Chapter I

Introduction to Computer Science

Chapter I Topics

1.1 Teaching 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 twenty-five 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 40 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 we. This is not for the sake of glorifying myself, but 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, a son-in-law, David, and six 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-Magnum 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 off 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 too 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 Java* just like my previous text book that was called *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 Java after C++, you can learn this material. You can learn it well, even 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 Roller Coaster Emotions

Nobody, in my experience as a computer science teacher, expressed the *roller coaster emotions* of computer science learning better than Howard Smith. Howard is not his real name, but the young man may not like to see his name in print. A typical encounter in the computer lab with Howard went something like this. "Mr. Schram!! Mr. Schram!! I love computers! I am going to be a computer scientist! Computers are so cool!" This excitement would normally follow some segment of Howard's assignment that worked correctly. Unfortunately, Howard was soon faced with an uncooperative computer in the next stage. Frequently, there were problems, and Howard could be seen in the lab jumping up and down as he exclaimed, "Mr. Schram, I hate computers! I don't want to touch another computer. Computers hate me. Mr. Schram, I want to drop this course. I want out! I want out now, this minute! I hate, hate, hate computers!"

Few people show emotions to the degree that Howard did, but he did illustrate a very important point. There is nothing steady about learning computer science. Your time in the computer lab is primarily devoted to making the computer perform some assigned task correctly. You will experience the excitement when the computer responds correctly and you will also experience the frustration when your computer just does not cooperate.

This type of emotion is so different from studying a history book or writing an essay. Essays do not go through stages of working and not working. An essay slowly and steadily becomes better and even when your essay is far from perfect, you may decide to turn it in anyway. A computer program can have some small glitch and will not work at all. Is this so bad? It is not bad at all and I have seen thousands of students learn computer science very successfully. I am mentioning the roller coaster feelings because I have observed it and it may be of help for you to realize that such feelings are very common. After you gain confidence, you may find that learning computer science is very rewarding and can be quite a bit of fun.

1.4 Tackling Computer Science

Students have all kinds of different learning and study approaches. You must accept one simple fact. Learning computer science, like learning most lab sciences, is most effective when you discover an important concept. Teachers can give you theorems, general laws and all kinds of facts about some scientific discipline. Yet, these laws were all discovered by scientists as they observed a wide variety of different situations.

The same philosophy will be used in the *Exposure* books. You will see many, many small programs. Most programs will focus on one or two concepts. Study each one of the program examples carefully. Check to see what the examples have in common.

Remember that reading and studying program examples have limited results. You need to get your hands dirty. Type in the program examples! Observe how the programs work. Change some of the program statements by adding, deleting and altering some of the information. The more you "play" with the program statements, the better your understanding will grow.

Steps In Learning Computer Science

- Read the assigned textbook pages before the lecture.
- Closely listen to the lecture. In particular, focus on any concepts that were confusing during your reading.
- If program examples are used at your computer, make sure that you are at the same place as the lecture.
- Play with the computer. Load or type small program examples and try different approaches.
- Ask questions, if you are confused. Concentrate your questions on the topic being discussed.
- Read the material again after the lecture for continuing exposure. Make note of any topics that are still confusing.
- Read the lab assignments before arriving for a scheduled lab. Think about the solution.
- Work on the lab assignment. Ask for help when you are stuck. Make sure to give clear questions.
- Make sure that you get clarification in any area that is confusing **before** you take a test.
- Pay particularly close attention when a test is returned and your teacher goes over the questions.
- Please realize that cramming does not work with computer science. You need the combination of text book reading, lecture/discussion, and hands-on lab assignments to discover patterns, learn topics and comprehend the bigger picture.

1.5 Excessive Help

A person who does an entire lab assignment without any help from anybody will not be accused of cheating on an assignment. Another person, whose effort on a lab assignment consists of copying somebody else's file does not gain any knowledge from such an approach. Blindly copying somebody's work and calling it your own is academic cheating. Few people argue with the examples stated above. The problem occurs in the middle. What exactly is right, what is wrong, and what might be called *academic dishonesty*?

Computer science students learn not only from their teachers, they also can learn a great deal from fellow students. Sometimes, the best teacher for you may be the person who has just discovered a solution to the problem that is also frustrating you. The name of the game is **learning**, and learning comes in many different forms with many different types of teachers. One day understanding clicks in your brain when you try something on the computer ... another day it may be a teacher's statement in a lecture ... perhaps you read a chapter in your text book again ... and certainly it can be the help of a fellow student.

Program assignments are a considerable part of your grade. Teacher policies fluctuate tremendously but your ability to finish computer lab assignments is an integral part of your total grade. You need to recognize that an easy temptation presents itself. How about simply copying the lab assignment from a good buddy or somebody who is about to become your good buddy? You know some nice kid who appears to be pretty smart and always finishes the assignment before you do.

There are two major problems with this approach:

- It is wrong because it is unethical.
- It is wrong because you are not learning.

If you do not realize that copying other people's work - with or without their consent - is wrong, then it is time you discuss this issue with your parents to get clarification on basic issues of right and wrong.

Computer lab assignments, that can be copied, are only a part of the total grade picture. You also have to take written tests and you have to take computer lab tests. If students study together and help each other with lab assignments, and if the students learn from their study group, they will learn and do fine on tests. Students who put in zero effort do not learn and fail tests miserably.

Those of you who are inclined to help your friends by letting them copy your efforts need to consider the type of friend that you are. A true friend will teach and prepare a buddy for upcoming quizzes and tests. Somebody who likes the quick popularity that comes with improper diskcopying allows a "friend" to fail in the future.

Excessive Help

Do not get excessive help. Do not copy lab assignments. It is wrong. You know it is wrong, and you will not learn.

Do form study groups. Do work together and help each other understand difficult concepts. Do encourage your friends when they are struggling.

1.6 Computer Fundamentals

Getting started with computer science is none too easy. The course that you are taking, and the book that you are reading, assumes 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 programming.

1.7 A Brief Computer History?

It is easy to write a very fat book strictly on the topic of computer history. It is also a major snooze topic for most teenagers. Many young people enjoy working with computers, but listening to a stimulating lecture on the history of computers is another story. In many cases incoming high school students have learned computer history in middle school technology classes. It does seem odd to plunge into a computer science course without at least some reference to *where did this technology come from anyway?* What follows is a chronology of some of the major steps in the history of computers. There are many, many significant large and small contributions that are not mentioned. This at least is a small start.

The Abacus, 3000 B.C.

The Abacus was originally invented in the Middle Eastern area. This rather amazing computing device is still very much used in many Asian countries today. Skilled Abacus handlers can get basic arithmetic results just about as fast as you might get with a four-function calculator.

Napier Bones, 1617

John Napier used some bones marked with special scales to simplify arithmetic by using addition for multiplication and subtraction for division. It set the stage for the slide rule.

Slide Rule, 1622

William Oughtred created the slide rule. This device allows sophisticated mathematical calculations, which was widely in use until around 1970.

Numerical Calculating Machine, 1642

Blaise Pascal builds the first numerical calculating machine. This device works similar to the old car odometers and could perform addition and subtraction. Multiplication was performed with repeated additions.

Jacquard's Loom, 1805

Joseph Jacquard invents flexible cards that are punched with information in such a manner that it is possible to program how cloth will be weaved. It was one of the first examples of programming.

Analytical Machine, 1833

Charles Babbage invents a machine that can read instructions from a sequence of punched cards. This becomes the first general purpose computing machine. He is considered “The Father of Computers”. In the 1990s many malls had a video games store called *Babbages* which was named after him. You do not see those stores today because they were bought out by *GameStop*.

Programming, 1842

Countess Ada Lovelace was Charles Babbage’s assistant. She designs programs that work for Babbage's analytical machine. Some concepts in today’s modern languages are based on ideas she came up with before electronic computers were ever invented. She is considered “The Mother of Programming”. Today a programming language is named after Ada.

Tabulating Machine, 1884

Herman Hollerith invents a tabulating machine that records statistics for the U.S. Bureau of census. Hollerith starts a tabulating company, which after various name changes eventually becomes *International Business Machines*. Most people simply call it IBM.

Differential Analyzer, 1925

Vannevar Bush, a professor at MIT, builds a large scale computing machine capable of sophisticated mathematical computations.

ABC, 1940

The first electronic digital computer was invented by John Atanasoff and Clifford Berry at Iowa State University. They called it the *Atanasoff Berry Computer* or ABC.

Z3, 1941

Konrad Zuse builds a calculating machine capable of automatic computations in Germany during World War II.

Colossus, 1941-1944

This computer is developed in England in various stages and helps to decrypt the secret code message of German communication during Word War II.

Mark I, 1944

This relay-based computer is developed by Harvard University and IBM. Grace Hopper, then a Navy Lieutenant, becomes the first programmer of the Mark I.

ENIAC, 1946

The ENIAC (Electronic Numerical Integrator and Computer) is the first functionally useful fully-electronic computer. The computer is two stories tall, weighs 80 tons, contains 19,000 vacuum tubes, and is programmed by walking inside the computer.

Mark II, 1947

On September 9, 1947 this computer stopped working. A technician found and removed *moth* from one of its relays. This was the first *computer bug*. The actual moth is currently on display at the San Diego Computer Museum.

UNIVAC I, 1951

The UNIVAC I (UNIVersal Automatic Computer) was the world's first commercially available computer. While the Mark I and the ENIAC were not for sale, any company with enough money could actually purchase a UNIVAC I computer. The computer became famous when it correctly predicted the results of the 1952 presidential election.

IBM 701, 1953

IBM starts to sell the first computer with a stored computer program. This computer uses vacuum tubes and is called a first-generation computer.

FORTRAN, 1954

FORTRAN (FORmula TRANslator) is invented by John Backus at IBM. It is the first commercially successful programming language and is designed for engineers and mathematicians.

Integrated Circuit, 1958

Jack Kilby, of Texas Instruments, at Richardson, Texas, invents the planar transistor, which allows creation of integrated circuits and later micro chips in a very small space. This invention is a turning point between the monstrously huge and expensive computers of the past, and the much smaller and cheaper modern computers of today.

Video Games, 1958/1962

The first video game was called *Tennis for Two*. It was created by William Higinbotham and played on a Brookhaven National Laboratory oscilloscope. Since this game did not use an actual computer monitor, some give credit for the first video game to *SpaceWar* written by Stephen Russell at MIT in 1962.

COBOL, 1959

The business programming language COBOL (COmmon Business Oriented Language) is developed based on the design of Grace Hopper.

IBM 360, 1964

IBM sells the first series of compatible computers. This means that a program created on one computer can be transported and used on another, compatible computer.

BASIC, 1964

Tom Kurtz and John Kemeny of Dartmouth create BASIC (Beginners All-purpose Symbolic Instruction Language). This language will later be the first programming language for personal computers.

Pascal, 1969

Niklaus Wirth, a Swiss professor, creates the programming language Pascal, designed for teaching computer science with proper structured programming to university students. It becomes the first language used for the AP Computer Science Examination.

Altair, 1975

Altair becomes the first personal computer. It is created by Ed Roberts and Bill Yates. The computer costs $397.00 and has storage for 256 bytes.

Apple Computer, 1977

The Apple Computer Company is created and introduces the Apple II Personal Computer. It becomes the first commercially successful personal computer.

Tandy/Commodore, 1977

Commodore and Tandy start selling personal computers. The Commodore and the Apple computer require a television interface to view computer operations. The Tandy RadioShak computer has its own CRT (monitor).

VisiCalc, 1979

Dan Bricklin created VisiCalc, a spreadsheet program, which becomes the first wide spread software to be sold. Dan Bricklin initially lived in a hut somewhere in the mountains of Montana and received an average of $30,000.00 a week for his invention. He did not benefit from the tremendous boom in the spreadsheet market, because his software could not get a patent.

WordStar, 1979

MicroPro releases WordStar, which becomes the most popular word processing software program in the late seventies and eighties.

IBM PC, 1981

IBM introduces the IBM P.C. It is a computer with a monochrome monitor and two floppy drives. Hard drives are not yet available for personal computers. IBM's entry into the personal computer market gives the personal computer a serious image as a true business computer and not some sophisticated electronic game playing machine.

MS-DOS, 1981

IBM decides not to create its own operating system for the personal computing market and decide to out-source development of its operating system for its trivial little personal computer department. Many companies reject IBM proposal. Microsoft, an unknown little company run by Bill Gates, agrees to create the operating system for the IBM Personal Computer and becomes a company larger than IBM.

Portability and Compatibility, 1982

The Compaq Portable is known for two things. It is the first *portable* computer. By today's standards it is nothing like a modern laptop. The 28 pound computer was the size of a small suitcase, and looked very much like one as well. The removable bottom was the keyboard which would reveal a 9 inch monitor and a couple of floppy drives. Compaq is also the first computer to be 100% compatible with an IBM PC.

Macintosh, 1984

Apple starts to sell the Apple Macintosh computer. The mouse technology was already developed earlier by Xerox Corporation and Apple actually introduced this technology with its Lisa computer in 1982. The Lisa computer costs $10,000 and was a commercial failure. The "Mac" was the first commercially successful computer with the mouse/windows technology. A *macintosh* is a type of apple – and the favorite for the person who created the Macintosh – hence its name.

Laptop PCs, 1991

Most PC vendors start to sell laptop PCs. The early laptops all have monochrome monitors and lack the power and memory storage of desktop computers.

Windows 3.1, 1992

Microsoft introduces Windows 3.1, which is its first major windows software seller. It is a Windows type interface, but it is based on the DOS operating system.

Windows 95, 1995, U.S.

Microsoft introduces Windows 95, which now is an operating system similar to the Macintosh computer.

The Last Twelve Years

Since 1995 there has been an explosion in personal computers with greater memory, faster processors, larger hard drives, LCD monitors, laptop computers that rival desk top computers in capability, computer networking with wire and then wireless connections and the use of the Internet in incredibly large numbers. The irony of computer progress is that even the experts have been very wrong in their predictions. In 1946 when the first fully electronic computer, the ENIAC, was created, it was stated that the world would never need more than ten ENIACs. Years later at the time when most personal computers had 640,000 Bytes of memory, Bill Gates announced that personal computers will not need more than 640,000 bytes of memory. This chapter is written on a personal computer with one Giga Byte of memory, which is 1,000,000,000 bytes of memory. You may not know precisely what one byte is, but 1,000,000,000 bytes is a whole lot more memory than 640,000 bytes of memory. Yet, your children will be amazed at the primitive machines you are using right now, because their future cell phones and wrist watches will have more power and memory than the desktop and laptop computers have today. Even the cheapest cell phone on the market today has more power than the $10,000,000 ENIAC computer had back in 1946.

1.8 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

Three Areas Where Computers Beat People

- 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.

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 and 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.

1.9 Messages 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 one 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 other 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, below, a long signal is shown with a bar. A short signal is indicated by a square. Consider the first five letters in the Morse code system, shown in the figure 1.2 table.

Figure 1.2

First Five Letters In Morse Code		
A	■ ▬	short - long
B	▬ ■ ■ ■	long - short - short - short
C	▬ ■ ▬ ■	long - short - long - short
D	▬ ■ ■	long - short - short
E	■	short

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 **BAD** would be signaled as follows:

▬ ■ ■ ■ ■ ▬ ▬ ▬ ■ ■

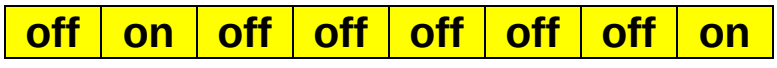
The secret of Morse code is the fact that electricity can be *turned on*, and it can be *turned off*. This means that a flashlight can send long and short beams of light and a buzzer can send long and short buzzing sounds. With an established code, such as Morse code, we can now send combinations of 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.

1.10 Electronic Memory

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, and perhaps you even remember something about punched computer cards, but we still have not gotten to an electronic memory.

Suppose you line up a series of light bulbs. How about picking eight bulbs? Each light bulb is capable of being turned **on** and **off**. With these **eight** light bulbs we can create **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**. In Computer Science, **1** means **on** and **0** means **off**.

Figure 1.3



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

The number system you use is **base-10**. Counting and computation in base 10 is not simpler than other bases because it is base 10. It is simpler because you use base 10. Sounds confusing, does it not? In elementary school, you practiced multiplication tables in base 10. How many multiplication tables did you practice in base 5 or base 8? Not too many, right? Rumor has it that people developed a base 10 system, because of our ten fingers. Now in base 10, digits range from **0** to **9**. After the largest digit **9**, we must use two digits, like **10**, **11**, **12**, **13**, **14** etc. to count higher.

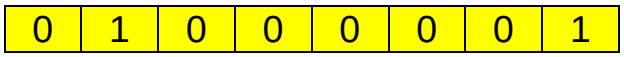
Mathematically speaking, counting and computation are possible in different bases. A number system that is very skimpy in digits is **base-2**. Only the digits **0** and **1** are used. Many digits are needed for even small-valued numbers. The first 32 numbers in **base-2**, with the equivalent **base-10** values are shown in figure 1.4.

Figure 1.4

Base 10	Base-2	Base 10	Base-2
0	0	16	10000
1	1	17	10001
2	10	18	10010
3	11	19	10011
4	100	20	10100
5	101	21	10101
6	110	22	10110
7	111	23	10111
8	1000	24	11000
9	1001	25	11001
10	1010	26	11010
11	1011	27	11011
12	1100	28	11100
13	1101	29	11101
14	1110	30	11110
15	1111	31	11111

Now consider these three "8-light-bulbs" combinations in figure 1.5. Each one of these combinations of **on** and **off** light bulbs can be viewed as a **base-2** number.

Figure 1.5



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

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

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

You are looking at **A**, **B**, **C** on the majority of today's personal computers. By convention, at least the convention of the **A**merican **S**tandard of **C**omputer **I**nformation **I**nterchange (**ASCII)**, number **65** is used to store the letter **A**. Combinations **0** through **127** are used for a standard set of characters. The second group from **128** through **255** is used for a special extended set of characters.

Now we are finally getting somewhere. We can use eight lights for each character that needs to be stored. All we have to do is place thousands of light bulbs in a container and you can store bunches of information by using this special **binary** code. There is another big bonus. Mathematically speaking, computations can be performed in any base. With our clever binary system, we now have a means to store information and make electronic calculations possible as well.

We have now learned that information can be stored in base-2 numbers. Base-2 numbers can store characters by using a system that equates numbers like the base-2 equivalent of **65** to **A**. At the same time, mathematical operations now become an electronic reality. In other words, the magic of on/off switches allows both electronic storing of information as well as electronic computation.

We can also add some terminology here. A single bulb can be **on** or **off** and this single light represents a single digit in **base-2**, called a **Bi**nary digi**t**, which is abbreviated to **Bit**. We also want to give a special name to the row of eight light bulbs (**Bit**s) that make up one character. This row shall be called a **Byte**. Keep in

mind that **Byte** is not plural for **Bit**. There is one problem with storing characters in a single byte. You only have access to 256 different combinations or characters. This may be fine in the United States, but it is very inadequate for the international community. **Unicode** is now becoming very popular and this code stores characters in 2 bytes. The result is **65,536** different possible characters. Java has adopted **Unicode**, as have many technical organizations. The smaller **ASCII** code is a subset of **Unicode**.

Bits, Bytes and Codes

Bit is a **Bi**nary digi**t** that is either **0** (off) or **1** (on).

1 Nibble = 4 bits

1 Byte = 8 bits.

1 Byte has **256** different numerical combinations.

2 Bytes has **65,536** different numerical combinations.

ASCII uses one byte to store one character.

Unicode uses two bytes to store one character.

Early computers did in fact use **one** vacuum tube for each **Bit**. Very large machines contained thousands of vacuum tubes with thousands of switches that could change the status of the tubes. Miles of wires connected different groups of vacuum tubes to organize the instructions that the computer had to follow. Early computer scientists had to walk inside giant computers and physically connect wires to different parts of the computer to create a set of computer instructions.

The incredible advances in computer technology revolve around the size of the bit. In the forties, a bit was a single vacuum tube that burned out very rapidly. Soon large vacuum tubes were replaced by smaller, more reliable, vacuum tubes. A pattern was set that would continue for decades. Small is not only smaller, it is also better. The small tube gave place to the pea-sized transistor, which was replaced by the planar transistor and bits were miniaturized, becoming smaller and smaller. Today, a mind-boggling quantity of bits fits on a microchip.

This is by no means a complete story of the workings of a computer. Very, very thick books exist that detail the precise job of every component of a computer. Computer hardware is a very complex topic that is constantly changing. Pick up a computer magazine, and you will be amazed by the new gadgets and the new

computer terms that keep popping up. The intention of this brief introduction is to help you understand the essence of how a computer works. Everything revolves around the ability to process enormous quantities of binary code, which is capable of holding two different states: **1** and **0**.

1.11 Memory and Secondary Storage

Electronic appliances used to have complex --- cables everywhere --- dusty interiors. Repairing such appliances could be very time consuming. Appliances, computers included, still get dusty on the inside, but all the complex wires and vacuum tubes are gone. You will now see series of boards that all have hundreds and thousands of coppery lines criss-crossing everywhere. If one of these boards is bad, it is pulled out and replaced with an entire new board. What used to be loose all over the place, vacuum tubes, transistors, resistors, capacitors and wires, is now neatly organized on one board. Electronic repair has become much faster and cheaper in the process.

In computers the main board with all the primary computer components is called the **motherboard**. Attached to the motherboard are important components that store and control information. These components are made out of chips of silicon. Silicon is a semiconductor, which allows precise control of the flow of electrons. Hence we have the names memory chip, processing chip, etc. We are primarily concerned with the **RAM** chip, **ROM** chip and the **CPU** chip.

I mentioned earlier that information is stored in a binary code as a sequence of **ones** and **zeroes**. The manner in which this information is stored is not always the same. Suppose now that you create a group of chips and control the **bits** on these chips in such a way that you can not change their values. Every **bit** on the chip is fixed. Such a chip can have a permanent set of instructions encoded on it. These kinds of chips are found in cars, microwaves, cell phones and many electronic appliances that perform a similar task day after day.

Computers also have information chips that store permanent information. Such chips are called **R**ead **O**nly **M**emory chips or **ROM** chips. There is a bunch 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. Once again, information is stored in many **bytes**, each made up of eight **bits**, but this information requires a continuous electric current. 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 amount of memory in a computer is measured in bytes, not bits. Modern computers have so many bytes that usually memory is indicated as so many **kilobytes** or **megabytes** of memory. Kilobytes are rapidly fading from the computer scene and **gigabytes** are alive and well in current computer terminology.

Figure 1.6

Measuring Memory		
KB	Kilo Byte	1 thousand bytes
MB	Mega Byte	1 million bytes
GB	Giga Byte	1 billion bytes
TB	Tera Byte	1 trillion bytes
PB	Peta Byte	1 quadrillion bytes
EB	Exa Byte	1 quintillion bytes

The *measuring memory* diagram, in figure 1.6, may get a frown or two. The information is technically incorrect. The diagram is meant to help remember the measure size in a rounded manner. After all, **Kilo** does mean one-thousand. Technically speaking one **KB** is $\mathbf{2^{10}}$ or **1,024** bytes. Using the same logic you can compute that one **MB** is $\mathbf{2^{20}}$ or **1,048,576** bytes. I am sure you can live very comfortably using the previous "not really correct" diagram.

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 brain of the computer. You cannot call this chip **ROM** or **RAM**. On this tiny little chip are lots of permanent instructions that behave like **ROM**, and there are also many places where

information is stored temporarily in the manner of a **RAM** chip. The **CPU** is one busy little chip. You name it, the CPU does the job.

A long list of operations could follow here but the key notion is that you understand that all the processing, calculating and information passing is controlled by the Central Processing Unit. The power of your computer, the capabilities of your computer, and the speed of your computer is based on your CPU chip more than any other computer component.

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, and CDs that can store information permanently.

We have stored information on **rust** for quite some time. Did I say **rust**? Yes, I did and perhaps you feel more comfortable with **iron oxide**. Tiny particles of iron oxide on the surface of a tape or disk are magnetically charged positively or negatively. In a different manner than the internal computer, but with a similar logic, coded information is stored on a tape or a disk.

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.

You might be confused about the currently popular **CD-ROM**s. You can see that they are external to the computer, but **ROM** implies Read Only Memory. CDs store enormous amount of information. The information is permanent and thus behaves like **ROM**. When you use a CD with a computer it behaves as if you had added extra ROM to your computer internally. CDs do not use rust; they are far too sophisticated for such a crude process. The CD is coded with areas that reflect and absorb laser light. Once again we can create a code system because we have two different states, on and off.

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.

1.12 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.

1.13 Program Languages

You are happy with the fact that a program is a set of instructions to perform a given task. So what needs to be done is understand how instructions are communicated to a computer. At this stage, we have talked quite a bit about binary code, digital information and you have a pretty good hunch that our computers are not using *Star Trek* technology. So do not try to talk to a computer in English and tell it to finish your history homework.

For decades computer scientists have struggled with computers understanding human language. It has not been easy. We have made remarkable progress and a brief history of programming languages will help to understand where we are today, and what you will be doing this school year in computer science.

Programming In Machine Code

Programming the first computers was an unbelievably difficult task. Individual vacuum tubes had to be switched on or off. Instructions were sequenced in early computers by physically plugging wires from one computer memory segment to another. It is true that the early computers were amazingly fast. They were electronic, and they amazed people with their speed and accuracy, but programming those early computers was incredibly time consuming. Very few computers existed in those early days, and large teams of programmers and technicians were necessary for programming even simple jobs. Later, computers did improve the program process by allowing tape and cards to be used for program input. This was far more efficient than walking inside a computer, but there still remained the tedious process of thousands of **1s** and **0s** that had to be entered. Mistakes were very easily made, and very difficult to detect.

Programming in Assembly Language

It did not take long before computer scientists realized that computers are the perfect tool to help people with programming. After all, a program is a set of instructions that can be executed by the computer. This set of instructions, or program, can certainly include instructions that help to translate a more human set of instructions into computer machine code. A special computer language was created called **Assembly Language**.

With Assembly Language, programmers were able to give instructions to the CPU with a short code for every type of task performed by the CPU. Every CPU instruction had a *mnemonic* name. For instance, **`mov AX,1234`** is the

instruction to place the value 1234 into the **AX** register. The **AX** register is a temporary calculation scratch pad in the CPU. Assembly Language brought another improvement by using hexadecimal (base-16) numbers rather than binary (base-2) numbers. There exists a unique relationship between base-2 and base-16 numbers, such that any set of four-digit base-2 numbers can be represented by one base-16 number. It is certainly easier to work with one-fourth the digits.

A programmer first wrote a set of instructions in the special Assembly Language code. This code was then "translated" by a so-called "Assembler." The Assembler translated the mnemonic short word instructions into pure, binary machine code that could be processed by the computer.

Programming with Interpreters and Compilers

Assembly Language started the ball rolling. A computer program had been created to make the job of creating a machine language program simpler. This process could be continued and a more sophisticated program could translate a program that is very similar to human language. One person, who must be mentioned, is the late **Grace Hopper**. Grace Hopper was a lieutenant in the Navy when she first worked extensively with computers. She was largely instrumental for developing translating programs that allow programming in a human-style language. She was one of the main developers of the popular program language, COBOL, and was very influential in the development of early translators.

Two types of translating programs were created, **interpreters** and **compilers**. An interpreter takes a program and translates the program one line at a time. After each line is translated, the resulting machine code is executed. A compiler translates the entire program into a machine code file and then executes the file. It is easier to create an interpreter, but the execution speed is slower than a compiler. A compiler is far more complex to create, but it executes much faster than an interpreter. The majority of today's program languages use compilers for translators. The language you will learn, Java, oddly enough is both a compiled and an interpretive language. How this is possible will be explained soon.

Low-Level and High-Level Program Languages

Languages that are very close to computer binary code are called **low-level**. Machine code and Assembly Language are low-level languages. Languages that are closer to human languages are called **high-level** languages. Some languages are very high-level today, and many programming tasks have already been performed. With many languages, it is possible to click and drag on program objects that have already been created and insert them inside your program.

So why not simply write your programs in English? Is that not a matter of creating some kind of translating program that takes English instructions and creates a machine code file for the computer? This is certainly what has been attempted for many years, but translating human languages has been very elusive. Consider the following example. In the sixties, computer scientists tried to write a program that would translate English into Russian and Russian into English. This is really the same problem of properly understanding the meaning of human language. The scientists decided to test their program by entering an English sentence. The resulting Russian sentence was then entered back into the computer and the final result should be the original English sentence. Should, that is, if the computer could translate correctly in both directions. The original sentence entered was:

The spirit is willing but the flesh is weak

I do not have a clue what the Russian result was, but I do know the final English result. Our computer scientists were quite surprised with

The Vodka is great but the meat is rotten

This little experiment showed the major problem with human languages. Human languages like English are idiomatic. We use all these idioms and proverbs, and slang and special nuances that are meaningless to the computers. How can computers figure out human language, when humans are confused? The bottom line is that programming requires a restricted language. This language can have human type words, but the words selected, the symbols and punctuation used all must have a very precise meaning. The manner in which the program language structures its statements is called syntax. Program languages must have very precise syntax. Compilers first check to see if a program has correct syntax. Only after the syntax checks out, is the next step of translating into binary code performed.

A Brief History of Program Languages

The first successful programming language for the mathematics and scientific community, **FORTRAN** (**FOR**mula **TRAN**slation language), was released in 1956. FORTRAN was an excellent language for mathematics and science, but it could not handle the record processing required for the business world.

In 1960, **COBOL** (**CO**mmon **B**usiness **O**riented **L**anguage) was created (largely by Grace Hopper) for the business community and the armed forces. COBOL became extremely successful when the Department of Defense adopted COBOL as its official programming language.

PL/1 followed, trying to be a language for everybody. PL/1 attempted to be both an excellent language for science and for business. The result was an extremely cumbersome language that never gained much popularity. Then there were a host of many other short-lived languages.

BASIC (**B**eginner **A**ll-purpose **S**ymbolic **I**nstructional **C**ode) was designed for beginning college students. **BASIC** became the first popular program language for personal computers in the seventies. **BASIC** required little memory, and it was the only language that could initially be handled by the first micro computers.

In the late seventies, early eighties, **Pascal** took a strong hold in the educational community. **Pascal** was developed by Niklaus Wirth, specifically for the purpose of teaching proper computer science programming techniques. This language was adopted for the Advanced Placement Computer Science Examination in 1984 and stayed in that position until the 1998 exam.

In the early seventies, the UNIX operating system was developed at the Bell laboratories. This operating system was written in several languages some of which were called **BCPL** or just plain **B**. A later version of the language was called **C**, since it followed **B**. In the eighties, the **C** language became very popular as the language of choice for operating systems.

As the demands for sophisticated computer uses grew, so did the demand for ever more sophisticated computer programming languages. A new era with a powerful programming technique was born called *Object Oriented Programming* (**OOP**). You are hardly in a position to appreciate the finer features of **OOP**. Right now appreciate that a new language had to be developed to incorporate the power of OOP. Bjarne Stroustrup combined the popularity of the existing C language with the demands for OOP and developed C++. C++ includes all the previous capabilities of C, many improvements on the early C, and the new features of object oriented programming. C++ actually is somewhat of a hybrid language since object-oriented Style Programming is possible, but so is the older C-style programming, which today's modern programmers frown upon.

There is something else that you need to understand about this program language stuff. You can say that there is one level above the compiler, called the linker. It did not take computer scientists long to figure out that the majority of programs use the same source code in each and every program. This is the type of code

used with input and output, mathematical calculations, etc. It seemed silly to keep creating this code over and over again. Special library files of handy computer routines were created and already translated into special machine code files. Now these files cannot be executed by the CPU because the instruction set is not complete. Such files are known as object files.

When modern programmers write a compiled program, like C++, they write their own personal source code and the compiler checks their syntax to make sure everything is written correctly. The first pass through the program is the compile pass. Now a second pass links the compiled code with any other library file and combines it into one nifty file that can be used by the computer's CPU.

Java Comes on the Scene

C++ will continue to be a very important programming language for years to come. The industrial strength features and the tremendous number of programs written in C++ guarantee its survival.

In the early Nineties Sun Microsystems worked on a language called **Oak**. The main focus of this new language was to be platform independent and object-oriented. Platform independent means that the language does not cause problems as programs are transported between different hardware and software platforms.

Oak was used internally by Sun Microsystems for about four years and released to the public in 1995 with the new name **Java**. Java was a big success largely because the new language was perfectly suited for the Internet.

Universities also adopted Java very rapidly. By the late Nineties object-oriented programming was taught everywhere and the new language Java gave no choice like C++. This appealed to many college professors who did not like the fact that C++ allowed, older non-OOP programming.

The College Board adopted Java as the AP Computer Science language to be used for its examination starting with the 2003-2004 school year. And you have just started a computer science course that will use Java as its programming language.

1.14 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 use 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.

SneakerNet

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.

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 *Local Area Networks* 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.

Cient-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. Servers can be used for printing, logon authentications, 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.

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 30. 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 a school, a business, your home, wants to be connected to the outside world and giant world-wide network known as the Internet, you have access to millions of lines of telecommunications. This will cost money and every person, every school. Every business who wants this access needs to use an

Internet Service Provider 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.

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.

1.15 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 at the end of this chapter you do need some exposure to the more common terms you will encounter in the computer world. You have already learned about different types of memory, programming languages and networking. There are a few important computer terms that you need to understand.

For starters, it is important that you understand the hardware and software difference. These are the two biggest divisions in the computer business. Hardware involves all the computer components that can be seen, felt, picked up and dropped, like a monitor, a mouse, a jump drive. Software involves the set of computer instructions that are coded on a disk, a CD, or a hard drive.

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. In reality it contains lots 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 were already mentioned. These are hardware, 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. In the old days a monitor was called a CRT (Cathode Ray Tube). This was appropriate with the bulky monitors that looked like old televisions. Today many monitors use LCD (Liquid Crystal Display) screens. It is common now for monitors to be 17, 19, 21 or even 24 inches. (I am looking at a 24# monitor as I edit the 2008 version of this chapter) This has changed since the early computer days of 13 and 15 inch monitors.

Other external peripheral devices include a printer, keyboard, mouse, scanner, jump drive or memory stick.

There are many internal peripheral devices that are connected to the computer inside the computer case. These devices include the disk drive, CD ROM drive, digital camera memory 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 runs an application on a computer. The whole reason why a computer exists is so that it can assist people in some type of application. For centuries, accounting kept large, complicated spreadsheets of business numbers. Now these same numbers are entered on an electronic spreadsheet. You will not

experience the horror of finding some mistakes on the early pages of a thirty page, typed, research paper. This type of mistake necessitates retyping the majority of the pages. Today, this is a simple matter of inserting the corrections on a word processor program. Spread sheets and word processors were the first software applications. Currently, there are thousands of other applications to assist people in every possible area from completing tax returns to playing video games.

System software involves the instructions that the computer requires to operate properly. A common term is OS (operating system). The major operating systems are Windows XP, Window Vista, UNIX and MAC OS. It is important that you understand the operation of your operating system. With 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 install additional applications and computer protection against losing data and viruses.

1.16 Summary

This has been an introductory hodge-podge chapter. It is awkward to jump straight into computer science without any type of introduction. Students arrive at a first computer science course with a wide variety of technology backgrounds. Some students know a little keyboarding and Internet access along with basic word processing skills taught in earlier grades. Other students come to computer science with a sophisticated degree of knowledge that can include a thorough understanding of operating systems and frequently knowledge of one or more program languages as well.

The secret of computer information storage and calculation is the binary system. Information is stored in a computer with combinations of base-2 **ones** and **zeroes**. Individual binary digits (**bits)** store a one or a zero. A one means true and a zero means false. A set of eight bits forms one **byte**.

A byte can store one character in memory with **ASCII**, which allows **256** different characters. The newer, international **Unicode** stores one character in two bytes for a total of **65536** different characters.

Neither this chapter nor this book explained any details about the operating system. Operating systems change frequently, or at least operating system versions change to a new-and-improved model about every two or three years. A solid knowledge of your computer's operating system is vital. Writing a computer program requires knowledge of editing text, saving and loading files and moving around the hard drive's directory system efficiently.

If your basic computer knowledge is weak, make sure to pick up additional information from your teacher, library or bookstore. Technology is evolving and students arrive in computer science classes with increased sophistication. In this course no prior knowledge whatsoever about any programming language or programming logic is assumed. However, a fundamental knowledge of basic computer operations is essential.

Sun Microsystems created Java to be a programming language that is portable on many computer platforms, a so-called *platform-independent* language. They also wanted the language to be compatible with web page development.

Early computers were stand-alone work stations. The first networked computers used a "peer-to-peer" network. This was followed by LANs (Local Area Networks) that connected dedicated specialty servers with computers and printers for a common purpose. The Department of Defense developed the Internet as a means to provide communication at war time.
Individuals, schools and businesses can set up a LAN at their private location without paying a fee beyond the cost of the necessary hardware and software. Connection to the Internet requires an ISP (Internet Service Provider) and a monthly connection fee.

Many computers today, especially laptop computers, have wireless network connections. Such connections are convenient, but they are not as reliable and there is also a greater security risk.

Computers use hardware and software. Hardware peripheral devices are the visible computer components. There are external peripheral devices, such as monitors, keyboards, printers and scanners. There are also internal peripheral devices like disk drives, CD ROM drives, network interface cards and video cards.

Software falls into two categories of application software and operating system software. Application software includes the common applications of word processing and spreadsheets, but also tax return software and video games. Operating system software runs the computer and allows the user to personalize the computer to his or her needs and organize data.