-Alike (SA)
   (c) Non-commercial (NC)
   (d) No Derivative Works (ND)
   (e) The license(s) associated with this textbook

5. What is Moore's Law? Identify an issue impacting continued, exponential growth.

9.3 Global Impact on Society

Innovations in computing have had an impact on people and societies throughout the world. This impact has not always been a positive, or beneficial, impact. Computing has enabled breaches of security, invasion of privacy, fraud, identity theft, and many other negative criminal or unethical activities.

9.3.1 Beneficial and harmful effects of computing

9.3.1.1 Legal and ethical concerns

Computing has had an impact on the law, as well as ethics. The protection of *intellectual property*, by copyright and patent laws is discussed below.

- Copyright laws were intended to foster creativity; they ensure that authors and other creative artists are compensated for their work. Computers make it very easy to copy copyrighted works, making it difficult to enforce copyright laws.

- In 1999 a peer-to-peer filesharing network named *Napster* gave users the capability of sharing files. Many people used Napster to share digital audio files, typically in MP3 format. MP3 is a digital audio format which compresses the data significantly, while maintaining good quality sound. Prior formats were too large to be shared easily. It wasn't long before a huge number of copyrighted audio files (songs, concerts, recordings) were

available at no cost via Napster. The recording studios filed a lawsuit against Napster for copyright violation. Napster claimed it had not violated any copyrights because it had not copied anything; it simply made it convenient for users to make illegal copies.

- Patent laws are intended to foster innovation in engineering and scientific areas. A patent gives the developer exclusive rights to produce and market a device or process, typically for a term of 20 years. With the advent of computing, software developers filed for patents. Also, researchers filed to patent *algorithms*. Suddenly the courts were facing numerous cases of patent fraud which were not at all clear. How can one determine whether a program or an algorithm, similar but not identical to a patented program or algorithm, is an illegal instance of patent fraud?

- With the advent of mobile phones and digital communications, the service providers routinely maintained *metadata* on phone calls. Metadata is not the actual content of the call, but involves relevant information such as identity of the caller, identity of the person being called, date, and time of the call. Many consider the retention of metadata to be an invasion privacy, and object to the unethical usage of this data - it is sold as valuable marketing information.

- When a suspect is charged with a crime, that suspect's personal property can be confiscated and used as evidence by the prosecution. Suddenly we are faced with a confiscated mobile phone containing much pertinent information; not only the phone calls, but information on the suspect's location at various times, apps used by the suspect, emails, etc. However, if the phone is locked, there is no law requiring the suspect to unlock the phone, and the suspect is further protected by the fifth amendment. There have been cases where the FBI appealed to the manufacturer (Apple) to provide a key to unlock the phone, but Apple claimed that it could not provide any such key. Moreover, Apple claimed that it would not build a *back door*[28] into future phones, as this would compromise the security that it wished to provide to all customers.

- Related to the above item, the primary law in the United States governing access to private cellular phone data is the Electronic Communications Privacy Act of 1986(ECPA). This act states that "some information can be obtained from providers with a subpoena; other information requires a special court order; and still other information requires a search warrant." The PATRIOT Act of 2001 was passed in response to the terrorist attacks of September 11 of that year. It eased restrictions on wiretaps

[28] A back door on a cryptographic system is a built in key, maintained by the developers of the system, and unknown to the users. With a back door, the developer can decrypt anything encrypted by a user. Experts, such as Harvard professor Bruce Schneier, have repeatedly decried the dangers and threats to security afforded by back doors.

and surveillance of US citizens, and was extended for several years, finally expiring in 2020.[29]

- Social media originally allowed anyone to post information as long as it was not considered pornography, slander, etc. Recently social media corporations such as facebook have come under attack for permitting false, and potentially damaging, information to be posted.

  For example, in 2018 it was revealed that the British political consulting firm Cambridge Analytical had been granted access to large amounts of facebook data, and used it to discredit politicians running for office.

  It was also claimed that other foreign countries, most notably Russia, used facebook to influence the 2016 US presidential election. Since that time, facebook, and other social media corporations (instagram, twitter, e.g.) have taken stronger measures to censor, or limit, what kinds of things can be publicly shared.[30]

9.3.1.2 Peer-to-peer networks

A *peer-to-peer* network has no central authority. Each member stores all the necessary software and data to be shared by peers on the network. Napster, described above, is an example of a peer-to-peer network.

Another example of a peer-to-peer network is a digital currency known as *Bitcoin*. The Bitcoin software, developed in 2008 by Satoshi Nakamoto,[31] Bitcoin is, as of the writing of this book, a huge success, though the value of a Bitcoin has fluctuated drastically, as a result of speculation. Bitcoin users download the *wallet* software, and purchase Bitcoins, or fractions of a Bitcoin[32] from a Bitcoin dealer on the internet. Bitcoins can then be used for legitimate purchases from any vendor which accepts Bitcoin. All transactions are verified by Bitcoin miners using cryptographically secure algorithms.

Since all Bitcoin transactions are anonymous, Bitcoin has been used for drug deals, money laundering, and ransomware. Since there is no central authority for Bitcoin, as there is with a bank, the FBI is unable to file charges against Bitcoin.

9.3.1.3 Legal and ethical aspects of public information

Access to public information, whether it be anonymous or authenticated,[33] raises legal and ethical concerns. Is it ethical to use a government repository

[29] Source: Kirkpatrick, K, "Who Has Access to Your Smartphone Data?", *Communications of the ACM*, October 2020.

[30] For a description of how facebook decides what should be censored, see the New Yorker magazine, Oct. 19, 2020.

[31] Satoshi Nakamoto is the name of the author of a paper presented at a virtual conference. No one has ever met Nakamoto face-to-face. It could be a pseudonym for a group of people, or a completely fictitious name.

[32] A Satoshi is 10^{-8} Bitcoin

[33] *Authentication* is the process of establishing a user's true identity. When you tell the bank your social security number, you are undergoing the authentication process.

of information to aid marketing efforts? To discover your neighbor's shopping habits? To discover which doctors have been sued for malpractice? Is it legal to break in to a government database, to find out who is being audited for income tax purposes?

9.3.1.4 Censorship

When is censorship of digital information appropriate? Does the first ammendment of the U.S. constitution (freedom of speech) give you the right to post anything you like on Facebook? Most legal scholars would say that freedom of speech is limited; speech, or expression, which infringes on someone elses rights is not allowed by the first ammendment. Thus, social media, such as Facebook, Twitter, and Instagram, go to great lengths to censor postings appropriately.

In addition, social media generally have an interest in limiting inappropriate postings which they consider to be offensive to most people. This decision is made to ensure that they maintain a strong user base, but it complicates things for the social media corporations, because they are global organizations, spanning many regions and cultures. What is acceptable to one culture may be very offensive for another culture.

Some non-democratic governments will censor information on the internet to disallow expression which is critical of the government or deemed otherwise inappropriate by the government.

The Arab Spring was a series of uprisings and protests, beginning in Tunisia in 2011, in which people discontented with the status quo used social media to organize large protest demonstrations. More recently there have been fewer organized protests, probably because the governments have utilized censorship to maintain control.

9.3.1.5 Open source

To fully understand the phrase *open source* we must first describe the origin of the word *source* in this context. A *compiler* translates a program written in a high-level language, such as Java, into a series of binary coded instructions, a language known as *machine language*. The original Java language version of the program is known as the *source* version or *source code*. The source code can be read and modified by programmers but cannot be directly executed by a computer. The machine language version, also known as *executable*[34], cannot be read or understood by programmers[35], but can be directly executed by the computer. We can now distinguish between proprietary software, open software, and open source software.

[34] A file with a .exe extension is typically an executable, machine language, program.

[35] There is software which can *decompile* binary machine code, to a more readable form known as *assembly language*, or even to a high level language such as Java. This process, known as *reverse engineering*, does not produce the original source code which would have meaningful names for data, and English descriptions of the code.

- *Proprietary software* is software which can be purchased. The customer typically purchases the machine language version of the software, not the original source version. The developers of the software maintain private ownership of the source code, so that it can be improved by only the original developers.
- *Open software* is available free of charge, but only the executable version is available. The developers maintain private ownership of the source code.
- Open *source* software is software for which the source code is available, generally free of charge. Users are then welcome to make improvements and compile the source to obtain an executable version.

There are several varieties of licensing options available for open source software.[36] The original developers of open source software are not directly paid for the time and effort put forth, but they may, individually, or collectively, market their services and expertise to those who are using the software. For example the open source linux operating system has become so popular that companies such as Red Hat offer consulting services which provide support for linux users.

Open source software raises some interesting ethical and legal issues:

- If someone violates the Creative Commons license, is that a violation of copyright laws?
- If open source software causes financial, or other, losses, can the developers be sued? Faulty software controlling safety-critical systems such as aircraft, x-ray machines, and medical equipment can cause injury or death.
- Who should be named in the suit if many individuals have contributed to the developement of faulty software?
- Can documentation in the software be legally copied to develop competing software? A famous example of this legal question is a lawsuit for $8.8 billion filed by the Oracle corporation against Google in 2010. Oracle's Java programming language, including its API, [37] is open source, with restrictions on use. Oracle claims that Google is making unfair use of that API in its Android operating system. This lawsuit eventually came to the U.S. Supreme Court, which was to have heard the case in March 2020, but postponed it because of the COVID-19 pandemic.

9.3.1.6 Privacy and security

Privacy and security have become more important in today's world. Identity theft occurs when any information that can be used to identify a person is

[36] See Creative Commons, above, in this chapter

[37] API,or *Application Program Interface* defines the source-level components of a software system, describing how they can be used.

obtained illegally. Sometimes this information is obtained through legal means, but it can be just as harmful to the victim. Clearly, if someone obtains your credit card number, they can attempt to charge purchases to your account. They can also use identity information to obtain a better credit rating, gain access to your financial accounts, and in general wreak havoc on your life.

There are many instances where invasion of privacy may seem innocuous.

- When you make a personal phone call, or send a text message, who has access to the content of the call? Who has access to the metadata[38] for that transmission?
- When you post information on Facebook or Instagram, or use Twitter, you are providing those companies with valuable information about yourself. You are also exposing this information to the world.[39]
- Your smart phone is equipped with GPS technology. As you travel, your phone knows your location, and that information is useful to the phone provider.
- As you browse and search using a web browser such as Google Chrome, Apple Safari, or Mozilla Firefox, your browsing history is available to the provider. This is valuable marketing information which can then be sold.

9.3.1.7 Anonymity

The question of the legality of anonymous expression has existed at least since the origin of the United States. The US Supreme Court has repeatedly recognized rights to speak anonymously derived from the First Amendment (freedom of speech). The internet makes anonymous speech convenient, valuable, and potentially dangerous.

When creating an email account, for example on Gmail, one does not need to provide one's actual identity (name, residence, ssn, etc) but Google does store some information about you: your IP address, which is required whenever accessing the internet.

9.3.1.8 Exploitation of information by social media, and cable providers

Technology anables the collection, and exploitation of information on individuals and other entities. When you post something on Facebook, even a like or dislike, you are providing Facebook with valuable information. It may not seem like much, but in the aggregate over millions of users, this information is valuable marketing information provided to Facebook.

This exploitation is also prevalent with cable providers. Many people contract with the large cable providers, such as Comcast and Verizon, which offer

[38] The metadata consists of the date, time, caller id, called id, etc. but not the actual content of the voice or text transmitted.

[39] Facebook, Instagram, and Twitter may claim security and privacy features, but these claims have not always been as robust as one might wish.

voice, television programming, and/or internet acceess. When you choose a TV channel to watch, the provider makes note of it. When you mute, search, turn on closed captions - this is all valuable information for the provider.[40]

9.3.1.9 Exploitation of information by web services

Web searches and browser histories provide valuable information to companies providing search engines, such as Google or Yahoo. Knowing who is searching for what particular kind of apparel, their demographics (geographic location, age, sex, occupation) is information maintained by Google, and sold to various vendors and marketing information services. Did you search for a restaurant on Google Maps? Google will remember this, and make use of it. Did you take a selfie in Paris? Apple will remember this, and make use of it.

More recently, Google has been profiting immensely by selling advertising. The advertising can be targeted to a user's historical preferences.

9.3.1.10 Threats to curators of information

Governmental curators of information include the Social Security Administration, Medicare/Medicaid, the Internal Revenue Service, the various cabinet departments: State, Defense, Energy, etc.

Private curators of information include banks, insurance companies, credit databases, and retailers.

Each of these agencies is responsible for maintaining large amounts of data necessary to meet their responsibilities and/or operate their business. This data is usually sensitive and confidential. A security breach could be devastating. A few examples of security breaches are:

- In December, 2013, the Target retailing coporation revealed a security breach in which up to 110 million customer accounts had been accessed. Hackers[41] gained access to customers names, credit/debit card numbers, card expiration dates, and CVV security codes.

- A *credit bureau* is a data collection agency which gathers account information from creditors and sells that information to consumer reporting agencies. This information us used to determine whether an individual is a good credit risk for loans and purchases. In the summer of 2017 the American credit bureau Equifax reported a security breach. The private records of more than 140 million individuals had been compromised. This was one of the largest instances of identity theft in history.

[40] Some people think of cable TV as a means to 'vote'; if you don't like the political slant of a TV news broadcast, you switch to a different channel. If many people do this, the TV network gets the message.

[41] The term *hacker* was originally used in the early days of computing to be a person who worked late into the night developing software and/or learning about existing software. Today the word takes on a more nefarious connotation as one who illegally breaks into secured sites or databases.

- Security breaches also occurred at government agencies. in April 2015 the U.S. Office of Personnel Management[42] discovered a data breach which occurred some time in 2014. Approximately 21 million personnel records were illegally accessed. Information such as social security numbers, names, date and place of births, residences, and even fingerprints were compromised. The victims included former government employees, and even those who had undergone a government background check.

9.3.1.11 Targeted advertising

As mentioned above, advertising can be targeted depending on the demographics of the users. This can have negative consequences; many people feel that this kind of advertising is an invasion of privacy. Advertisements on web sites can be distracting and/or obscure the important information. Users often have an option to block ads, or to selectively block ads. Also, there is ad-blocking software which purportedly will prevent ads from popping up. Most users are not aware, that when they click on an option to block ads, they are providing their IP address, and possibly other information, to the web server, and possibly the advertising entity. A worst-case scenario is that the link to block ads is actually malware which can infect the user's computer.

9.3.1.12 Intellectual Property

Intellectual property encompasses original writings, data, analyses, and other artifacts created by an individual for various reasons. When others access this information, in a way that is not intended, it is considered an ethical violation. If the information is copyrighted, unauthorized access is a violation of copyright laws, and is punishable by law.

Most users of social media will share intellectual property with the social media servers, and in consequence, with everyone in the world. This often has negative consequences. The posting of offensive images or writing is one example. Social media will often claim that the distribution will be limited to a restricted access list of users. However, the security systems can be, and have been, fallible. It is not recommended that people share sensitive intellectual property on social media. When sending sensitive informaton via email, it should be encrypted.

9.3.1.13 Copyright of derived work

In copyright law, a *derivative* work is one that includes elements of a copyrighted original work. The derivative work becomes a second and separate work from the original, and my be governed by a separate copyright.

Some examples of derivative works are translations, movie adaptations, move remakes, and musical arrangements.

[42]The Office of Personnel Management maintains records of civil servants - non-military government employees.

Whether the original copyright holder is protected from derivative works depends on the particular copyright laws of the country in which they are applied. Also, it is possible to specify in a copyright whether derivative works are permitted.

With the Creative Commons licensing system described above, derivative works are permitted, unless the license specifies ND (No Derivative Works). ND may be specified in combination with the other rights, such as BY (Attribution), SA (Share Alike), and NC (Non-commercial).

9.3.1.14 Digital Millenium Copyright Act

We have already described the issues of file-sharing systems such as Napster, with respect to copyright laws. The *Digital Millenium Copyright Act* (DMCA) of 1998 was a direct result of the problems caused by these file-sharing systems. The DMCA made it illegal to produce and/or distribute artifacts which *intended* to circumvent existing copyright laws. The question of intent was left open to interpretation by the courts. This law also held accountable the internet service providers (Comcast, Verizon, AOL, etc.) in cases where they enabled illegal distribution of copyrighted materials.

9.3.1.15 Open and open source artifacts

The internet has become a vast source of valuable information, including documents and programs, often referred to as *open* software. Here we distinguish between open and *open source* software.

- Open software is available free of charge.[43] Examples of open software include:
  - Software distributed free in *binary* or *executable* form. This is the output of a compiler, intended to be executed on a particular machine. It is unintellible to humans[44] and thus is not amenable to edits, corrections, or additions.
  - A document in PDF format is another example of open software. It can be viewed, copied, and distributed (pursuant to the particular license), but it is not feasible to make edits to the document. An open textbook is usable by students, but not amenable to corrections or additions by anyone, including the teachers or faculty who may have adopted the book.

  Think of open software as similar to a free CD-ROM storing music.

[43] Some examples of open software may impose a slight processing or storage fee, but are still considered open.

[44] It is possible to *reverse engineer* an executable program, converting the binary code to readable text, using software known as a disassembler, or a decompiler; however the resulting text is usually not sufficent to make significant improvements to the original software.

- Open source software is also available free of charge, but the original source documents which were used to create the software are also available. This means that anyone who obtains the software can make edits, corrections, or additions, presumably to improve the software. Open source programs include not only the binary files, but the source files used to create the binaries, and all relevant documentation needed to further develop the software.

  Software which is open source is subject to a process known as *forking*. When an individual or entity uses the entire package of source programs to produce a separate product for distribution, we say the project has forked. This means that there are two separate development efforts taking place, and the product may evolve into two distinct versions, the original version and the forked version. Users, and other developers, then have the option to choose the one which best meets their needs. If most choose the forked version, it will thrive, and the original version may wither. If most continue to choose the original version, it will continue to thrive, and the forked version may wither. It is by this natural selection process that open source software has the potential to achieve a high degree of quality.

  Examples of open source software include:

  - The operating system known as Linux, first released in 1991 by the Finnish-American software engineer Linus Torvalds. Originally developed for the Intel x86 processor, it now runs on many different platforms. Companies, such as Red Hat, have evolved to support Linux.
  - Google Chromium is open source software which is a web browser. The source code enables users to develop web browsers with more features. One such improved browser, known as Chrome, was also produced by Google.[45]
  - The Java programming language was originally developed by Sun Microsystems. The software consisted of the compiler, interpreter,[46] supporting libraries, and debugging software. Java was open, but not open source. When Sun was acquired by the Oracle Corporation in 2009, Java became open source.
  - LibreOffice is a package of programs, similar to Microsoft Office, for word processing, spreadsheets, presentation, etc. It was forked from OpenOffice in 2010. LibreOffice can read and produce documents in Microsoft Office format. LibreOffice claims to be 100% compatible with Microsoft Office.[47]

[45] Chrome is not open source

[46] The Java compiler does not compile to native machine code; it compiles to *byte code*, which is then executed by an *interpreter*. The byte code is standard across all platforms. This provides Java software with *portability* one of its strongest features.

[47] It is a mystery to this author why entities in the public and private sectors continue to outlay money for Microsoft Office.

- Textbooks, such as this one, which make the original source files available, in addition to the PDF, are open source.[48] Authors of open source textbooks include:
  - Allen Downey of Olin College of Engineering has authored at least 7 books on programming, data structures, and statistics, under the umbrella of Green Tea Press.[49]
  - Seth Bergmann of Rowan University includes this book as his fourth open source textbook.[50]
  - David Eck and Carol Critchlow, of Hobart William Smith College have produced an open source textbook on Computer Science Theory.
- Open source textbooks are amenable to improvement by anyone with the time and interest in doing so. People can correct errors, include their favorite exercises, and provide entire sections or chapters for addition to an existing textbook. This also means that open source textbooks can be *forked*.

9.3.2 Exercises

1. This section of the chapter opens with a statement warning that with all the great things that come from our developing, digital world, there are negative consequences (like data breaches, fraud, privacy invasion, identify theft). Can you find a current event article (within the past year) that highlights a negative impact of technology and answer the following:

   Summarize what happened. Identify the group(s) of people negatively impacted. Identify if anyone benefitted from the event.

2. Explain what a peer-to-peer network is and provide an example of a peer-to-peer network.

3. This section of the book explores censorship, which continues to be a complicated, and often debated topic. Can you think of an example in which you feel censorship is appropriate? Please explain why.

4. The "Privacy and Security" section of this chapter clearly identifies areas where your privacy could be invaded/compromised every day! Prior to reading this section, did you consider that these are areas of privacy concern? Will you do anything differently now to protect your privacy?

5. What is *intellectual property*?

[48] See *Publishing Research Quarterly*, Vol. 30, No. 1, March 2014, for a more complete description of Open Source Textbooks.

[49] Although the source files for Prof. Downey's works are freely available, copies are also distributed for a nominal fee by Amazon.

[50] See `http://rdw.rowan.edu/oer`.

6. Regarding copyright of derived works:

   Define derivative work. Provide an example of a derivative work. Explain how you know whether it is legal to copy a derivative work.

7. What is the Digital Millenium Copyright Act?

8. What is the difference between open software and open source software? Provide an example of open software which is not also open source, and provide an example of open source software.

9.4 Social contexts for innovations in computing

Innovations in computing do not take place in a vacuum. New developments are often the result of changes taking place in our society. Conversely, new developments in computing have an impact, and cause economic and cultural changes in our society.

9.4.1 Contexts

A context is the milieu in which something occurs. Innovations in computing occur within the social, economic, and cultural contexts of our society. Computing innovations in one country, or socioeconomic context, can be very different from the innovations which occur in a different country, or context.

9.4.1.1 Reaction to a pandemic

One example of computing innovations occurring as a response to events in a social context has to do with the COVID-19 pandemic of 2020. Prior to this time there was web conferencing software which allowed people to share voice, video, and computer screens across the internet. However, these systems were used primarily in some business contexts, and most business meetings continued in a face-to-face manner. Simpler systems, such as Skype and Facetime were popular, but did not have the features needed for conferencing.

The pandemic changed all this. Suddenly educational, health-care, financial, legal, and other environments were suddenly faced with the need for high-quality conferencing systems. More innovative systems, such as Zoom, Cisco Webex, and Blackboard Collaborate soon became ubiquitous. These systems underwent substantial revision to meet the much greater bandwidth (i.e. usage) and functionality (features) requirements.

9.4.1.2 Cellular telephone technology

An example of a context which has been affected by innovations in computing is the advent of cellular telephone technology in third world countries. In some countries, African in particular, the poor economy did not allow for the construction of a nation-wide telephone system (entailing connections with wiring

to all inhabited regions); it was simply too expensive. In the late twentieth century, with the advent of cellular technology, many of these countries went directly to cellular systems, bypassing telephone poles and wires entirely. These countries have stopped trying to establish new landlines, as the cellular system is more cost effective.

9.4.1.3 Distribution of computing resources

Computing resources around the world are not uniformly distributed. People living in wealthy nations have convenient access to the internet, and all that implies. These people can communicate freely with friends, relatives, and total strangers. They also have greater access to entertainment, games, music, sports, gambling, etc.[51]

In many countries lack of access to computing is due to economic considerations. In some countries access is intentionally limited by the governing officials to maintain authoritarian control of the government.[52]

9.4.1.4 Digital Divide

The *digital divide* is the uneven distribution in the access to, use of, or impact of information or computing technologies, among distinct groups of people. This phrase was first used by Larry Irving of the U.S. National Telecommunicationd and Information Administration in 1999. People in lower socioeconomic brackets typically have minimal, if any, access to computing.

The most direct result of this lack of access has to do with the availability of information. People cannot discover helpful resources, because the resources are generally available on the internet; informatiion on resources such as medicare, medicaid, affordable health care, food assistance, educational scholarships, etc. is available primarily on the internet.

A less direct consequence of the digital divide is that lower-class children entering school are lagging behind middle and upper-class children in ther ability to use digital computers and devices.

9.4.1.5 Infrastructure

Who owns the internet? Who ensures that it is secure? Who is responsible for its maintenance?

The internet is supported by a combination of commercial (private) and government (public) entitites. The protocols and software were first developed in universities, and funded by DARPA (the Defense Advanced Reserch Projects Agency), an arm of the Department of Defense. Today the National Science Foundation continues to support research and development which contribute to the security and stability of the internet.[53]

[51] Are these really a benefit of greater access to computing?
[52] See the discussion of the Arab Spring uprising, above
[53] Just imagine the impact to our society if the internet were to 'crash'

The software and algorithms which are the internet are in the public domain. They are accessible to everyone. The security lies in the cryptographic algorithms which are used for authentication and confidentiality.

9.4.2 Exercises

1. This section begins in noting that computing innovations do not occur in a vacuum. In what contexts do computing innovations occur?

2. Innovations in video conferencing systems were sparked by necessity in response to the COVID-19 pandemic of 2020-21. Video conferencing became not only an integral part of education but an integral part of daily life. Based on your own experience, can you describe other ways you used video conferencing technology as part of your life/routine.

3. Define digital divide. Explain how you think the digital divide impacted access to virtual education for students of low socioeconomic status during the COVID-19 pandemic of 2020-21.

4. Consider your daily reliance on the internet – maybe you need it to do your job, to go to school, to order food. How would an internet "crash" for a long period of time (think a month) impact your daily life?

9.5 Research

Research is an investigative process in which there is generally a specific question to be answered, or a specific hypothesis to be tested. The research process is facilitated, accelerated, and generally improved when the appropriate technologies and tools are used. The research process will often involve accessing information, and evaluating the credibility of the sources of that information.

9.5.1 Information management

The research process will involve:

- Accessing the necessary information
- Managing that information
- Maintaining accurate attribution of the information (i.e. citations)

9.5.1.1 Online sources

Traditionally, information used in the research process has been primarily peer-reviewed articles in journals and other periodicals. Reliability was generally assumed to be rather high, if not perfect. Today most of these journals are online, in addition to being bound hard copies. However, a number of other sources are also online. Unfortunately they are often:

- Not peer reviewed - The peer review process involves finding experts willing to take the time to review a submitted manuscript. Once the reviewers have been obtained, it could be several months before all the reviews are complete. Peer reviewing improves the quality of publications but has a major impact on the time it takes for a submitted manuscript to be published.

- Not permanent - When something is posted online there is no guarantee that it will persist. The article could be modified, or even removed, at any time. This would have a serious impact on the credibility of all subsequent research which has made use of the temporary article.

- Difficult to cite - All research making use of existing information should list citations appropriately. If the only citation is a URL, the citation lacks credibility.

9.5.1.2 Search tools

Years ago the best search tool that researchers had was known as the *Readers Guide to Periodical Literature* which was a hard-bound reference updated monthly. It contained citations for every peer-reviewed journal in the sciences and social sciences. It had copies sorted by author, title, and subject to facilitate quick search.

Modern search tools can be more helpful in finding specific information quickly. Boolean criteria, and search categories, combine to afford logical expressions that expedite a search.

For example, we may be investigating the possible causes of a disease that we believe is carried by a mosquito. We know that John Smith and Harriet Jones have previously published articles on this disease. We could generate a search expression such as:

```
FIND: AUTHOR="Smith, John" OR "Jones, Harriet" AND
      SUBJECT="COVID-19" OR "CORONA" OR "Mosquito" AND
      DATE=1990 - PRESENT
```

9.5.1.3 Plagiarism

Easy access to online information is a double-edged sword. It has many advantages, discussed above, but it is also vulnerable to plagiarism. Plagiarism is the unauthorized use of information produced by others, generally done in such a way that it appears the information was originally produced by the plagiariser. Plagiarism, whether it is scientific research or a school term paper, is a serious offense. If the plagiarised information is copyrighted, the plagiarism is a violation of copyright laws and is a punishable offense.

Many teachers and college professors forbid the usage of online or non-peer reviewed materials. Most, however, will insist on appropriate citations for anything which is obtained, online or otherwise. A scholarly work which lists many

URLs for the citations will have less credibility than a work which cites peer-reviewed publications.

To help researchers establish the credibility of a citation, the International Standards Organization (ISO) now approves a standard scheme known as DOI (Digital Object Identifier).[54] A published article is assigned a DOI, which is independent of the article's location or URL. The article's URL can change, but the DOI continues to provide access to the article. A DOI aims to be 'resolvable' to some form of access to the article to which the DOI refers. Referring to an online document by its DOI supposedly provides a more stable link than simply using its URL. An example of a DOI for an article on DOIs by Marc Langston is:

`doi:10.1016/j.iheduc.2003.11.004`

This article appeared in the journal, *The Internet and Higher Education* in volume 7, issue 1, of 2004. It can be located by using the DOI website at `dx.doi.org`.

9.5.2 Credible and appropriate sources

Given the nature of online information - it is easily available, temporary, often not peer-reviewed; how can we assess its credibility? How can we determine whether it is appropriate for use as a citation in our current research?

9.5.2.1 Credibility

The *credibility* of a source is a measure of its accuracy, reliability, validity, and permanence. A particular experiment produces *reliable* information if repeated experiments under the same conditions produce the same results. A particular experiment produces *valid* information if the results represent an accurate measure. Note that the reliability and validity are independent. An experiment can be reliable, but not valid. An experiment can be valid, but not reliable. To determine the credibility of a source we should examine:

- The reputation of a source - Have we seen this source cited elsewhere, in many other works. Have colleagues reported good things about the publications coming from this source? Does this source have a long-standing reputation, or did it just begin in the last year?
- Credibility of the author(s) - Do they have appropriate credentials? Do they have advanced degrees in the appropriate areas? Were those degrees awarded by accredited, and preferably prestigious universities.
- Credibility of a web-site - Does the web-site have a long-standing reputation for credibility? In this text we have numerous citations from Wikipedia; we believe that Wikipedia has met most of the critera for

[54] DOI was first developed by the International DOI Foundation in 2000, and is responsible for its maintenance, credibility, and security.

credibility mentioned above. Wikipedia is constantly evolving toward improved reliability, accuracy, and validity.

- Sponsorship, if any - Research sponsored by a large government agency such as the National Science Foundation, the National Institutue of Health, etc. is likely to be credibile; it has already undergone extensive review in order to obtain funding from those institutions. Research from an entity attempting to further its own predetermined viewpoints is less credible. Examples would be political parties, organizations known to be affiliated with a political party, and possibly religious groups.

9.5.2.2 Appropriate information

Simply because an information source is credible does not mean that we should use it. It must also be relevant to our research. If it clearly supports our hypothesis, or includes results which are preconditions for our results, then it is probably appropriate.

9.5.3 Exercises

1. This section of the textbook focused on Research, and it begins with a list of problems associated with online research. List the three main problems; which one do you believe is the most problematic and why?

2. (a) Define plagiarism.
   (b) Most institutions of secondary and higher education have strict policies against plagiarism and consequences associated with plagiarism. Describe your understanding of plagiarism policies and consequences associated with your secondary or higher education institution.

3. (a) Define credibility of a source.
   (b) List the recommended ways of examining source credibility.

4. This section of the textbook was focused on research. Imagine you have to write a research paper on the impact of COVID-19 on mental health. How will the information provided in this section assist you with your research?

Glossary

abstract - Having no evident details; non-concrete

abstraction - The process of separating ideas from specific instances of those ideas

accessor method - A method which returns the value of a field, also known as a getter

actual parameter - A parameter value to be passed to a called method, also known as an *argument*

adder - A hardware component capable of adding fixed-length binary integers

AI - artificial intelligence

algorithm - A well-defined sequence of steps to solve a given problem, which terminates with a correct solution

Amazon - Originally a vendor of books, online. Evolved into other products as a distributor and software services

American Standard Code for Information Interchange - An 8-bit numeric code for each character (ASCII); a subset of Unicode

AND - A boolean operation which results in `true` only if both operands are `true`

API - Application Program Interface

application program interface - The information needed to use a software module, (API)

argument - A parameter value to be passed to a called method, also known as an *actual* parameter

array - A homogeneous collection of values which is mapped directly to the computer's main memory

artifact - Something created by humans, using tools; not occurring naturally

artificial intelligence - A quality or feature of a computational artifact which makes it appear to have natural intelligence (AI)

ASCII - American Standard Code for Information Interchange

assembler - Software which translates symbolic machine instructions programs to binary machine language

assembly language - A version of machine language with symbolic operations and memory locations

assignment - The binding of a data value with a variable

AWS - Amazon Web Services: Cloud computing services such as storage, fast computing, and applications

bandwidth - A measure of the quantity of information which can be transmitted over a communications channel per unit of time; usually measured in bits per second

base - In mathematics, the value of b in b^e; radix

binary - Base 2 number system

binary search - A search algorithm applied to a sorted List, which eliminates half the values from consideration on each comparison

bit - A binary 0 or 1; a binary digit

Bitcoin - A a digital currency which uses cryptographic algorithms to ensure security

Bitcoin mining - The process of creating new Bitcoins computationally

block - A group of statements in an algorithmic procedure

Block - A language for expressing algorithms, using graphic diagrams

byte - 8 bits

boolean - A data type with only two possible values: `true` and `false`

Cambridge Analytica - A British political consulting firm which was involved, with Google, in a scandal during the 2016 US presidential election

client-server - A client makes use of resources provided by a server

cloud computing - Software services, such as applications and data storage provided over the internet

combination - In algorithms, the process of calling a procedure from another procedure

code review - A software development practice in which bugs are discovered by carefully reviewing the source statements

comment - A programmer-supplied description, ignored by the compiler

compiler - A program which translates a program written in a high-level language to an equivalent program in machine language

conjunction - A logical operation which is false only when both operands are false: OR

constant - A data value in a program, supplied by the programmer

control structure - A programming construct enabling an altered flow of execution

core - A CPU within a (personal) computer

crowdsourcing - Usage of the internet to obtain ideas, goods, or services from a large group of people

data compression - The process of manipulating information so that it requires fewer bits

conditional - In an algorithm a decision operation; an IF statement

control structure - In an algorithm or program, a construct which determines which statement(s) are to be executed

data analytics - The application of data science

data science - The study of accessing and interpreting (large) quantities of raw data

data stream - A specification of parallelism involving vector operations (see Flynn taxonomy)

data structure - An organization of data to facilitate operations such as search or sort

database - Software which enables one to organize and access data

De Morgan's Laws - Boolean identities: The negation of a conjunction is the same as the disjunction of the negations; The negation of a disjunction is the same as the conjunction of the negations

debug - To remove the errors from a program

decision problem - A problem which as a solution of either 'yes' or 'no'

decryption - The process of restoring information to its original form after it has been encrypted

digital - The quality of discrete components of information, as opposed to analog

digital logic - Components of a digital device performing boolean operations

disjunction - A logical operation which is true only when both operands are true: AND

DISPLAY - An output statement in algorithms

distributed system - A system in which many computers/users are simultaneously collaborating to share resources

distributed computation - A distributed system designed to solve a computation-intensive problem

distributed file system - A distributed system designed to allow seamless access to file systems on a network

distributed database - A distributed system designed to allow seamless access to multiple file systems on a network

distributed operating system - A distributed system designed to allow seamless access to multiple operating systems on a network

download - Transfer information from a remote, central computer, in the 'cloud', to a local computer or digital device

encapsulation - A feature of object-oriented programming languages in which the internals of an object are not directly available to the client

encryption - The process of modifying information for the purpose of confidentiality, such that it can be restored to its original form

equivalence of algorithms - The determination that two different algorithms always produce the same result and the same side effects for a given input

exception - A software artifact which permits the programmer to trap and handle run-time errors or failures

exponent - The e in b^e

expression - A variable, a constant, or an operation on two expressions

extremum problem - The problem of finding a minimum or maximum value in a List of values

Facetime - A communications application which includes audio and video

fault tolerance - The capability of a computer, digital device, or network to recover from the failure of one or more components

floating point - An approximate data representation for numbers which need not be whole numbers, and which may be very large, or very close to 0

Flynn taxonomy - Nomenclature expressing the type(s) of parallelism in a computer

FOR_EACH statement - An iteration structure in an algorithm, associated with a List

formal parameter - A parameter in a method definition

free format - A lexical property in some programming languages in which white space is ignored by the compiler

G suite - Office applications offered by Google in the cloud, via a web browser

getter - See accessor method

Global Positioning System - System of satellites sending signals to earth, enabling devices to determine their exact location (GPS)

Google - Originally a search engine, evolved into email, computers, tablets, browsers, and cloud computing services

GPS - Global Positioning System

heuristic - A series of steps which attempts to solve a given problem but which may terminate with an incorrect, or approximate, solution

high-level language - A language such as Java, Python, or C++ which enables humans to develop software; a programmming language

IDE - interactive development environment

IF statement - A one-way selection structure in an algorithm

IF-ELSE statement - A two-way selection structure in an algorithm

input - Data supplied by the user of a program

instagram - A photo and video sharing service

interactive development environment - software used to edit, compile, and test programs being developed; IDE

interface - An adapting layer between two or more entities

instruction stream - A specification of parallelism in executing processes (see Flynn taxonomy)

IO - Input and output

iteration structure - A control structure permitting repeated exection of a statement, or block, in an algorithm; a loop

java - A high level programming language developed by Sun Microsystems, later acquired by the Oracle corporation

learning management system - Software serving as an aid for teachers

library - A collection of software modules, such as classes

linkedin - A business and employment-oriented service

List - A Collection in which order is maintained, and duplicate values are permitted

loop - An iteration structure in an algorithm or program

loop body - The statement, or block of statements, to be executed repeatedly in a loop

machine language - The language of binary coded instructions which can executed by the CPU

machine learning - Software or hadware capable of improving its own performance as it executes.

massive open online course - Educational material available free on the internet (MOOC)

Messenger - A Facebook application which can be used for transmission of plain text

method - In Java, a procedure with optional parameters and an optional returned value

MIMD - Multiple instruction streams, multiple data streams, in Flynn's taxonomy for parallelism

MISD - Multiple instruction streams, single data stream, in Flynn's taxonomy for parallelism

mobile computing - Devices such as smartphones and tablets which are easily portable, and which can be used for communication and sensing the environment

MOOC - Massive Open Online Course

mutator method - A method which permits the client to change the value of a field, also known as a setter

nested loop - In an algorithm, or program, a loop defined to be entirely in the loop body of another loop

NOT - A boolean operation which results in the logical complement of its operand

one-way selection - In an algorithm, or program, a control structure with only one possible choice of execution paths; an `if` statement with no `else` part

operation - A calculation on one or two data values, producing a new data value, with possible side effects

OR - A boolean operation which results in `false` only if both operands are `false`

output - Data produced by a program for a user

parallel computing - The simultaneous computation of independent tasks, programs, or instructions

parameter - A variable used to send information to a procedure

pixel - One of the small dots making up an image; a picture element

post-test loop - In an algorithm or programming language, an iteration

structure in which the loop body is executed once before the termination condition is tested

pre-test loop - In an algorithm or program, an iteration structure in which the termination condition is tested before the first execution of the loop body

program - A sequence of binary coded instructions in the computer's memory

programming language - A language such as Java, Python, or C++, which enables humans to develop software; a high-level language

radix - The base of a number system, e.g. 2, 8, 10, 16

RAID - Redundant Array of Inexpensive Disks; a fault tolerant storage system **recursive procedure** - In an algorithm, a procedure which invokes itself

redundancy - The replicaton of a component of a computer, digital device, or network which will allow it to continue processing after a failure of the compenent. Used for fault tolerance.

RETURN - A statement in a procedure which terminates the procedure, and which may return a specific value to the calling procedure

robotics - The discipline in which artifacts with human-like attributes and capabilities are constructed

run time - The execution of a program, as opposed to the compilation

run-time error - An error in a machine language error, detected when the program is executing

scale - A property of a system which performs well when provided with much data to process

search - The problem of finding a given target value in a List of values

selection structure - A programming construct enabling a program to take one of a few possible execution paths

sensor - A digital device capable of gaining information from its environment, such as a motion detector

sequence structure - A control structure in which statements are executed sequentially in the order in which they occur in the program or procedure

sequential computing - Execution of instructions, tasks, or programs one at a time (non-parallel computing)

sequential search - A search algorithm which examines all elements of a collection until the desired value is found, or determined not to be in the collection

client-server - See *client-server*

SETI - Search for ExtraTerrestrial Intelligence

SETI at Home - A distributed computing project in which anyone could participate in SETI

setter - See mutator method

side effect - A change in a program's state, or output, resulting from an operation

SIMD - Single instruction stream, multiple data streams, in Flynn's taxonomy for parallelism

simulation - A process in which a natural or artificial phenomenon is modeled in a computational artifact

SISD - Single instruction stream, single data stream, in Flynn's taxonomy for parallelism

Skype - A communications application which includes audio and video

social media - Communication media which enable people to communicate through text, images, and video

sorting - The process of arranging the values in a collection in ascending (or descending) order

space - Computer memory or secondary storage for data, usually measured in bits or bytes

statement - In a procedure or program, an executable assignment operation, procedure call, IF statement, or REPEAT statement

stochastic simulation - A simulation with probabilistic element(s)

string - Data consisting of a sequence of characters

TCP/IP - Transmission Control Protocol / Internet Protocol

Team - A communications application (Microsoft) which includes audio and video

Text - An algorithm description language using plain text

thread - An independent process which can execute simultaneously with other threads on a computer

Transmission Control Protocol / Internet Protocol - A standard communication protocol on the internet (TCP/IP)

traveleing salesman problem - A shortest path problem in graph theory

truth table - A table showing all possible values of a boolean expression involving several variables

twitter - An internet based service allowing users to post messages known as 'tweets'

two's complement - A binary representation system for negative, as well as positive, whole numbers

two-way selection - In an algorithm or program, a selection structure with a choice of two possible execution paths; an `if - else` statement

upload - Transfer information from a local computer to a remote, central computer, in the 'cloud'.

user interface - Hardware and/or software used for human interaction with a device or program

variable - In an algorithm or program a name representing a memory location which may be assigned a value

virtual reality - Hardware and software which can give a person the the auditory and visual information needed to effect any surroundings

visualization - The process of displaying information or natural phenomenon in a form easily understood by humans

Webex - A communications application which includes audio and video

WeChat - A Chinese version of Messenger

WhatsApp - A text and voice messenging service

wikipedia - A free online encyclopedia in which articles are submitted by experts from all over the world.

world wide web - The linking of documents on the internet

world wide web - The linking of documents on the internet

youtube - An internet based repository for digital video/audio

Zoom - A communications application which includes audio and video

The following glossary entries are for the C++ version of this textbook

! - Logical NOT operator, in C++

& - Bitwise AND operator, in C++

&& - Logical AND operator, in C++

| - Bitwise OR operator, in C++

|| - Logical OR operator, in C++

˜ - Bitwise NOT operator, in C++

application program interface - The information needed to use a C++ class or header (API)

function - A code section designed to be invoked from other functions

if statement - A one-way selection structure in a C++ program

if-else statement - A two-way selection structure in a C++ program

signature - The part of a Java method, Python function, or C++ function defining the access mode, return type, method name, and parameter list

standard template library - STL: A collection of C++ classes and functions which are used for strings, graphics, user interfaces, etc.

void function - In C++, a function with no explicit return value

while statement - In C++, an iteratioon structure which does not specify the number of times the loop body is to be executed; a pre-test loop

Index