Chapter I

Introduction to Computer Science

Chapter I Topics

1.1 Learning the Exposure Way

1.2 Exposure Equation

1.3 Getting Started

1.4 How Do Computers Work?

1.5 Communicating with Morse Code

1.6 Storing Data Electronically with 1s and 0s

1.7 Memory and Secondary Storage

1.8 Hardware and Software

1.9 What Is Programming?

1.10 Networking

1.1 Learning the Exposure Way

One of the most important elements of learning in a classroom is the "student teacher relationship." Who claims that? Me, Leon Schram, a computer science teacher for more than thirty years. If you are a student at John Paul II High School reading this book, your suspicions were correct. Something happened to Mr. Schram back in the Sixties when he was in Vietnam. You might suspect too much exposure to Agent Orange? It has probably caused some brain cell damage. You know the slow dissolving one brain-cell-at-a-time kind of damage. Makes sense, it has been about 45 years since Vietnam and this guy is acting very weird these days.

Many years ago I learned quite a lesson about relationships with students. I had a young lady in my class who had trouble with math. No, let me be honest, this girl was pitiful with any type of mathematical problem. Our school had some type of talent show and I went to see it. I was absolutely blown away. This girl, *my bless-her-will-she-ever-learn-math-student*, had the voice of an angel. She was animated, she was incredibly talented, and suddenly I saw her differently. That talent show knocked me straight out of my tunnel-vision-view of a struggling math student. I told her the following day how impressed I was with her performance. She beamed and was so pleased that I had noticed. Something neat happened after that. Sure, she still agonized in math but she worked, and tried, harder than she had done previously. At the same time, I tried harder also. The teacher/student relationship, which had developed, made her a better student and it made me a better teacher.

I have been a teacher for a long time, and I have been a student for even longer. As a teacher, and as a student, I have seen my share of books that hardly ever get opened. Students try hard to learn. However, they do not get it, get bored, get disinterested, get frustrated, feel stupid, are intimidated.... pick your poison. The book just sits somewhere and becomes useless. Now this particular book may be the most informative, most complete, most up-to-date, most correct book on the subject, but if the book sits and occupies space, the content matters little.

How does an author of a book develop a "student-teacher" relationship? That is tricky business, but if you are really curious what I look like, try my web page at **http://www.schram.org**. It is my aim to write this book in the first person using an informal, verbal style of communication. I want to talk to you, rather than write to you. By the way, I do not recommend this writing style for English-type teachers. I will guarantee you that they are neither amused nor impressed. My English teachers certainly have never cared much for my writing style and I have been told at many occasions that my writing lacks the scholarly flavor *one* associates with college style writing. For one thing, *one* does not use *I*, *one* uses *one*. Do you know how boring it is to keep saying *one* does this and *one* does that? Well, I do not know what *one* does, but I do know what *I* do, so I will keep

using *I* and sometimes *we*. This is not for the sake of glorifying myself, but rather to keep an informal style. Students have told me in the past that it appears that I am talking to them. In fact, I have been accused of writing with an accent. In case you all do not realize it I was neither born in Texas nor the United States. I was born in Holland, moved all over the place, ended up in the United States, took a test in English, US government and US history, and then became an US citizen. Six months after my new citizenship I was rewarded with an all-expenses-paid trip to Vietnam. Today, I have forgotten several languages I learned in my youth, and I cannot speak any language I remember without an accent.

A few more items on this personal relationship business: I was born in 1945. I have been married to the sweetest wife in the world since April 9, 1967. Her name is Isolde. I have four children, John, Greg, Maria, and Heidi. Furthermore, I have a daughter-in-law, Diana, two sons-in-law, David and Mike, and ten grandchildren. My favorite activities are skiing, rock climbing, rappelling, SCUBA diving, ballroom dancing, traveling and writing. Now there is a slight problem. You know a little about me, but I know nothing about you. Well perhaps we will meet one day, and I may also hear about you from your teachers.

By the way, there is a way that I can get to know more about you. You can drop me an e-mail line. Now please read this carefully. You may wish to send me a note and say hi. However, do not write and ask for advice on programs that do not work. You have a computer science teacher. Your teacher knows your assignments, knows your background, and knows your computer science level. Your teacher can help you far more effectively than I possibly can at a distance. Another point, I get frequent e-mail from students requesting to buy some of these books that you are reading. I do not publish paper copies for sale. School districts purchase a copy license, and each school district uses their own method for making copies. You can approach your teacher about acquiring copies from them, if that is an option at your school. Good, now that we have that straight, let me move on.

You may get the impression that all this informal style, and such, may take away from the serious business of learning computer science. Do not believe that for one minute. There are many pages ahead of you and each topic is going to get a thorough treatment. Also do not think, just because this introduction is light-hearted, that everything will be in the same style. Some topics are duller than dirt and you will just have to digest some discomfort and make the best of it. Other topics are pretty intense, and I will do my best to make it clear what is happening, but there are no promises. I will try hard to do my part and you need to do yours. Just exactly what you need to do to learn will come a little later in this chapter.

1.2 Exposure Equation

Somewhere during the time when Cro-Magnon Man told the Neanderthals to take a hike - basically my early Twenties - a neat Sociology professor said something very interesting. What he said ended up having a profound attitude on my teaching and my writing. He claimed that *nothing in life is obvious*, and he told the following story.

Imagine a young boy in the Amazon jungles. This boy has always lived in the jungle without any modern convenience. He has never been in a city and he has never seen a television or seen a book. Now imagine that for reasons unknown this young boy travels to Colorado in the winter time. The little boy stands in a yard somewhere and watches the snow with bewilderment. He is astonished; he does not understand what is falling from the sky. Another little boy, about the same age, from Colorado, looks at the boy's behavior. The young Colorado boy does not understand the other boy's bewilderment. Why is the boy acting so odd, obviously it is snowing, so what is the big deal?

The Amazon boy is bewildered. The Colorado boy is confused that the other boy is bewildered. The professor asked us what the difference between the two boys is, and use only one word to describe that difference. The word is . . . . . .

EXPOSURE

The point made by my sociology professor was so wonderfully simple and logical. If you have zero exposure to something, you will be bewildered. If you have never in your life seen a plane, heard a plane fly by, seen a picture of a plane, heard anybody talk about a plane, you will be one frightened, confused puppy when the first plane comes by.

Here are a couple more examples. When I came to the United States, I had breakfast the way most Americans did. I poured myself a bowl of cereal, usually Corn Flakes, and then proceeded to warm up some milk on the stove. The warm milk was then poured over the Corn Flakes. As someone who grew up in Europe, it was "obvious" to me that warm milk is what you put on cereal. I found it very strange when I realized Americans put *cold milk* on cereal.

Many years later, I am on a cruise with my son John. An English couple sitting across from us looks at my son and the following dialog transpires:

English Man: "I say sir, what is that?"

John: "What is what?"

English Man: "That sir, what is that?" (He points to John's drink.)

John: "Do you mean my iced tea?"

English Man: "Iced Tea? Iced Tea? Good Lord Mildred. Did you ever hear of such a thing?"

English Woman: "My word, no Henry. I think I should like to try some of that."

The point trying to be made with these examples is that exposure theory states that nothing in life is obvious. What we do have are varying degrees of exposure. This means the next time somebody says: *"It is painfully obvious to see the outcome of this situation,"* do not get excited.

Translate that statement into: *"after being exposed to this situation for the last 20 years of my life, and having seen the same consequences for the same 20 years, it is now easy for me to conclude the outcome."*

Well, this good sociology professor – I wish I remembered his name and could give him credit – impressed me with his *obvious-bewilderment-exposure* theory. Based on his theory I have created an equation:

Bewilderment + Exposure = Obvious

This equation states that **bewilderment** is where you start. With zero **exposure** it is not logical to immediately comprehend a new concept, let alone consider it to be **obvious**. However, let enough **exposure** come your way, and yes, you too will find something **obvious** that was confusing at the first introduction. This means that you need some special sympathy for your first instructor and your first book. I have had that theory work for, and against me. Students have come to me and said, "it made so much more sense to me when it was explained in college." I have had the opposite claim with, "if only my first instructor had explained it as well as you did I would not have had so much trouble."

So just what is the point here? Something very weird happens with academic learning. Students, and I mean students of all possible ages, in all kinds of academic topics, expect understanding on the first go-around. Students open their books, and with some luck read a topic once. Perhaps a second, brief reading occurs immediately before an examination. And more than once, I have been told *you told us that yesterday* when I am repeating an important, probably confusing, point that requires repetition.

Now let us switch the scene to athletics, band, orchestra, cheerleading and the drill team. How many band directors practice a half-time show once? Have you heard of a drill team director sending girls home after they rehearsed their routine for the first time? Seen anybody on the swim team get out of the pool after one lap lately, claiming that they know that stroke now? How about basketball players? Do they quit practice after they make a basket? Do the cheerleaders quit the first time they succeed in building a pyramid? You know the answers to these questions. In the area of extra-curricular activities, students get *exposed to death*. As a matter of fact, some of this exposure is so frequent, and so severe that students are lucky to have any time left over for some basic academic exposure. But guys, learning is learning, and it does not matter whether it is physical, mental, artistic or everything combined. You cannot learn anything well without exposure. And yes, in case you have not noticed by now, I believe so strongly in this philosophy that I call these books *Exposure Computer Science* just like my previous text books that were called *Exposure Java* and *Exposure C++*.

I am harping on this topic because I believe that so many, many, students are capable of learning a challenging subject like computer science. The reason that they quit, fail, or hardly even start is because they interpret their bewilderment as an indication of their aptitude, their inability to learn. One little introduction, one short little exposure and a flood of confusion overwhelms many students. Fine, this is normal; welcome to the world of learning.

Let me borrow from my favorite sport, skiing. I have watched a whole bunch of people learn to ski. I have taken many students on ski trips and I have taught a fair number of people of different ages to ski. You know, I have never seen anybody get off the lift for the first time and carve down that bunny slope like an Olympic skier. I have seen many people stumble off the lift wondering what to do next. I have watched these brand-new skiers as they ski down for the first time all bend over, out of balance, skis flopping in the breeze, poles everywhere, and usually totally out of control. A close encounter with the snow normally follows. Fine, you did not look so swell on the first time down, or even the first day, or maybe the first trip. After people's first taste of skiing, I have seen many determined to enjoy this wonderful sport and I have watched as they made fantastic progress. I have also seen other people who quit after minimal practice, and concluded they were not capable of learning to ski. Normally, the boring, obligatory list of excuses comes attached free of charge.

This means whether you are a high school student, learning computer science for the first time or a teacher, learning Python after Java, you can learn this material. You can learn it well, with surprisingly little pain, provided you do not self-inflict so much permanent pain with all this business of self-doubt and *I cannot do this* or *I cannot do that.* So you are confused. You do not understand? Great, join the crowd because the majority of the class is confused. Open your mouth, ask questions, hear the explanation a second time around and allow time to flow through your brain. There is not a channel you can change ... a button you can push ... a pill you can take ... or a specialist you can pay, to soak up knowledge easily. You need exposure, and surprise ... *exposure takes effort*. Ask a champion swimmer how many laps they have completed? Ask anybody who is accomplished in any field how they arrived. Let me know if they arrived on day one. I want to meet that extraordinary person.

1.3 Getting Started

Getting started with computer science is none too easy. Keep in mind that *computer science* is not *computer literacy*. *Computer Science* is also not *Computer Applications*. *Computer Science* is essentially *computer programming*. The course that you are taking, and the book that you are reading, assume that this is your first formal computer science course. Furthermore, it is also assumed that you have no knowledge of programming. If you do know some programming, fine, but it is not any kind of a prerequisite. This means that we should start at the beginning. However, does the beginning mean an explanation like: *this is a monitor*; that *is a printer*; *here is the power button*; *there is the network server*? Probably not. Today's high school students usually have been behind a variety of computers since elementary school. Many students have heard stories on how computers make mankind's life simpler. Did you know how long it used to take to fill out useless paperwork? Today we can finish ten times the useless paperwork in a fraction of the old time with the aid of computers. Progress is good. Anyway, this chapter will provide a brief computer history and provide some information about computer components, networking and related topics. However, the primary focus of this chapter will be on understanding how the computer works. What makes it tick? How does it store information? How does it manage to calculate, and how is information stored? And, what is a program anyway? Basically, you will get introductory information necessary to start learning computer programming.

1.4 How Do Computers Work?

Human beings do not spend money on expensive items unless such items somehow improve human capabilities. Cars are great. They move faster than humans, they do not get tired, and they keep you comfortable in bad weather. They are expensive, but the expense is worth it. Computers process information and do this processing better in many areas compared to human beings. The three areas in which a computer is superior to a human being are shown in Figure 1.1.

Figure 1.1

3 Areas Where Computers are Superior to Human Beings

- Computers are faster
- Computers are more accurate
- Computers do not forget

You may be quick to accept that *computers are faster*, but you are not so sure about the other two. Too often you have heard the term *computer error* and you also remember hearing about data that was lost in the computer.

Well, let us start our computer lesson right now by clearing up some basic myths. *Computers do not make errors.* Sure, it is possible for a computer to give erroneous information. However, the computer is nothing but a stupid machine that faithfully, and always accurately, follows instructions. If the instructions given by a human being to a computer are faulty, then the computer will produce errors. At the same time, many so-called *computer errors* are caused by sloppy data entry. A person who receives an outrageous electric bill is told that the computer created an erroneous bill. True, the computer printed the bill, but not until a data-entry clerk had slipped an extra zero in the amount of electricity used for the previous month.

Perhaps you are still not convinced. After all, what about the situation when a computer breaks down? Won't that cause problems? Broken computers will certainly cause problems. However, your computer will not work at all. Your computer applications will not work and you are stuck, but the computer does not suddenly start adding **2 + 2 = 5**.

You may also have heard that people lose their computer information because of problems with disk drives. Once again this happens, but computer users who keep their computers and diskettes in a proper environment, along with a sensible backup system, do not have such problems.

With everything that we see computers do today, it is not surprising that some people think that computers are also more intelligent than human beings. Yes, computers can do amazing things, but what must be understood before we go on is that COMPUTERS ARE STUPID. They have no intelligence. They also have no creativity. All a computer can do is to follow your directions.

Well, you give up. No point arguing with a stupid book that cannot hear you. Fine, the computer is *faster*, the computer is *more accurate*, and sure the computer *does not forget*. But how is this managed electronically? You know that electricity is incredibly fast, and you have every confidence that the flip of a switch turns on a light or a vacuum cleaner. Today's computers are electronic. Just how does electricity *store information*? How does a computer perform *computations*? How does a computer translate keyboard strokes into desirable output? These are all good questions and an attempt will be made here to explain this in a manner that does not become too technical.

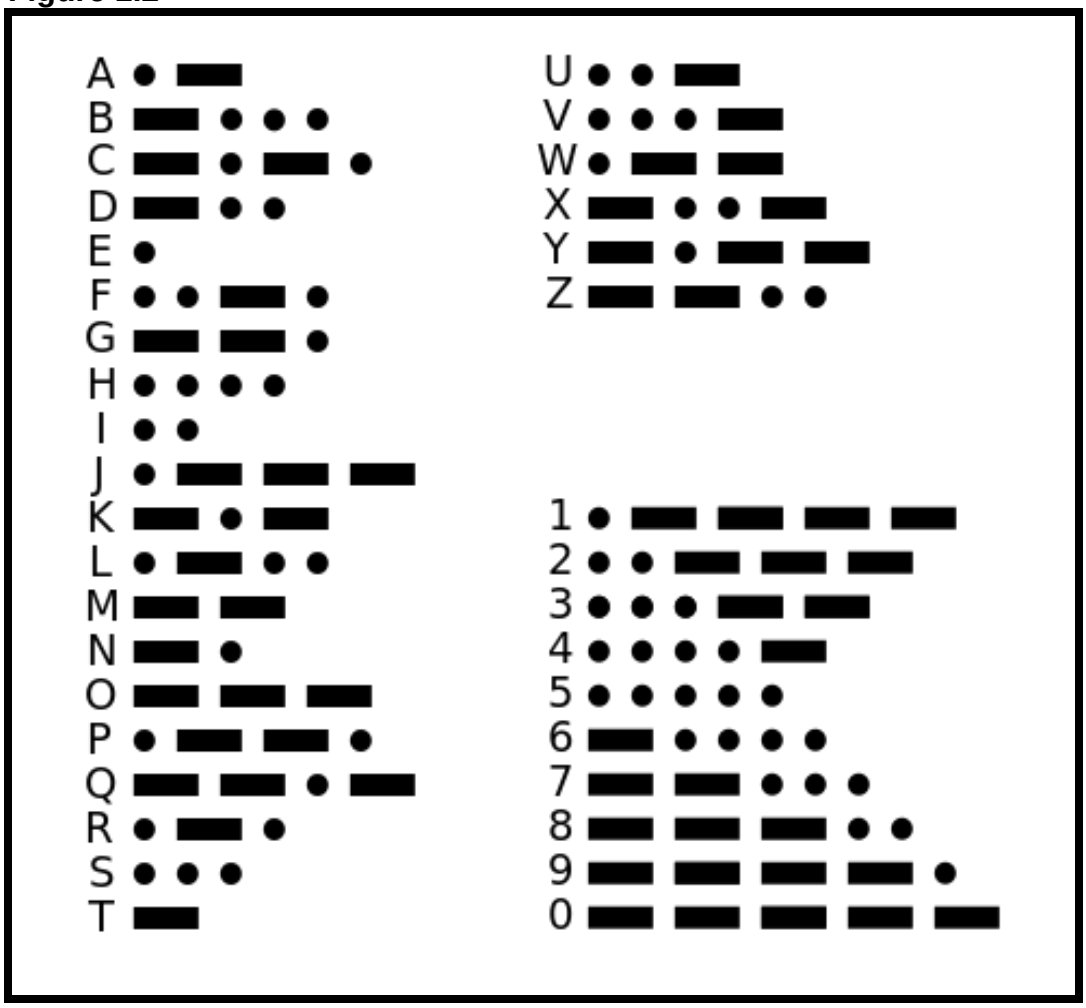
1.5 Communicating with Morse Code

Unless you are a Boy Scout or Navy sailor, you probably have little experience with *Morse Code*. Today's communication is so much better than Morse code, but there was a time when Morse code was an incredible invention and allowed very rapid electronic communication.

Imagine the following situation. Somehow, you have managed to connect an electric wire between the home of your friend and yourself. You both have a buzzer and a push button. Each of you is capable of "buzzing" the other person, and the buzzer makes a noise as long as the button is pressed. You have no money for a microphone, you have no amplifier and you have no speakers. Furthermore, your mean parents have grounded you to your room without use of the telephone. But you do have your wires, your buzzers and your buttons. Can you communicate? You certainly can communicate if you know Morse code or develop a similar system.

Morse code is based on a series of *short* and *long* signals. These signals can be sounds, lights, or symbols, but you need some system to translate signals into human communication. Morse code creates an entire set of short and long signal combinations for every letter in the alphabet and every number. Usually, a long signal is three times as long as a short signal. In the diagram shown in Figure 1.2 a long signal is shown with a bar and a short signal is indicated by a circle.

Figure 1.2



You, and your buddy, can now send messages back and forth by pressing the buzzer with long and short sounds. Letters and numbers can be created this way. For instance the word **EXPO** would be signaled as follows:

The secret of Morse code is the fact that electricity can be *turned on*, and it can be *turned off*. This means that a flashlight can send long and short beams of light and a buzzer can send long and short buzzing sounds. With an established code, such as Morse code, we can now send combinations of long and short impulses electronically. Very, very brief pauses occur between the *shorts* and *longs* of a letter. Longer pauses indicate the separation between letters. This basically means that electronically we can send human messages by turning electricity on and off in a series of organized pulses. Does this mean that Samuel Morse invented the computer? No, he did not get credit for starting the computer revolution, but it does serve as a simple example to illustrate how electricity can process letters by translating **on** and **off** situations into letters and numbers.

1.6 Storing Data Electronically with 1s and 0s

Fine, Morse code explains how letters can be translated into electronic impulses. This explains electronic communication, but Morse code does not store any letters. Morse code signals are sent and they are gone, followed by the next signal. If you doze off, you miss the signal and it is too bad. Luckily, somebody became clever and a special device was invented that printed **dots** (short signals) and **dashes** (long signals) on a paper tape as the message was received. Now that explains a paper memory, but we still have not achieved electronic memory.

The earliest electronic computers from 1939 on into the 1940s achieved electronic memory by using *Vacuum Tubes*. Vacuum tubes are about the size of a normal light bulb – and like light bulbs, vacuum tubes can turn on and off. When a tube is "on" it is storing a **1**. When it is "off" it is storing a **0**. While vacuum tubes allowed the electronic computers function, they still had their share of drawbacks.

First, vacuum tubes are big and bulky. As I said, they are about the size of a light bulb. With 8 of these vacuum tubes the computer could process a single character. In order to process anything of consequence a computer would need thousands and thousands of vacuum tubes. This is why the early computers were so massive back then. For example, the first electronic general purpose computer, the ENIAC, had over 17,000 vacuum tubes. Imagine how big a machine would need to be if it had about 17,000 normal size light bulbs. It would be the size of a gymnasium!

There is something else that vacuum tubes have in common with light bulbs. They generate heat and eventually *burn out*. If a vacuum tube burns out, the computer stops working until the vacuum tube is replaced. The heat is a another issue. A single light bulb can get pretty hot after a while. Imagine the heat produced by 17,000 light bulbs.

During this time period, workers often complained about unsafe working conditions due to the intense heat. Keep in mind that did not have the same "labor laws" back then that we do today.

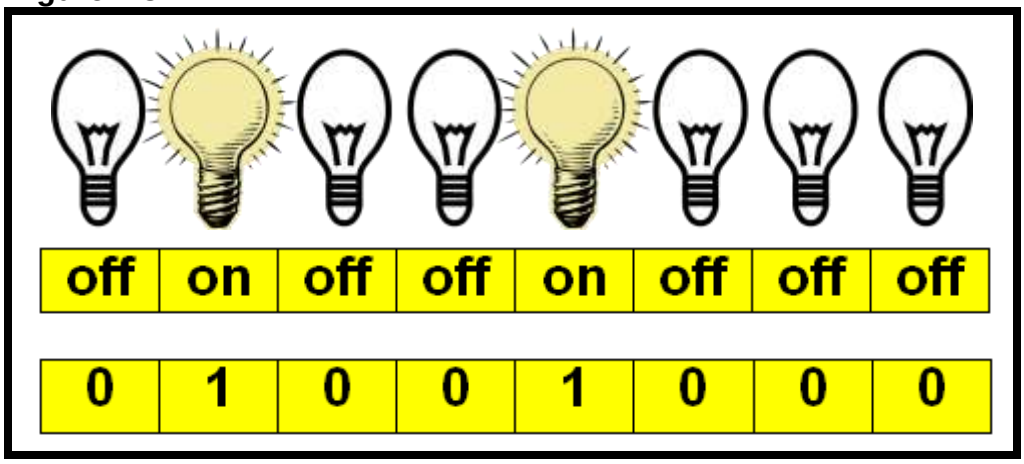
A Brief History of Character Encoding

At its most fundamental level, all digital information comes down to a series of 1s and 0s. We call these binary digits *bits*. When you have a group of 8 bits, it is called a *byte*. In the early days, life was simple. We had a 7-bit encoding system called ASCII (***A**merican **S**tandard **C**ode for **I**nformation **I**nterchange*). Using 7-bits allowed for 2^7 or 128 different characters which could easily be stored in 1 byte. This was enough to store all of the standard or "typeable" characters, which include the CAPTIAL letters, lowercase letters, numerical digits, punctuation marks, and the other characters that you see above the numbers on your keyword.

ASCII was fine, until we wanted to have digital conversations with countries like China, Japan and Korea. This gave birth to Unicode, which initially used 16 bits allowing for 2^{16} or 65,536 different characters that were stored in 2 bytes. It was thought that this would be enough for all of the characters from every language. Actually, it wasn't, and when emojis came along things became even more complicated. As of May 2019, the Unicode Character Set has expanded to include almost 138,000 different characters. While there are different ways to encode these, the most popular is UTF-8 (*Unicode Transformation Format*). This is a flexible coding system that can use as few as 8 bits – allowing it to be backwardly compatible with systems that still use ASCII – or as many as 32. In other words, UTF-8, uses between 1 and 4 bytes. This allows it to handle as many as 1,112,064 different characters which should be plenty, at least for a while. There is also a UTF-16 which uses between 2 and 4 bytes. This is the format used by the language Java. There is also a UTF-32 which always uses 4 bytes.

OK, now suppose you line up a series of 8 vacuum tubes. They might be difficult to find in the 21st century so 8 light bulbs will work just as well. Each light bulb is capable of being turned **on** and **off**. With these **8** light bulbs we can create 2^8 or 256 different combinations. Two tables are shown in Figure 1.3 below. The first diagram shows **on** and **off**. The second diagram uses **1** and **0**. Remember, in Computer Science, **1** means **on** and **0** means **off**.

Figure 1.3



In this particular example, the *second* and *fifth* bulbs are **on**, and all the other bulbs are **off**. This represents only one of the 256 different combinations. Figure 1.5 will show three more combinations. It certainly is not Morse code, but by using the Morse code example, we can imagine that each of the 256 combinations is assigned to a letter, a number, or some other type of character.

Before we go on, we need to truly understand our own number system. The number system that we use is called the *decimal* number system or *base-10*. It is called "base-10" because it has 10 digits (**0** – **9**). Legend has it that people developed a base-10 system because of our ten fingers. Aside from 10 digits, there is something else that is significant about base-10 numbers. Every digit in a base-10 number represents a multiple of a *power of 10*. Consider the base-10 number **2,345,678** as it is shown in Figure 1.4.

Figure 1.4

10^6	10^5	10^4	10^3	10^2	10^1	10^0
1,000,000	100,000	10,000	1,000	100	10	1
2	**3**	**4**	**5**	**6**	**7**	**8**

Now consider these three "8-light-bulbs" combinations in Figure 1.5. Each of these combinations of **on** and **off** light bulbs can be viewed as a base-2 number. In the same way that every digit in a base-10 number represents a multiple of a *power of 10*, every column in a base-2 number represents a *power of 2*. The math is identical. The only thing that changed is the *base*.

Figure 1.5

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	**1**	**0**	**0**	**0**	**0**	**0**	**1**

01000001 (base-2) = **65** (base 10)

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	**1**	**0**	**0**	**0**	**0**	**1**	**0**

01000010 (base-2) = **66** (base 10)

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	**1**	**0**	**0**	**0**	**0**	**1**	**1**

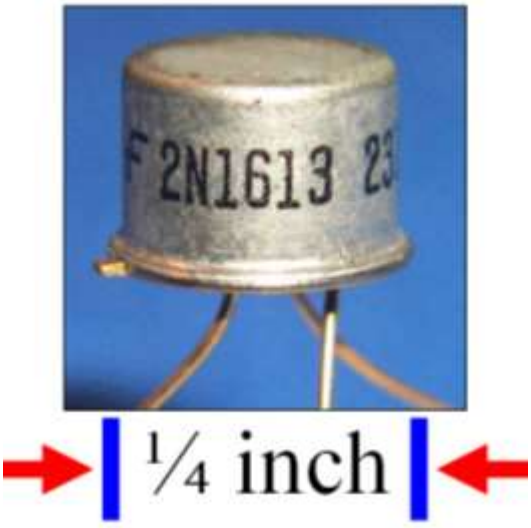
01000011 (base-2) = **67** (base 10)

In Figure 1.5, you are looking at the 8-bit representation of capital letters **A**, **B**, and **C** on the majority of today's personal computers. By convention, at least the convention of the ***A**merican **S**tandard **C**ode for **I**nformation **I**nterchange* (ASCII**)**, number **65** is used to store capital letter **A**. In the same way, the number **66** is used to store capital letter **B** and **67** is used to store capital letter **C**.

Have you noticed that whenever you enter a password, it usually matters whether you enter the letters as capital letters or lowercase case letters? We often say that passwords are *case-sensitive*. Why would it matter if you enter a lowercase **a** instead of a capital **A**? Remember, to the computer, it is all just 1s and 0s. You have already seen the 8-bit combinations for capital letters **A**, **B**, and **C**. What needs to be understood is that lowercase letters **a**, **b**, and **c** have different 8-bit combinations from their capital counterparts. While the ASCII code for capital **A** is **65**, the ASCII code for lowercase **a** is actually **97**.

1.7 Memory and Secondary Storage

In the history of Computer Science, one of the most significant inventions was that of the transistor. This changed computers forever. Transistors have certain key advantages over vacuum tubes. First, they are much smaller. This allowed computers to become much smaller and considerably less expensive. Second, transistors do not get hot and do not burn out like vacuum tubes. This means we no longer have to deal with the issues of intense heat and replacing burned out vacuum tubes. Eventually, this led to the invention of *integrated circuits* and later *microchips*. Today's microchips actually hold billions of transistors.



Microchips are made out of silicon. Silicon is a semiconductor, which allows precise control of the flow of electrons. In today's computers the main board with all the primary computer components, including several microchips, is called the *motherboard*.

There are actually 3 different types of microchip stored on the motherboard. One of these stores permanent information for the computer. Such chips are called ***R**ead **O**nly **M**emory* chips or *ROM* chips. There is a quite a bit of information in the computer that should not disappear when the power is turned off, and this information should also not be altered if the computer programmer makes some mistake. A ROM chip can be compared to a music CD. You can listen to the music on the CD, but you cannot alter or erase any of the recordings.

Another type of chip stores information temporarily. When the power is gone, so is the information in these chips. Computer users also can alter the information of these chips when they use the computer. Such chips can store the data produced by using the computer, such as a research paper or it can store the current application being used by the computer. The name of this chip is ***R**andom **A**ccess **M**emory* chip or *RAM* chip. Personally, I am not happy with that name. I would have preferred something that implies that the chip is **R**ead and **W**rite, but then nobody asked for my opinion when memory chips were named.

The most significant chunk of silicon in your computer is the *CPU* chip. *CPU* stands for ***C**entral **P**rocessing **U**nit* and this chip is the *brains* of the computer. It handles all processing and calculations.

Computer terminology has actually borrowed terms from the *Metric System*. We all remember that a *kilometer* is 1000 meters and a *kilogram* is 1000 grams. This is because the Metric System prefix *kilo* means *1000*. In the same way, a *kilobyte* is about 1000 bytes. Why did I say "about"? Remember that everything in the computer is based on powers of 2. If you are going to be really technical and picky, a *kilobyte* is exactly 2^{10} or 1024 bytes. For our purposes, 1000 bytes is close enough. Other metric system prefixes are shown in Figure 1.6.

Figure 1.6

Measuring Memory			
KB	Kilo Byte	1 thousand bytes	1,000
MB	Mega Byte	1 million bytes	1,000,000
GB	Giga Byte	1 billion bytes	1,000,000,000
TB	Tera Byte	1 trillion bytes	1,000,000,000,000
PB	Peta Byte	1 quadrillion bytes	1,000,000,000,000,000
EB	Exa Byte	1 quintillion bytes	1,000,000,000,000,000,000
ZB	Zetta Byte	1 hexillion bytes	1,000,000,000,000,000,000,000
YB	Yotta Byte	1 septillion bytes	1,000,000,000,000,000,000,000,000

Modern computers now have memory is measured in *gigabytes* and hard drive space is measured in *terabytes*. Kilobytes and megabytes are rapidly fading from the computer terminology. Your children will probably be working with *petabytes* and *exabytes*. Your grandchildren will probably be working with *zettabytes* and *yottabytes*.

Secondary Storage

I just know that you are an alert student. ROM made good sense. RAM also made sense, but you are concerned. If the information in RAM is toast when you turn off the computer… then what happens to all the stored information, like your research paper? Oh, I underestimated your computer knowledge. You do know that we have hard drives, diskettes, zip diskettes, tapes, CDs, DVDs and USB jump drives that can store information permanently.

We have stored information with *rust* for quite some time. Did I say *rust*? Yes, I did. Perhaps you feel more comfortable with the term *iron oxide*. Tiny particles of iron oxide on the surface of a tape or floppy disk are magnetically charged positively or negatively. Saving information for later use may be a completely different process from simply storing it in memory, but the logic is still similar.

Please do keep in mind that this information will not disappear when the power is turned off, but it can be easily altered. New information can be stored over the previous information. A magnetic field of some type, like a library security gate, heat in a car, dust in a closet, and peanut butter in a lunch bag can do serious damage to your information.

The on/off state is the driving force of the digital computer. What is *digital*? Look at your watch. You can see digits, and you see the precise time. There is no fractional time. A clock with hour, minute and second hands is an *analog* device. It measures in a continuous fashion. A measuring tape is also *analog*, as is a speedometer with a rotating needle. What is the beauty of digitizing something? With digital information it is possible to always make a precise copy of the original.

It is easy to transfer, store and use digitized information. Entire pictures can be converted to a digitized file and used elsewhere. I am sure you have been in movie theaters where "digital" sound is advertised. So digital is the name of the game. Just remember that not all digitizing is equally fast. The internal memory of the computer is digital and it uses electronics. The access of a hard disk involves electronics, but the information is read off a disk that rotates and only one small part of the disk is "readable" at one time. Accessing a disk drive is much slower than accessing internal memory.

1.8 Hardware and Software

Computer science, like all technical fields, has a huge library of technical terms and acronyms. Volumes can be filled with all kinds of technical vocabulary. Have no fear; you will not be exposed to volumes, but you do need some exposure to the more common terms you will encounter in the computer world. Some of these terms will be used in the following section on the history of computers.

For starters, it is important that you understand the difference between *hardware* and *software*. Computer *hardware* refers to any physical piece of computer equipment that can be seen or touched. Essentially, hardware is *tangible*. Computer *software*, on the other hand, is *intangible*. Software refers to the set of computer instructions which make the computer perform a specific task. These computer instructions, or *programs*, are usually encoded on some storage device like a CD, jump drive or hard drive. While CDs, jump drives and hard drives are examples of tangible hardware, the programs stored on them are examples of intangible software.

Computer Hardware and Peripheral Devices

There are big, visible hardware items that most students know because such items are difficult to miss. This type of hardware includes the main computer box, the monitor, printer, and scanner. There are additional hardware items that are not quite as easy to detect.

It helps to start at the most essential computer components. There is the CPU (Central Processing Unit), which controls the computer operations. The CPU together with the primary memory storage represents the actual computer. Frequently, when people say to move the CPU to some desk, they mean the big box that contains the CPU and computer memory. This "box" is actually a piece of hardware called the *system unit* and it actually contains a lot more than just a bunch of memory chips. There are also many *peripheral devices*.

What does periphery mean? It means an imprecise boundary. If the computers are located on the periphery of the classroom, then the computers are located against the walls of the classroom. Computer hardware falls into two categories. There are *internal* peripheral devices and *external* peripheral devices.

External peripheral devices are located outside the computer and connected with some interface, which is usually a cable, but it can also be wireless. The first external peripheral device you see is the monitor. Other external peripheral devices include a printer, keyboard, mouse, scanner, and jump drive. There are

many *internal* peripheral devices that are connected to the computer inside the system unit. These devices include the disk drive, CD ROM drive, hard drive, network interface card and video card.

Computer Software

Computer software provides instructions to a computer. The most important aspect of this course is to learn how to give correct and logical instructions to a computer with the help of a programming language. Software falls into two categories. There is *system software* and *application software*. Usually, students entering high school are already familiar with applications software.

Applications software refers to the instructions that the computer requires to do something specific for you. The whole reason why a computer exists is so that it can assist people in some type of *application*. If you need to write a paper, you load a *word processor*. If you need to find the totals and averages of several rows and columns of numbers, you load an *electronic spreadsheet*. If you want to draw a picture, you load a *paint program*. Word processors and electronic spreadsheets are the two most common *applications* for a computer. Currently, there are thousands of other applications available which assist people in every possible area from completing tax returns to designing a dream home to playing video games.

NOTE: People often talk about the "apps" on their cell phone.
App is just an abbreviation for *application software*.

System software refers to the instructions that the computer requires to operate properly. A common term is ***O****perating* ***S****ystem* (OS). The major operating systems are Windows, UNIX, Linux and the MAC OS. It is important that you understand the *operation* of your operating system. With an OS you can store, move and organize data. You can install new external devices like printers and scanners. You can personalize your computer with a desktop appearance and color selections. You can execute applications. You can install additional applications. You can also install computer protection against losing data and viruses.

1.9 What Is Programming?

Computer Science is a highly complex field with many different branches of specialties. Traditionally, the introductory courses in computer science focus on programming. So, what is programming? Let us start by straightening out some programming misconceptions. Frequently, I have heard the phrase: *just a second sir, let me finish programming the computer*. I decide to be quiet and not play teacher. The person "programming" the computer is using some type of data processing software. In offices everywhere, clerks are using computers for a wide variety of data processing needs. Now these clerks enter data, retrieve data, rearrange data, and sometimes do some very complex computer operations. However, in most cases they are not *programming* the computer. Touching a computer keyboard is not necessarily programming.

Think about the word *program*. At a concert, you are given a program. This concert program lists a sequence of performances. A university catalog includes a *program of studies*, which is a sequence of courses required for different college majors. You may hear the expression, *let us stick with our program*, which implies that people should stick to their agreed upon sequence of actions.

In every case, there seem to be two words said or implied: *sequence* and *actions*. There exist many programs all around us and in many cases the word program or programming is not used. A *recipe* is a program to cook something. A well-organized recipe will give precise quantities of ingredients, along with a sequence of instructions on how to use these ingredients.

Any parent who has ever purchased a *some-assembly-required* toy has had to wrestle with a sequence of instructions required to make the toy functional. So we should be able to summarize all this programming stuff, apply it to computers and place it in the definition diagram below.

Program Definition

A *program* is a sequence of instructions, which enables a computer to perform a desired task.

A *programmer* is a person who writes a program for a computer.

Think of programming as communicating with somebody who has a very limited set of vocabulary. Also think that this person cannot handle any word that is mispronounced or misspelled. Furthermore, any attempt to include a new word, not in the known vocabulary, will fail. Your communication buddy cannot determine the meaning of a new word by the context of a sentence. Finally, it is not possible to use any type of sentence that has a special meaning, slang or otherwise. In other words, *kicking the bucket* means that some bucket somewhere receives a kick.

A very important point is made here. Students often think very logically, write a fine program, and only make some small error. Frequently, such students, it might be you or your friends, become frustrated and assume some lack of ability. It is far easier to accept that small errors will be made, and that the computer can only function with totally clear, unambiguous instructions. It is your job to learn this special type of communication.

Early Programming in Machine

Back in the 1940s, computers like the ENIAC were incredibly difficult to program. Programming the ENIAC required rewiring the machine, and machines like the ENIAC had thousands of wires. Around the same time there was device called the Mark-I. While this was technically a *calculator*, it required the manipulation of its 1,440 switches. Just entering a program into those early computers was hard enough, but what if the program did not work? On the ENIAC, you would be looking as a sea of thousands of wires, and you need to find out which is plugged into the wrong port. On the Mark-I, maybe you flipped switch #721 *up*, but it should actually be *down*. Seeing that amidst the other 1,439 switches is not easy. This is programming in "Machine Language" or "Machine Code" where you are directly manipulating the 1s and 0s of the computer's binary language.

"Amazing Grace"

Grace Hopper was one of the first programmers of the Mark-I. She is also credited with making the term *debugging* popular after a couple colleges pulled the first literal computer bug (a moth) out of the Mark-II. However, her biggest accomplishments came in the area of computer languages.

In the 1940s, Grace Hopper did not like the way we were programming computers. The issue was not that it was difficult. This issue was that it was tedious. She knew there had to be a better way. The whole reason computers were invented in the first place was to do the repetitive, tedious tasks that human

beings do not want to do. It should be possible to program a computer using English words that make sense, rather than using just 1s and 0s.

Imagine the presidents of the United States and Russia want to have a meeting. The American President speaks English. The Russian President speaks Russian. Who else needs to be at this meeting? They need a *translator*. Grace Hopper understood this. If she wanted to program a computer with English words, she would need some type of *translator* to translate these words into the machine language of the computer.

Grace Hopper wrote the first *compiler* (a type of translator) in 1952 for the language *A-0*. The language itself was not successful because people either did not understand or did not believe what a compiler could do. Even so, A-0 paved the way for several other computer languages that followed. Many of which were also created in part or in whole by Grace Hopper.

Her immeasurable contributions to computer science have earned her the nickname "Amazing Grace". The Cray XE6 *Hopper* supercomputer and the USS *Hopper* Navy destroyer are also named after her.

High-Level Languages and Low-Level Languages

Languages like Machine Language are considered *Low-level Languages* because they function at, or very close to, the level of 1s and 0s. In contrast, a *High-level Language* is a language that uses English words as instructions. BASIC, Pascal, FORTRAN, COBOL, Java and Python are all examples of high-level languages. Do realize that this is not the same English that you would use to speak to your friends. A high-level language consists of a set of specific commands. Each of these commands is very exact in is meaning and purpose. Even so, programming in a high-level language is considerably easier than programming in a low-level language.

In terms of *high-level* and *low-level* there are a couple more classifications of programming languages. One is *very-high-level*. If *low-level* is programming with 1s and 0s and *high-level* is programming with English words, then what is "very-high-level"? A *very-high-level* language is one where you program with pictures. This style of programming is used to introduce programming concepts at some elementary or middle schools.

The other term you may hear is *medium-level language*. So how does this work? While low-level languages are very difficult, they are also very powerful and give you complete control over the computer. This is similar to the way a manual transmission is more difficult to drive than an automatic, but it gives you more control over the car. High-level languages are easier, but they have less power. Very-high-level languages have even less power. However, there are a few languages that use English words, like a high-level language, but give you total control over the computer, like a low-level language. It should make sense that these hybrid or "medium-level" languages are very popular with professional programmers.

Compilers and Interpreters

Grace Hopper invented the first *compiler*, which is one of two types of translators. The other is an *interpreter*. An interpreter functions by going through your program line by line. Each line is individually translated into machine code (1s and 0s) and then is executed.

A compiler is more efficient. A compiler first goes through and translates the entire program into machine code. This creates a second file which is executable. On modern Windows computers, this would be a **.exe** file. After the translation is complete, you can execute the executable file as many times as you wish. So, for the reasons of speed and efficiency, most modern languages today use a compiler.

Computer Translators

A *translator* (compiler or interpreter) translates a high-level language into low-level machine code.

A *compiler* translates the entire program into an executable file before execution.

An *interpreter* translates one program statement at a time during execution.

Once compilers and interpreters were invented, several languages followed. Most of these languages were created for a specific purpose or with a specific group of people in mind:

Year	Language	Purpose / Significance
1952	A0	Grace Hopper's first compiled language.
1957	FORTRAN	First successful programming language. Designed for mathematicians, scientists and engineers for number crunching.
1959	COBOL	Created (largely by Grace Hopper) to handle record processing to make it ideal for the business community. Adopted by the Department of Defense.
1964	PL/1	Attempted to combine the math features of FORTRAN with the business features of COBOL into one ultimate language that would be "Everything for Everyone". In reality, the language was too complicated and just flopped.
1964	BASIC	Designed as a simple, easy-to-learn language to give non-math and non-science majors the ability to use computers.
1967	Logo	A visual language designed for to introduce young children to programming. Introduced the concept of *Turtle Graphics*.
1969	Pascal	Designed by Niklaus Wirth specifically for the purpose of teaching proper programming techniques, as opposed to the *quick-and-dirty* style of programming in BASIC.
1972	C	Designed for writing compilers. Combined the readability of high-level languages with the power of low-level languages, essentially becoming a "medium-level language". This made it very popular with professional programmers.
1983	C++	Added **O***bject* **O***riented* **P***rogramming* (OOP) to the existing language C.
1990	HTML	Created by Tim Berners-Lee along with the first web server and the first web browser. No he did not "invent the Internet," but he did invent the *Word Wide Web*. HTML is the language used by all web browsers. Without HTML, "surfing the net" would not be possible.
1994	Python	A powerful language that was designed to be simple like BASIC, but less "wordy" than Pascal. Python uses an interpreter for simplicity.
1995	Java	The first *Platform Independent* computer language. "Platform Independence" means that a program created on one computer will work and have the exact same output on any other computer. Requires the use of OOP. This is the language taught in AP® Computer Science-A.
2006	Lego NXT	A "very-high-level language" where you program with pictures instead of words. An example of NXT is below:

So what language will you learn in this class? The answer is Python. In addition to being a powerful language that is simple to learn, several universities have officially switched to Python as the language taught to students in their first Computer Science class.

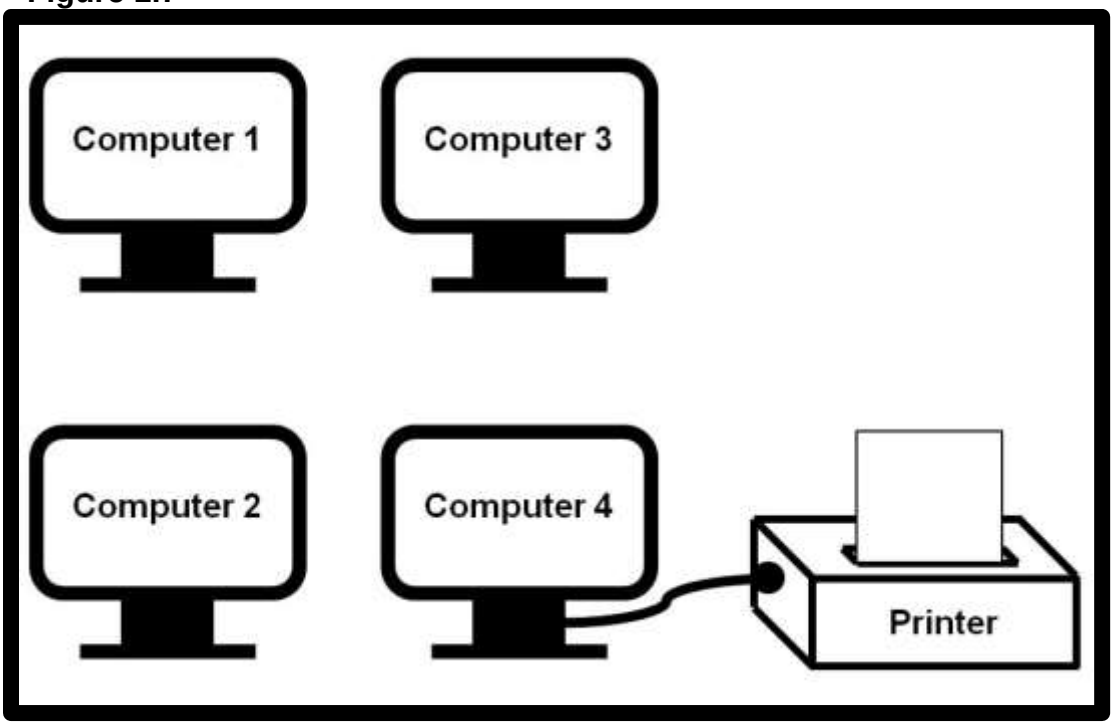
1.10 Networking

When you grow up in a certain environment, it can easily seem natural, as if the environment was always the way you see it. Today's students think it is perfectly normal that computers can send e-mail, play video games and surf the Internet. It was not always that simple. Computers evolved and yes computers do seem to evolve much faster than almost any other type of human creation.

Sneaker Net

Early personal computers were not networked at all. Every computer was a stand-alone computer. Some computers were hooked up to printers and many others were not. If you needed to print something, and you were not directly connected to a printer, you stored your document on a floppy diskette, walked to a computer with an attached printer, and then printed the document. If a group of computer programmers worked together on a project, they needed to get up frequently with stored information to share it with other members in the team. Running around to share computer information is now called the *Sneaker Net* because sharing files or printing files requires you to *put on your sneakers* and walk to another computer. It may not be very clever, but it does illustrate the environment.

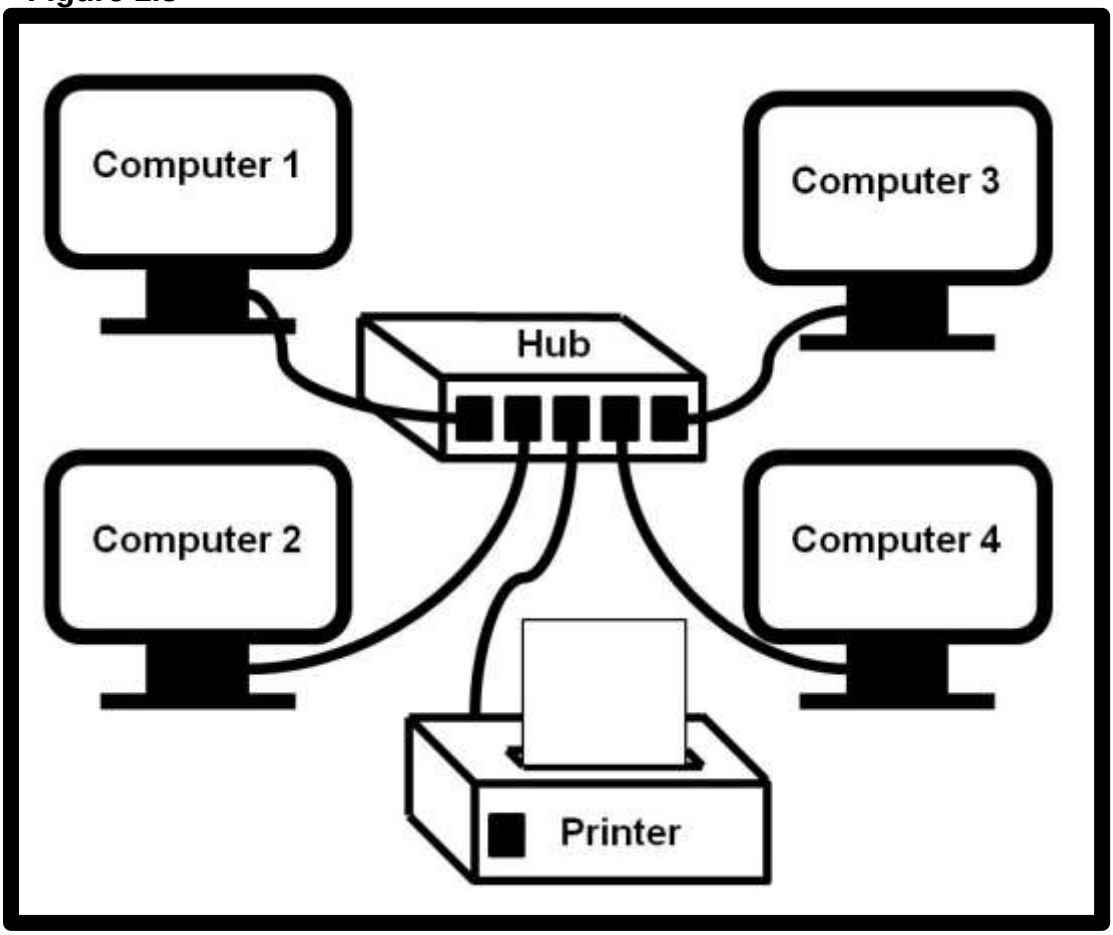
Figure 1.7



Peer-to-Peer Networks

Computers did not wake up one day and hooked up to the Internet. The first practical networks for personal computers were peer-to-peer networks. A peer-to-peer network is a small group of computers with a common purpose all connected to each other. This small network allowed a single printer to be used by any computer on the network and computers could also share information. These types of networks were frequently called ***L**ocal **A**rea **N**etworks* or LANs. Initially, the networks were true *peer-to-peer* networks. This means that every computer on the network was equal. All computers were personal computer work stations.

Figure 1.8

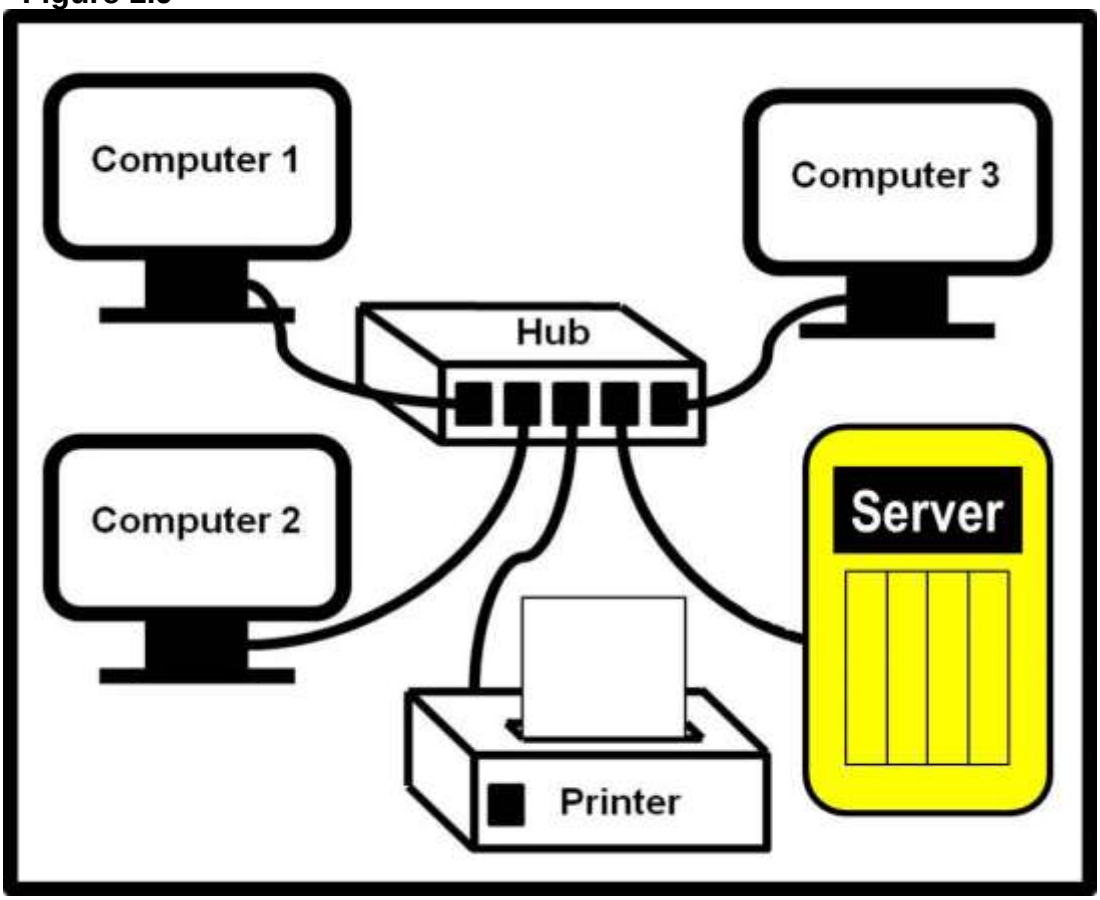


Client-Server Networks

Peer-to-peer networks do not work well when networks get large. Special, dedicated computers, called servers, were needed. A *server* is a specialty computer that is connected to the LAN for one or more purposes. A server provides "services" to the other computer in the network which are called the *clients*. Servers can be used for printing, login authentication, permanent data

storage, web site management and communication. Many businesses would have multiple servers set up in such a manner that some servers exist for the purpose of backing up the primary server. Using backup systems, tape storage or other backup means insured computer reliability in case of computer failure.

Figure 1.9



The Department Of Defense Networking Role

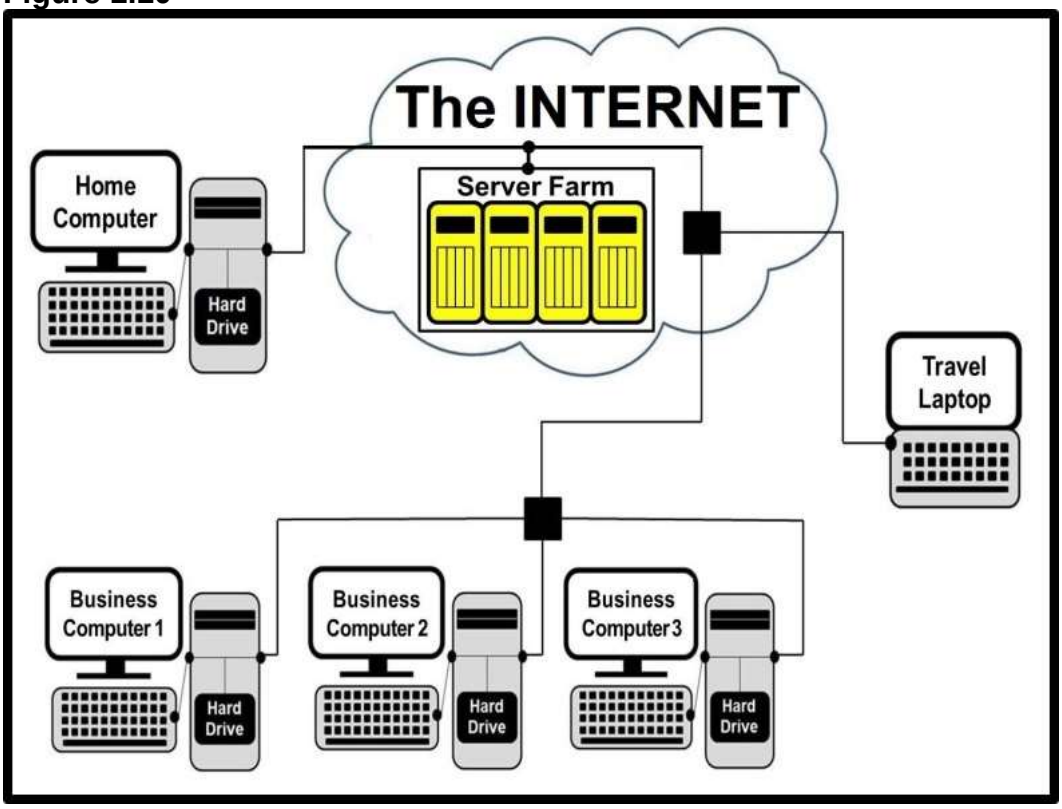
It may come as a shock to you, but the Internet was not created so that teenagers could play video games and download music. The Internet has its origins in the "Cold War." If you do not know what the "Cold War" is ask your Social Studies teacher, your parents or any old person over 40. During the Cold War there was a major concern about the country being paralyzed by a direct nuclear hit on the Pentagon. It used to be that all military communications traveled through the Pentagon. If some enemy force could knock out the Pentagon, the rest of the military establishment communication would be lost. A means of communication had to be created that was capable to keep working regardless of damage created anywhere. This was the birth of the Internet. The Internet has no central location where all the control computers are located. Any part of the Internet can be damaged and all information will then travel around the damaged area.

The Modern Internet

Students often think that the Internet is free. Well that would be lovely, but billions of dollars are invested in networking equipment that needs to be paid in some fashion. Computers all over the world are first connected to computers within their own business or school. Normally, businesses and schools have a series of LANs that all connect into a large network called an *Intranet*. An *Intranet* behaves like the Internet on a local business level. This promotes security, speed and saves money.

Now the moment you, your school, your business wants to connect to the outside world, to the giant world-wide network known as the Internet, you need access to millions of lines of telecommunications. This will cost money and every person, every school, and every business who wants this access needs to use an ***I**nternet **S**ervice **P**rovider* or ISP. You pay a monthly fee to the ISP for the Internet connection. The amount of money you pay depends on the amount of traffic that flows through your connection and the speed of your Internet connection.

Figure 1.10



These days, people are in love with the Internet and now especially they are in love with the *Cloud*. It is a little funny, because the *Cloud* is just a metaphor for the Internet. It came about because whenever people represent "The Internet" in a network diagram, a cloud symbol is used as seen in Figure 1.10.

Today many computers use a wireless connection to hook up to some local network, that in turn hooks up to the Internet. Wireless connections are convenient, but there are some problems. Signals are not always reliable, just like cell phones. You may be stuck in an area somewhere where the signal is weak. Furthermore, there is the security issue. Information that travels wireless is much easier to pick up by hackers than information that is channeled through cable.