

CPT 168

Programming Logic & Design

★ Work Book ★



 York Technical College

452 S. Anderson Road
Rock Hill, SC 29730
(803)327-8000

Contents

Welcome to CPT 168 – Programming Logic and Design.....	3
Module 1 – Introduction to Problem Solving.....	3
Lesson 1 - Problem Solving and Variables.....	3
Lesson 2 - Operators and Truth Tables	6
Lesson 3 - Functions	11
Lesson 4- Introduction to JavaScript.....	14
Module 2 – Control Structures	20
Lesson 1 - Flowcharts & Pseudocode.....	20
Lesson 2- Decisions Using IF Statements	27
Lesson 3- Nesting Decisions	32
Lesson 4- Loops	36
Lesson 5 - Nesting Loops.....	42
Lesson 6 - Case	46
Lesson 7 – Decision Tables	51
Module 3 – Modular Program Development	53
Lesson 1 - Terminology & Problem Organization	53
Lesson 2 - Scope	62
Lesson 3 – Bullet Proofing.....	65
Lesson 4 – Documentation	69
Lesson 5 – Introduction to Object Oriented Programming/Visual BASIC.....	71
Module 4 – Arrays & File Access	73
Lesson 1- Arrays & Parallel Arrays.....	73
Lesson 2 - Double Scripted Arrays.....	76
Lesson 3- Pointer Technique & Searching Arrays	80
Lesson 4 - File Access	84
Lesson 5 - Introduction to SQL.....	87

Welcome to CPT 168 – Programming Logic and Design.

Computers are wonderful tools, but like all tools the user has to learn how to use it. The instructions we give to computers are generally referred to as computer programs. There are many different languages we can use to give instructions to computers. This class aims to give you the basics of computer programming that are common to all programming languages without focusing on any particular language.

The class has four modules or units. Each module has a number of lessons that we will discuss in class, or you can watch in online videos at

<http://www.yorktech.com/science/craig/CPT168/lect.htm>

There is also a section of this workbook for each module with explanation, examples and homework problems. There is a test for each module, and homework from the workbook will be taken up at test time.

Module 1 – Introduction to Problem Solving

Lesson 1 - Problem Solving and Variables

Humans love to solve problems! Sometimes we actually end up creating more problems than we solve, but every progress of humanity involves coming to an understanding of a problem and finding a better way to do it. Whether the question is what to wear to school in the morning or how to solve the world's energy problems, there are six steps to problem solving:

- Identify the problem
- Understand the problem
- Identify alternative ways to solve the problem
- Select the best way to solve the problem
- List instructions to solve the problem using the selected method
- Evaluate the solution

Some problems are best solved by repeating a series of steps, whereas other problems need a little experience and common sense to solve. We break problems into two categories.

- **Algorithmic solutions** - solving a problem by following a series of steps - AKA an "algorithm"
- **Heuristic solutions** - solving a problem by using reasoning, knowledge, trial and error.

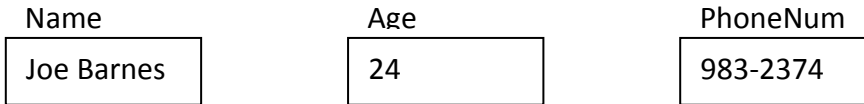
Starting a car always involves the same steps of putting the car in park, inserting the key and turning, but knowing if it is safe to pull out into traffic takes some experience.

Computers are very good at algorithmic problems; humans excel at heuristic problems.

Computers help solve problems by storing and processing data. Each little piece of data is known as a **variable**. If the computer needs to remember my name or a phone number or the number of books in the library it creates a memory allocation – kind of like a little box

to store that data in. Each of these boxes is given a name for our convenience and we can store information there, can retrieve it when needed, or can change the stored value if need be.

Here are three variables:



The first of these, “Name”, stores text information while the second “Age” keeps a number. Each variable has a specific **data type**, that is the kind of information stored.

The most common data types are listed below.

Data Type	Type of Information	Example
String	Text information	“Joe Barnes”
Character	A single letter	“t”
Integer	A whole number	17
Real	A decimal number	17.64552
Logical	True or False	T

Variable names should

- clearly tell what the variable is for
- not have any spaces or weird characters (like ^ or ?)
- be concise, since you will be typing it repeatedly

For example, to store the number of desks in a classroom, any of the following would be acceptable to the computer, but some are clearly better for us as we develop a computer program:

Possible Variable Name	Good Variable Name?
x	Doesn't specify purpose
Number Desks In Class	Can't have spaces
NumberDesksInClass	Too long
#Desks	Can't have weird characters
NumDesks	Not bad!

Some software companies have their programmers add a little more information to the variable name by starting the variable name with a single letter that gives the data type of the particular variable.

- s for string variables
- c for character variables
- i for integer number variables
- n for real number variables
- l for logical variables

Thus, “iNumDesks” may be the best variable possible name, since it quickly and clearly tells what the variable is for and what data type the variable is.

Similar to variables are constants. A **constant** is a piece of data that always has the same value. A couple of constants are the number of days in a week or the mathematical number pi.

Exercise 1.1 Define the following:

- Algorithm _____
- Algorithmic solution _____
- Heuristic solution _____
- Constant _____
- Variable _____

Exercise 1.2 Give a suitable name and data type for each value. Circle those that would be considered constants :

Value	Name	Data Type
The number of days in a week		
The outside temperature		
The percentage of people that can program computers		
The percentage of sales tax		
A phone number		
The last name of students		
Whether or not it is raining		
The letter grade for a class		

Exercise 1.3 Give an example of each of the common data types used in programming languages:

- Integer _____
- Real _____
- Logical _____
- Character _____
- String _____

Lesson 2 - Operators and Truth Tables

Operators do things – they operate, or work on stuff. For example, addition is an operation. $2 + 4 = 6$ takes two things and joins them in a process, or operation, called addition. Multiplication, $2 \times 4 = 8$ is a different operation. Here’s a list of the most basic math operators.

Operator	Symbol	Example
Addition	+	$4 + 3 = 7$
Subtraction	-	$4 - 3 = 1$
Multiplication	x	$4 \times 3 = 12$
Division	/	$4 / 3 = 1.3333333...$
Integer Division	\	$4 \setminus 3 = 1$
Modular Division	Mod	$15 \text{ mod } 12 = 3$
Power	^	$4 ^ 3 = 64$

These will all be familiar, except perhaps for a couple:

- Integer division – returns the integer part of a division problem. In regular division $8/5 = 2.5$, but in integer division we just want the integer part $8 \setminus 5 = 2$
- Modular division – when numbers go up to a certain number and “wraps around” to start over. For example, if it is ten o’clock now and you have to meet someone in five hours, instead of meeting at $10 + 5 = 15$ o’clock, we say we’ll meet at three. $10 + 5 = 15 \text{ mod } 12 = 3$. The clock wraps around after twelve.

Some operators are more important than others. The **order of operations** tells us the order these must happen in.

- Start inside the inner most **parenthesis** and work out
- Exponents are to be done before other operations
- Next, do multiplication and division, from left to right
- Finally, complete the addition and subtraction, again from left to right.

Here are a couple of examples to contemplate. The order of operations is followed in both examples and you arrive at the right answer. If you did not correctly follow the order of operations there are lots of possible wrong answers.

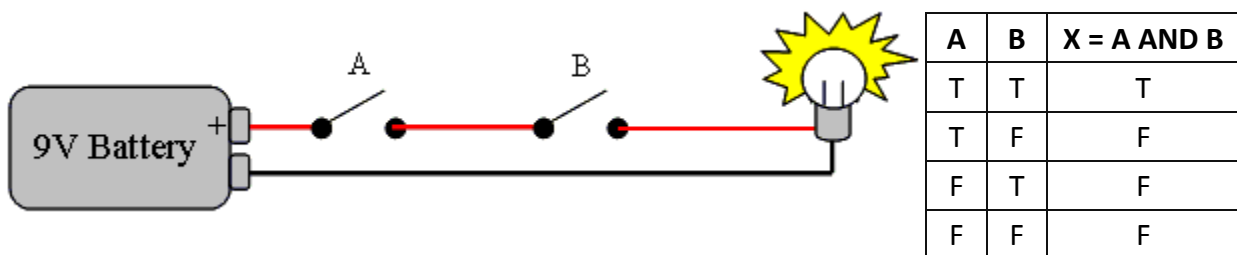
Example #1	Example #2
$3+(4-2)^2$ $= 3+(2)^2$ $= 3+4$ $= 7$	$2((9+2)3+(4-2)^3)$ $= 2((11)3+(2)^3)$ $= 2((11)3+8)$ $= 2(33+8)$ $= 2(41)$ $= 82$

Relational operators compare two quantities and decide if the expression is true or false.

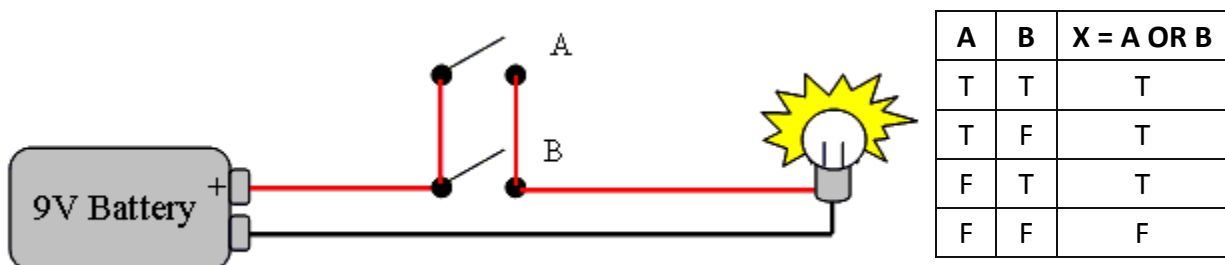
Operator	Symbol	Example
Equal to	=	3 = 3; true
Less than	<	3 < 7; true
Greater than	>	3 > 7; false
Less than or equal	<=	3 <= 3; true
Greater than or equal	>=	3 >= 17; false
Not equal to	<>	3 <> 3; false

When two or more **Logical operators** combine two quantities and decide if the expression is true or false.

We can also consider two logical values, we'll call them A and B. The **AND logical operator**. A AND B means look at both values and if both are true then the whole expression is true, otherwise the whole expression is false. This is just like having a flashlight with two switches like in the picture below – if you want the light to come on, both switches must be closed. The switches are considered true if closed, the light on represents a “true” state. The table beside the picture above is called a **truth table**. It shows all of the possibilities.



The **OR logical operator** compares the states of two variables but yields a true if at least one of them is true.



The **NOT logical operator** simply switches the state of a variable. If A = true, then NOT A = false

A	X = NOT A
T	F
F	T

Combinations of logically expressions can be evaluated one operation at a time. Similar to the mathematical order of operations, we need to start inside the parenthesis and work towards the outside. For example, in the logical expression (A AND B) OR C we would first resolve A AND B. If we call that result X, then we need to consider X OR C. The table below shows all of the possible combinations of A, B and C and the results of A AND B, followed by X OR C. The last column shows which combinations will “turn the light on”.

(A AND B) OR C				
A	B	C	① (A AND B)	① OR C
T	T	T	T	T
T	T	F	T	T
T	F	T	F	T
T	F	F	F	F
F	T	T	F	T
F	T	F	F	F
F	F	T	F	T
F	F	F	F	F

Here’s another example.

(A OR NOT B) AND C					
A	B	C	① NOT B	② A OR ①	② AND C
T	T	T	F	T	T
T	T	F	F	T	F
T	F	T	T	T	T
T	F	F	T	T	F
F	T	T	F	F	F
F	T	F	F	F	F
F	F	T	T	T	T
F	F	F	T	T	F

Exercise 1.4 List the arithmetic operators and give an example problem of each

Operator	Symbol	Example
Addition		
Subtraction		
Multiplication		
Division		
Integer Division		
Modulo Division		
Power		

Exercise 1.5 Evaluate the following mathematical expressions using arithmetic operators and the order of operations. Show the steps to your work

$4 + 3(7-2^2)^3$	$(8+(48+(1.68+(8.08+9))))^2$
$6^{iNum} - 4(rCost-6) \quad \text{if } iNum = 3 \ \& \ rCost = 4.27$	$6+2(4-1) \bmod 5$

Exercise 1.6 List the relational operators and give an example problem of each

Operator	Symbol	Example
Equal to		$3 = 3$; true
Less than		
Greater than		
Less than or equal		
Greater than or equal		
Not equal to		

Exercise 1.7 Evaluate the following relational expressions

Expression	Value (true or false)
$3(4-7)^2 = 25$	
$-6 > 2$	
$5(3-2) \leq 3+4+2$	
$4 \neq 7$	

Lesson 3 - Functions

Functions are similar to operators in that they “do things” – That is, they have a function. However, a function is a piece of programming that has already been written and all we have to do is call that function. Here for example are a few of the function that have already been written to perform various mathematical tasks. Rather than us writing a new piece of code to take the square root of 47, we simply type `nSqrtRoot = Sqrt(47)` – it calls the function `Sqrt`, which takes the square root and assigns it to the variable `nSqrtRoot`.

Math Functions		
<code>Sqrt(N)</code>	Square Root	<code>Sqrt(49) = 7</code>
<code>Abs(N)</code>	Absolute Value	<code>Abs(-17) = 17</code>
<code>Round(N,n)</code>	Round a number N to n decimal places	<code>Round(13.2385,2) = 13.24</code>
<code>Integer(N)</code>	Give the integer part of a number	<code>Integer(15.148) = 15</code>
<code>Random</code>	Generate a random # between 0 and 1	<code>Random = 0.245877513</code> <code>15*Random = 3.688162695</code>
<code>Sign(N)</code>	Tell if number is positive or negative 1 if +, 0 if -	<code>Sign(-3.76) = 0</code>

Rather than operating on numbers, these next functions are written to work with strings.

String Functions		
<code>+ (Concatenation)</code>	Join two strings	<code>“super” + “man” = “superman”</code>
<code>Mid(S, n1, n2)</code>	Get the middle part of string, starting at char n1 and getting n2 chars	<code>Mid(“superman”,3,4) = “perm”</code>
<code>Left(S,n)</code>	Get the n chars on the left of a string	<code>Left(“superman”,5) = “super”</code>
<code>Right(S,n)</code>	Get the n chars on the right of a string	<code>Right(“superman”,5) = “erman”</code>
<code>Length(S)</code>	Get the # of char in a string	<code>Length(“superman”) = 8</code>

These functions convert between strings and numbers.

Conversion Functions		
<code>Value(S)</code>	Change a string into a number	<code>Value(“57”) = 57</code>
<code>String(N)</code>	Change a number into a string	<code>String(57) = “57”</code>

Here are some functions that perform statistical duties.

Statistical Functions		
Average(list)	Give the average of a list of numbers	Average(4,6,8) = 6
Max(list)	Give the largest # in a list of numbers	Max(4,6,8) = 8
Min(list)	Give the smallest # in a list of numbers	Min(4,6,8) = 4
Sum(list)	Add a list of numbers	Sum(4,6,8) = 18

There are other types of information we can pull from the computer, such as date and time.

Utility Functions		
Date	Returns the date	Date = 03/27/2010
Time	Returns the time	Time = 11:25:44
Error	Returns control if an error occurs	

One of the easiest places to “play” with the data types and functions described in this section is in a spreadsheet. A spreadsheet is a computer program that allows the user to place data into a table. The data can be strings, numbers, dates, anything. Here’s a screenshot of some spreadsheet data.

	A	B	C
1	Row#	Last Name	Car
2	1	Jones	1971 Buick Skylark
3	2	Wilson	1986 Subaru Wagon
4	3	Smith	1973 VW Thing
5	4	George	2002 Ford E-350
6			
7			

All of these functions we’ve discussed, and many more, are built into spreadsheets. Here is an example of five numbers in the first row. Each cell has an address, the 4 is in the column marked A and row 1, so its address is A1. The sum, average and count are easily calculated by typing the expressions shown in quotes.

	A	B	C	D	E
1	4	3	7	8	12
2					
3	Sum	34	"=SUM(A1:E1)"		
4	Average	6.8	"=AVERAGE(A1:E1)"		
5	Count	5	"=COUNT(A1:E1)"		

	A	B	C	D
1				
2	Cheese is good			
3				
4	Length	14		
5	Left Characters	Chee	"=LEFT(A2,4)"	
6	Right Characters	is good	"=RIGHT(A2,7)"	
7	Middle	se is goo	"=MID(A2,5,9)"	

Given a string, here “cheese is good” we can use string functions.

Exercise 1.10 Evaluate the following relational expressions. Try these in a spreadsheet as well. Use the following variables:

- iNumCars = 7
- iNumStudents= 11
- cNumCars = "7"
- cNumStudents= "11"
- sSomeString = "the rain in Spain falls mainly on the plain"

Expression	Value
=AVERAGE(iNumCars,iNumStudents)	
=MAX(iNumCars,iNumStudents)	
= iNumCars + iNumStudents	
= cNumCars + cNumStudents	
=MID(sSomeString,15,7)	
=LEN(sSomeString)	

Lesson 4- Introduction to JavaScript

Similar to the way people converse in different languages, like English or Italian or Swahili, each programming language has its own “syntax” – the exact way the commands must be given to the computer. The easiest computer programming is called scripting. JavaScript is a small set of simple scripting commands that we’ll use to illustrate the concepts of variables, functions and operators we have been discussing.

JavaScript (JS) can be used with any internet browser and the output can produce or modify a web page. Web pages are most often written in **HTML** – Hyper Text Markup Language, which uses “tags” to mark the beginning and end of something. For example

```
<strong> The grass is green today </strong>
```

prints the line “**The grass is green today**” in strong (same as bold) on the screen. JS can be inserted directly into the HTML

Here’s a very simple example of JavaScript.

The Script	What the User Sees
<pre><html> <script type="text/javascript"> document.write("JavaScript is easy!"); </script> </html></pre>	JavaScript is easy!

In this example, the HTML tags `<html>` and `</html>` simply mark the beginning and end of the HTML document. The line that reads `<script type="text/javascript">` tells the browser that some JS is coming. `document.write("JavaScript is easy!");` instructs the HTML to write whatever is in quotes on the webpage. `</script>` tells the HTML doc that the script is over. The browser starts reading code at the top of the page and sequentially reads and executes one line at a time until it gets to the bottom.

Variables are easy to use in JS.

The Script	What the User Sees
<pre><html> <script type="text/javascript"> A = 5; B = 7; x=A+B; document.write(x); document.write("<p>"); document.write("That was easy!"); </script> </html></pre>	12 That was easy!

It's easy to dress it up a little:

The Script	What the User Sees
<pre><html> <script type="text/javascript"> A = 5; B = 7; x=A+B; document.write("Let's do math!<p>"); document.write(A, " + ", B, " = ", x); document.write("<p>"); document.write("That was easy!"); </script> </html></pre>	<pre>Let's do math! 5 + 7 = 12 That was easy!</pre>

COMPUTERS ARE IGNORANT!– they only do exactly what you tell them to do. They will not try to figure out what you meant to say. If your code is wrong you will not get the right result. Some things worth mentioning here:

- Every line of JS needs a semicolon at the end.
- All HTML tags need to be inside quotes. For example <p> means start a new paragraph.
- Variables should not be in quotes.
- All the text you want displayed should be in quotes.
- Use commas to separate things in quotes and things not in quotes.
- There are a couple of ways to try JS programming:
 - Use a text editor (like Notepad or Word). Save the file as *.htm. Double click the file to open with your favorite browser.
 - Go to http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables and either copy and paste any of these examples into or edit the examples they have. The left side will show the HTML/JS and the right will show the output.

There were lots of Math operators we discussed. Let's see how these look in JS. Any JS line with a double slash // will not be read by the computer – it's called a comment and simply makes code easier for humans to read.

The Script	What the User Sees
<pre><html> <script type="text/javascript"> //Define a couple variables Num1 = 45.55215; Num2 = -17; RndNum = Math.random() //Do the math stuff SQRTNum1 = Math.sqrt(Num1); ABSNum2 = Math.abs(Num2); ROUNDNum1 = Math.round(Num1); INTNum1 = Math.floor(Num1); //Display the results document.write("The SQRT of ", Num1, " is ",</pre>	<pre>The SQRT of 45.55215 is 6.749233289789292 The ABS of -17 is 17 The ROUND of 45.55215 is 46 The INT of 45.55215 is 45 A random # is 0.4037266626413717 A random # between 0 and 100 is 40.372666264137166</pre>

<pre> SQRTNum1); document.write("<p>"); document.write("The ABS of ", Num2, " is ", ABSNum2); document.write("<p>"); document.write("The ROUND of ", Num1, " is ", ROUNDNum1); document.write("<p>"); document.write("The INT of ", Num1, " is ", INTNum1); document.write("<p>"); document.write("A random # is ", RndNum); document.write("<p>"); document.write("A random # between 0 and 100 is ", 100*RndNum); </script> </html> </pre>	
---	--

The variable data types we discussed were Integer, Real, Logical, Character and String. In JavaScript, variables are “loosely typed”, meaning a variable can hold any data type, and can even change type. Having that power can be good, but you do have to be careful. Most languages are not as flexible. Here’s an JS example using strings.

The Script	What the User Sees
<pre> <html> <script type="text/javascript"> A = "School"; B = "Bus"; x=A+B; document.write(x,"<p>"); document.write(A, " + ", B, " = ", x); document.write("<p>"); document.write("That was easy!"); </script> </html> </pre>	<pre> SchoolBus School + Bus = SchoolBus That was easy! </pre>

The string operators we discussed look a little different, but work in JS.

The Script	What the User Sees
<pre> <html> <script type="text/javascript"> //define variables String1 = "mousetrap"; //count the number of characters in String1 StrLen = String1.length; //get the characters starting at #3 to #6 //corresponds to MID function //NOTE - JS counts starting at zero StrMid = String1.slice(2,5); //get the first 4 characters - LEFT function StrLeft = String1.slice(0,4); document.write("The number of characters in ", String1, </pre>	<pre> The number of characters in mousetrap is 9 The MID of mousetrap is use The LEFT of mousetrap is mous The RIGHT of mousetrap is trap </pre>

<pre> " is "); document.write(StrLen); document.write("<p>"); document.write("The MID of ", String1, " is "); document.write(StrMid); document.write("<p>"); document.write("The LEFT of ", String1, " is "); document.write("",StrLeft,"<p>"); //get the last 4 characters //corresponds to RIGHT function document.write("The RIGHT of ", String1, " is "); document.write("",String1.slice(StrLen- 4,StrLen),""); </script> </html> </pre>	
---	--

Incidentally, the best and most complete reference and tutorial for JavaScript is <http://www.w3schools.com/js/default.asp>. Check it out.

Exercise 1.11 Show the output of the following program.

The Script	What the User Sees
<pre> <html> <script type="text/javascript"> //Define a couple variables Num1 = 100; Num2 = 25; //Do the math stuff SQRTNum1 = Math.sqrt(Num1); SQRTNum2 = Math.sqrt(Num2); SumSqrt = SQRTNum1 + SQRTNum2; NumAv = (Num1 + Num2)/2 //Display the results document.write("The average of the numbers is ", NumAv,"<p>"); document.write("The SQRT of ", Num1 , " is ", SQRTNum1, "<p>"); document.write("The SQRT of ", Num2 , " is ", SQRTNum2, "<p>"); document.write("The sum of the sqrts is ", SumSqrt,"<p>"); </script> </html> </pre>	

Exercise 1.12 Fill in the missing lines of code in the following program.

The Script	What the User Sees
<pre> <html> <script type="text/javascript"> //Define a couple variables //Do the math stuff //Display the results document.write("Num1 is ", Num1,"<p>"); document.write(document.write("SumNum is ", SumNum,"<p>"); document.write("DiffNum is ", DiffNum, "<p>"); document.write("DivNum is ", DivNum, "<p>"); </script> </html> </pre>	<pre> Num1 is 27 Num2 is 14 SumNum is 41 DiffNum is 13 DivNum is 1.928571429 </pre>

Exercise 1.13 Fill in the missing lines of code in the following program.

The Script	What the User Sees
<pre> <html> <script type="text/javascript"> //Define a couple variables Str1 = "mississippi"; Str2 = "mighty mouse"; //Do the string stuff Str1Cut = Str2Front = Str2Back = StrHero = //Display the results document.write(document.write(document.write(</pre>	<pre> Str1Cut is mississip Str2Front is mighty Str2Back is mouse Who saved the day? mighty mouse Where did he save it? mighty mississip </pre>

<pre>document.write(document.write(document.write(document.write(</script> </html></pre>	
---	--

- Evaluate mathematical expression using the order of operations using arithmetic operators
- Evaluate relational expressions using relational operators
- Define the logical operators: NOT, AND, OR

Module 2 – Control Structures

The real power of computer programming is the ability of the computer to evaluate variables and conditions and to change the outcome of the program based on these values. A well written program has many logical outcomes – it can

- Decide to take different path **IF** a given condition is true or not
- Repeat a step **FOR** a certain number of times
- Repeat a step **WHILE** a condition is true
- Choose one of many paths depending on the **CASE**

Lesson 1 - Flowcharts & Pseudocode

What if we needed a program to calculate the cost of carpeting a room. In the program, the user should be asked for room width, room length and the carpet




cost per square yard. The computer will do the calculations and then display the cost. Looking at this list we can see that we will need to create the variables at the right.

Flowcharts are used to draw a picture of the logic and steps of your program. In this example we must get the dimensions of the room, calculate the area and find the cost.

Here are a couple of symbols we need to get started.

Steps of Program	
•	Get the length of the room in feet
•	Get the width of the room in feet
•	Multiply length x width to get area
•	Convert this area to square yards by dividing by 9
•	Get the cost of carpet per square yard
•	Multiply number of square yards x cost per square yard
•	Display Cost




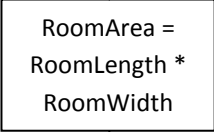
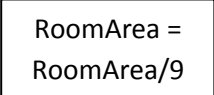

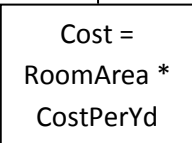
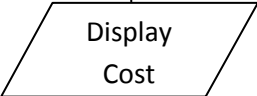
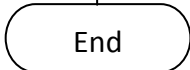
Variables Needed	
•	RoomLength
•	RoomWidth
•	RoomArea
•	CostPerYard
•	Cost

Flowchart Symbol	Purpose
	Terminator – goes at the beginning and end of the flowchart or module
	Process – a step in the program
	Input/Output – either gets information from the user or displays information to the user

In the flowchart below, start at the top and follow the line down the flowchart to see the order the program follows. You can see that the program

- asks the user a couple of questions, shown by the **Input/Output** parallelogram,
- does a couple calculations, shown in the **Process** rectangles,
- finally displays the cost to the user, either on the screen or to a printer, another **Input/Output** step.

The flowchart shows the entire logical process, order and steps of the program. It is a nice picture of what is happening.

Flowchart	Steps of Program
	Terminator – Start the program
	Input/Output - Get the length of the room in feet
	Input/Output - Get the width of the room in feet
	Process - Multiply length * width to get area
	Process - Convert this area to square yards by dividing by 9
	Input/Output - Get the cost of carpet per square yard
	Process - Multiply number of square yards x cost per square yard
	Input/Output - Display Cost
	Terminator - End the program

The next step in developing a program is to convert the flowchart into what we call “**psuedocode**”. This is not quite programming syntax, you don’t have to worry about getting semicolon and parenthesis in the right place, but it’s close enough to make it very easy to transition to actual programming language.

A couple of psuedocode commands are listed here:

- DECLARE – define a variable
- INPUT – ask the user for data and store it to a variable
- PRINT – display data to the user, usually on the screen
- = - assign a value to a variable

For the carpet cost problem, we look at the flowchart and write the following:

```

DECLARE RoomLength, RoomWidth, RoomArea,
CostPerYard, Cost AS REAL
INPUT "Please input room length", RoomLength
INPUT "Please input room width", RoomWidth
RoomArea = RoomLength * RoomWidth
RoomArea = RoomArea /9
INPUT "Please input the cost per sq.yd", CostPerYd
Cost = RoomArea * CostPerYd
PRINT Cost

```

Each computer language will have different syntax, and will look different, but the logic, the flow, the structure of the program will always be the same. For example, in JavaScript we would write the following:

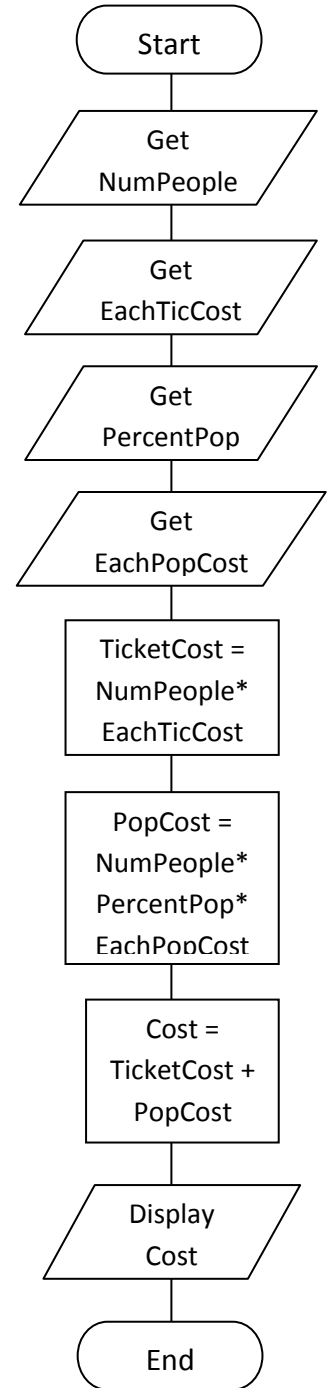
```

<html>
<body>
<script type="text/javascript">
//Get the room dimensions
RoomLength=prompt("Please enter the Room Length");
RoomWidth=prompt("Please enter the Room Width");

// Calc the room area
RoomArea = RoomWidth*RoomLength;
document.write("The room area is ",RoomArea, " square
feet<p>");
RoomArea=RoomArea/9;
document.write("The room area is ",RoomArea, " square
yards <p>");

//Get the carpet cost
CostPerYard=prompt("Please enter the Cost per yard");

```



```
Cost = CostPerYard*RoomArea;
```

```
//Display the cost  
document.write("The cost is $",Cost);  
</script>  
</body>  
</html>
```


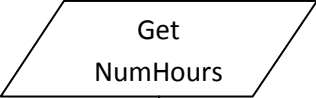
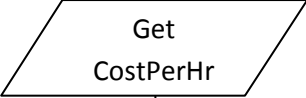
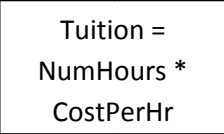
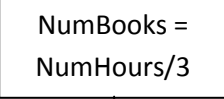
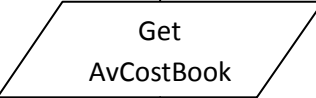
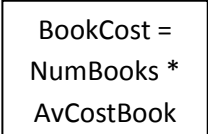
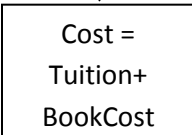

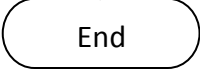
Let's try another program – this one will calculate the cost for a group to go to the movie.

Steps
<ul style="list-style-type: none">• Get # of people - NumPeople• Get cost per movie ticket - EachTicCost• Get % of people that want popcorn - PercentPop• Get cost of popcorn - EachPopCost• Calculate cost of tickets - TicketCost• Calculate popcorn cost - PopCost• Calculate total cost – Cost• Display Cost

Variable Name	Data Type	Description
NumPeople	Int	# of people
EachTicCost	Real	cost per movie ticket
PercentPop	Real	% of people that want popcorn
EachPopCost	Real	Get cost of popcorn
TicketCost	Real	Calculate cost of tickets
PopCost	Real	Calculate popcorn cost
Cost	Real	Calculate total cost

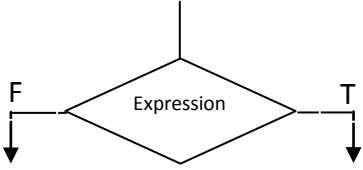
```
INPUT "Please input # of people", NumPeople  
INPUT "Please input cost per ticket", EachTicCost  
INPUT "Please input % of people that want popcorn", PercentPop  
INPUT "Please input cost per popcorn", EachPopCost  
TicketCost = NumPeople * EachTicCost  
PopCost = NumPeople* PercentPop* EachPopCost  
Cost = TicketCost + PopCost  
PRINT Cost
```

Exercise 2.1 Write in the descriptions of what this program does

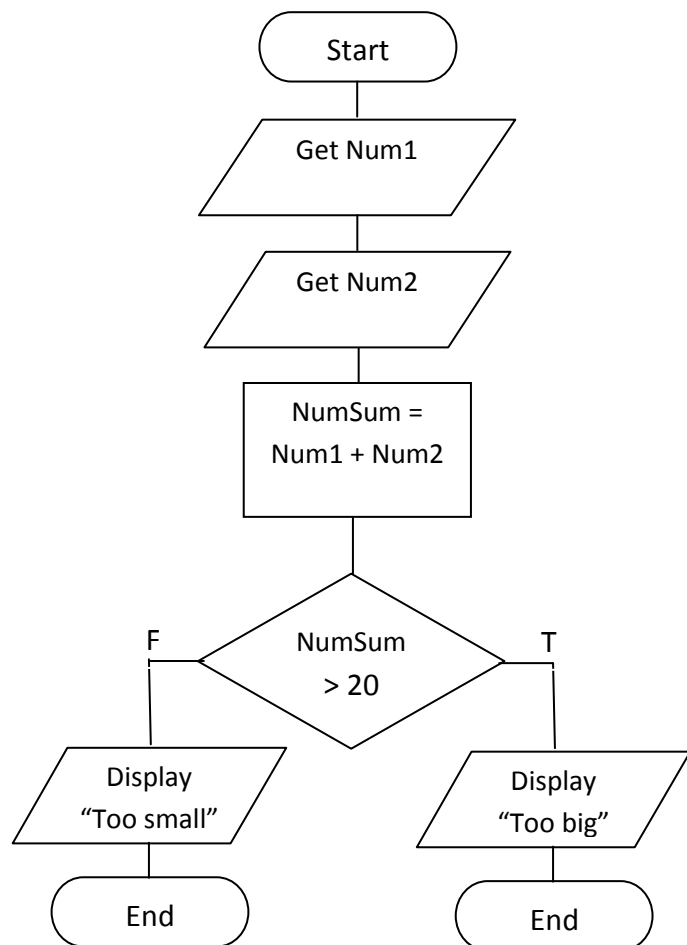
Flowchart	Steps of Program
	<p>Terminator - Start</p>
	
	
	
	
	
	
	
	
	<p>Terminator - End</p>

Lesson 2- Decisions Using IF Statements

Decisions are important in computer programming. The flowchart symbol is shown below, and clearly the path of the program goes different ways if the expression evaluates true or false.

Flowchart Symbol	Purpose
	Decision – chooses a path based on a logic evaluation

For example, if a program gets two numbers from a user, and adds the numbers, the program could return one of two possible responses depending on the size of the sum.



An **IF** statement makes decisions based on the state of a relational comparison. Here are a couple of ways to use an IF statement.

Program	Purpose
<pre> DECLARE Num1, Num2, SumNum AS REAL INPUT "Please input first number", Num1 INPUT "Please input second number", Num2 SumNum = Num1 + Num2 IF SumNum > 20 THEN PRINT "Too big" END IF </pre>	<p>Add two user input numbers and print "Too big" if the sum > 20</p>
<pre> DECLARE Num1, Num2, SumNum AS REAL INPUT "Please input first number", Num1 INPUT "Please input second number", Num2 SumNum = Num1 + Num2 IF SumNum > 20 THEN PRINT "Too big" ELSE PRINT "Too small" END IF </pre>	<p>Add two user input numbers and print "Too big" if the sum is greater than 20, or print "Too small" if the sum is <= 20</p>

Let's design a program to ask a user their favorite word; if they enter a word with more than 8 characters the program responds with "That's a big word!", otherwise it returns "Wimpy word!"

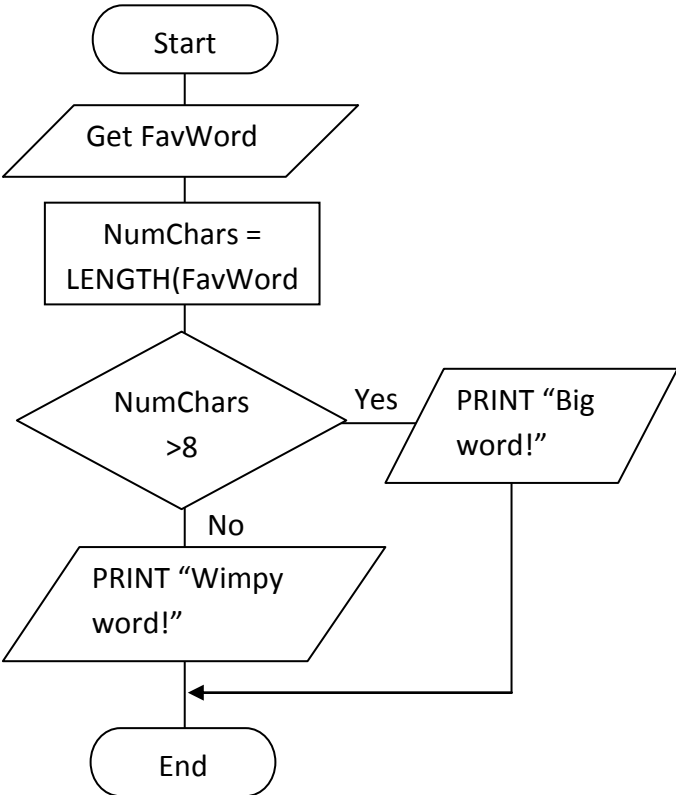
The steps we'll need to perform are:

- Ask user for word
- Count characters in word
- Decide if >8 and print appropriate response


Next, let's plan the variables we need to have

Variable Name	Data Type	Description
FavWord	String	Store the users favorite word
NumChars	Integer	The number of characters in FavWord

Then we can draw up a flowchart and translate that into psuedocode.

Flowchart	Psuedocode
 <pre>graph TD; Start([Start]) --> GetFavWord[/Get FavWord/]; GetFavWord --> CalcNumChars[NumChars = LENGTH(FavWord)]; CalcNumChars --> Decision{NumChars > 8}; Decision -- Yes --> PrintBig[/PRINT "Big word!"/]; Decision -- No --> PrintWimpy[/PRINT "Wimpy word!"/]; PrintBig --> End([End]); PrintWimpy --> End;</pre> <p>The flowchart starts with an oval labeled 'Start'. It proceeds to a parallelogram labeled 'Get FavWord'. This is followed by a rectangle labeled 'NumChars = LENGTH(FavWord)'. A diamond-shaped decision box contains 'NumChars > 8'. A 'Yes' path leads to a parallelogram labeled 'PRINT "Big word!"'. A 'No' path leads to a parallelogram labeled 'PRINT "Wimpy word!"'. Both paths converge to an oval labeled 'End'.</p>	<pre>DECLARE FavWord AS STRING DECLARE NumChars AS INT INPUT FavWord NumChars = LENGTH(FavWord) IF NumChars > 8 PRINT "Big Word!" ELSE PRINT "Wimpy Word!" END IF</pre>

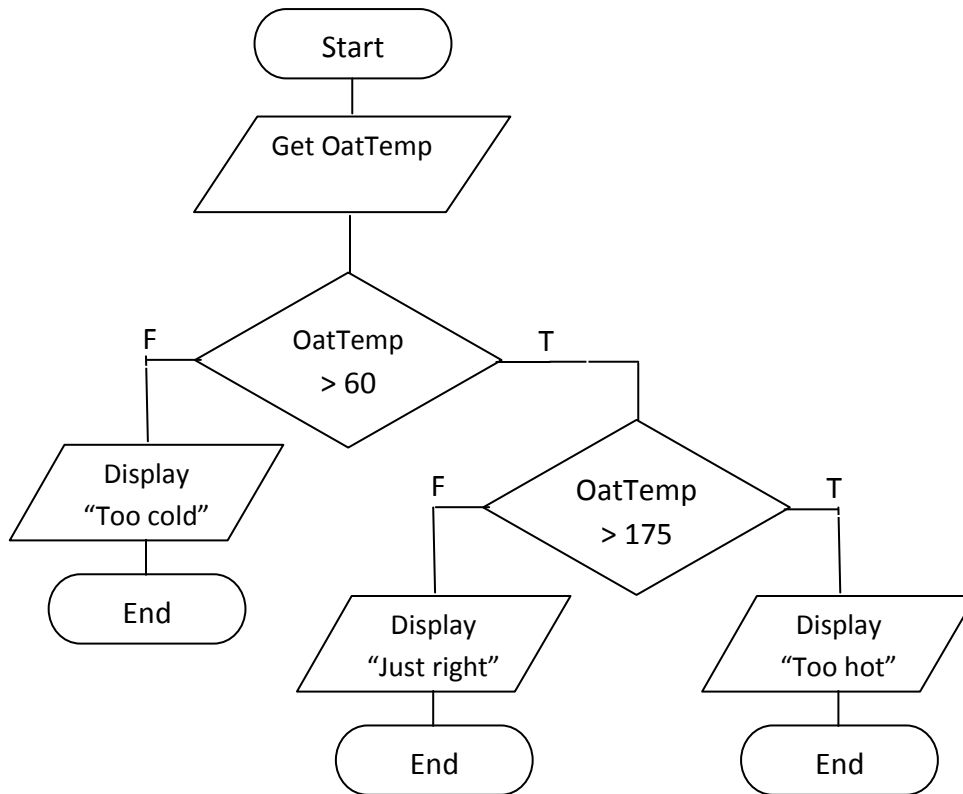
Exercise 2.4 Design a program that will generate a random integer between 0 and 5 and then to ask the user to guess that number. If they guess the number, print “Good Guessing” otherwise print “No Way!”

Flowchart	Psuedocode
 <pre> graph TD Start([Start]) --- End([End]) </pre>	

Variable Name	Data Type	Description

Lesson 3- Nesting Decisions

Sometimes we need to make decisions within decisions. These are referred to as nesting decisions. For example, if Goldilocks writes a program to check the temperature of the bear’s oatmeal, the program could return one of three possible responses depending on the temperature.



Her program would look like the following.

Pseudocode	Purpose
<pre> DEclare OatTemp AS REAL INPUT "Please input oatmeal temp", OatTemp IF OatTemp > 60 THEN IF OatTemp > 175 THEN PRINT "Too hot" ELSE PRINT "Just right" END IF ELSE PRINT "Too cold" END IF </pre>	<p>IF/ Print:</p> <ul style="list-style-type: none"> • < 60 / "Too cold" • 60 to 175 / "Just right" • > 175 / "Too hot"

Steps
<ul style="list-style-type: none"> • Get patient fee

Notice how indenting the text for each level of decision helps us to read the program. That way we can visually match up IF, ELSE, and END IF that go together. The computer doesn't care, but it makes a big difference for us mere mortals.

A doctor's office might make use of a nested decision in their billing office; imagine if a patient has insurance they pay a 20% copay, but if they do not, then they need to pay cash or credit. Developing this program would require the following:

- Ask if have insurance
- IF yes,
 - calculate copay/amount due
 - PRINT Amount Due
- IF no,
 - Amount due = fee
 - Ask if cash or credit
 - IF credit
 - ask MC or VISA
 - PRINT Amount charged
 - IF cash
 - Print Amount Due

Variable Name	Data Type	Description
Fee	Real	Amount of bill
InsYesNo	Logical	Does the patient have insurance
AmtDue	Real	Amount patient owes today
PayMeth	String	Method of payment
CredCard	String	Which credit card uses


```

Pseudocode

DECLARE Fee, AmtDue AS REAL
DECLARE InsYesNo AS LOGICAL
INPUT "Please input today's fee", Fee
INPUT "Does the patient have insurance?" ,InsYesNo
IF InsYesNo = 1 THEN
  AmtDue = .20*Fee
  PRINT "Today's fee is $",AmtDue
ELSE // InsYesNo = 0
  AmtDue = Fee
  INPUT "Is the patient paying by credit card or cash", PayMeth
  IF PayMeth = "card" THEN
    INPUT "Mastercard or VISA?",CredCard
    PRINT "$",AmtDue, " has been charged to ", CredCard
  ELSE //pay Cash
    PRINT "Thank you for paying $", AmtDue, " by cash"
  END IF
END IF


```

Exercise 2.6 Design a program that will ask for the outside temperature, and whether it is raining. If the user inputs a number smaller than 50, print “You might want a warm jacket” otherwise if it’s not raining print “No coats today!” or if it is, print “Still need a rain coat”

Flowchart	Pseudocode
	

Variable Name	Data Type	Description

Exercise 2.7 Design a program that will ask for the outside temperature. If the user inputs a number smaller than 40 print “Coat needed”, if it’s greater than 55 print “Nice day”, otherwise print “How about a jacket?”

Flowchart	Psuedocode
 <pre> graph TD Start([Start]) End([End]) </pre>	

Variable Name	Data Type	Description

Lesson 4- Loops Define and use counters and accumulators

Computers don't mind doing the same thing over and over, and sometimes we need a program to do one of the following:

- Repeat a step **FOR** a certain number of times
- Repeat a step **WHILE** a condition is true
- Repeat a step **UNTIL** a condition is no longer true

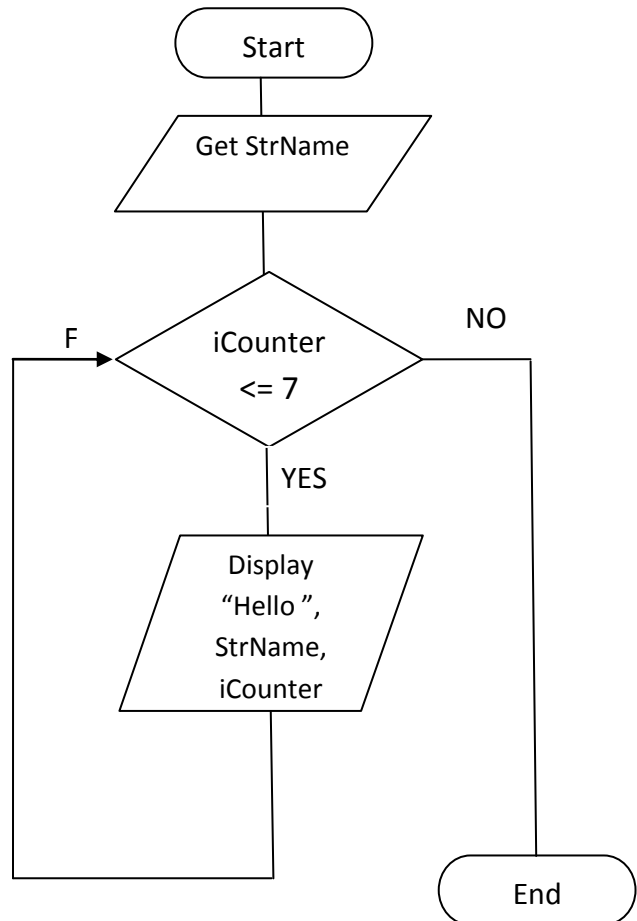
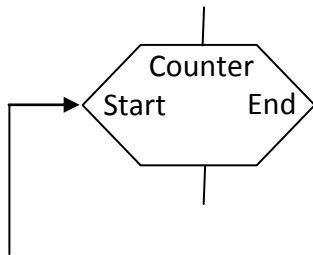
A **FOR** loop repeats a set of steps a given number of times. For example, the following psuedocode prints a statement 7 times.

Pseudocode	What the User Sees
<pre> DECLARE StrName AS REAL DECLARE iCounter AS INTEGER INPUT "Please input your name", StrName FOR iCounter =1 to 7 PRINT "Hello ", StrName, iCounter NEXT iCounter </pre>	<pre> Please input your name > John Hello John1 Hello John2 Hello John3 Hello John4 Hello John5 Hello John6 Hello John7 </pre>

Notice the FOR line starts iCounter at 1 and tells it to go up to 7. The line "NEXT iCounter" increments iCounter. In other words, it adds one every time the line executes until iCounter = 7, at which point the loop stops repeating.

All of the loop structures can be shown in a flowchart using a Decision diamond. This program, would be depicted as shown at the right.

Sometimes though you'll also see the Decision diamond replaced with something like the symbol below.



A **WHILE** loop repeats a set of steps as long as a condition is true. For example, the following pseudocode prints the statement as long as iSum <15.

Pseudocode	What the User Sees
DECLARE iSum AS INTEGER DECLARE StrName AS STRING INPUT "Please input your name", StrName iSum = 2 WHILE iSum < 15 PRINT iSum iSum = iSum +4 WHILE END PRINT "Thanks ", StrName	Please input your name > John 2 6 10 14 Thanks John

In the FOR loop the variable iCounter simply counted how many times the loop ran, so we call that a **counter**. Notice here that iSum is going up by 4 every time the While loop executes. We refer to a number that goes up by some amount on each run of the loop an **accumulator**.

An **UNTIL** loop repeats a set of steps until a condition is true. For example, the following pseudocode prints the statement as long as iSum <15.

Pseudocode	What the User Sees
DECLARE iSum AS INTEGER DECLARE StrName AS STRING INPUT "Please input your name", StrName iSum = 2 UNTIL iSum > 15 PRINT iSum iSum = iSum +4 UNTIL END PRINT "Thanks ", StrName	Please input your name > John 2 6 10 14 Thanks John

While the output in this example looks the same as the WHILE loop example, there are subtle differences that can be important. The order of the statements is also important – simply changing the order of the two lines inside the loop gives a different result.

Pseudocode	What the User Sees
DECLARE iSum AS INTEGER DECLARE StrName AS STRING INPUT "Please input your name", StrName iSum = 2 UNTIL iSum > 15 iSum = iSum +4 PRINT iSum UNTIL END PRINT "Thanks ", StrName	Please input your name > John 6 10 14 Thanks John

Exercise 2.8 Show what the user would see for the following program and fill in the variable table.

Pseudocode	What the User Sees
<pre> DECLARE iCount AS INT DECLARE iLetter AS INT PRINT "Truth Table Column Generator" PRINT FOR iCount = 0 to 1 IF iCount = 0 iLetter = F ELSE iLetter = T END IF PRINT iLetter NEXT iCount </pre>	

Variable Name	Data Type	Description

Exercise 2.9 Write the psuedocode to produce the following output and fill in the variable table. USE A FOR LOOP.

Psuedocode	What the User Sees
	Counting by sevens 0 7 14 21 28 35 42

Variable Name	Data Type	Description

Exercise 2.10 Write the psuedocode to produce the following output and fill in the variable table. USE A WHILE LOOP.

Psuedocode	What the User Sees
	Counting by sevens 0 7 14 21 28 35 42 49 56 63

Variable Name	Data Type	Description

Lesson 5 - Nesting Loops

Sometimes it is necessary to build a loop inside another loop – we call this a nested loop. Let's say we need to build a digital clock. We need to count to twelve over and over for the hours. Inside that, we need to count to 60 for the minutes. Inside that we need to count 60s for the seconds. The Psuedocode could be

Psuedocode	What the User Sees
DECLARE iHr, iMin, iSec AS INTEGER	1:1:1
FOR iHr = 1 TO 12	1:1:2
FOR iMin = 1 TO 59	1:1:3
FOR iSec = 1 TO 59	1:1:4
PRINT iHr,":",iMin,":",iSec	... later
NEXT iSec	1:2:45
NEXT iMin	1:2:46
NEXT iHr	... much later
	5:17:59
	5:18:00

A word about loops: be careful that there is a way out of your loop. I just wrote the code above in JavaScript and tested it in the W3 to try it out – five minutes later it was still churning away with no way to cut it off and I eventually had to kill my browser window. This program will print $12 * 59 * 59 = 41,772$ times, but it will take some time and may grab up all of the memory your computer has to offer.

Incidentally, the JavaScript clock code looks like this - there is no NEXT command at the bottom of the loop, everything is built in at the command. The braces show where the loop starts and stops.

```
<html>
<body>
<script type="text/javascript">
for (iHr = 1; iHr <= 12; iHr++)
{
    for (iMin = 1; iMin <= 59; iMin++)
    {
        for (iSec = 1; iSec <= 59; iSec++)
        {
            document.write(iHr,":",iMin,":",iSec,"<br/>");
        }
    }
}
</script>
</body>
</html>
```

It is also very easy to write a loop that never terminates – we call these “infinite loops”. For example, if I tell the computer to run the following program, why doesn’t the loop ever get finished?

Pseudocode	What the User Sees
<pre> DECLARE iNum AS INTEGER iNum = 2 WHILE iNum > 1 PRINT iNum iSum = iNum +4 WHILE END </pre>	<pre> 2 6 10 14 18 22 26 ... goes forever </pre>

Another example of nested loops might come about during a school field trip – four buses are going to carry 220 students that will be assigned seats by alphabetical order - bus 1 to the first 55 students, bus 2 to the next 55 and so on. We need a program that prints stickers to put on the student’s suitcases.

Pseudocode	What the User Sees
<pre> DECLARE iBus, iStud AS INTEGER FOR iBus = 1 TO 4 FOR iStud = 1 TO 55 PRINT "Bus#: ",iBus," / Student# ",iStud NEXT iStud NEXT iBus </pre>	<pre> Bus#: 1 / Student# 1 Bus#: 1 / Student# 2 Bus#: 1 / Student# 3 ... later Bus#: 3 / Student# 55 Bus#: 4 / Student# 1 Bus#: 4 / Student# 2 ... </pre>

Exercise 2.11 Show the output of the following program and fill in the variable table.

Pseudocode	What the User Sees
<pre> DECLARE iCount1, iCount2, iCount3 AS INT DECLARE iLetter1, iLetter2, iLetter3 AS INT PRINT "Truth Table Column Generator" PRINT FOR iCount1 = 0 to 1 FOR iCount2 = 0 to 1 FOR iCount3 = 0 to 1 IF iCount1 = 0 iLetter1 = F ELSE iLetter1 = T END IF IF iCount2 = 0 iLetter2 = F ELSE iLetter2 = T END IF IF iCount3 = 0 iLetter3 = F ELSE iLetter3 = T END IF PRINT iLetter1, iLetter2, iLetter3 NEXT iCount3 NEXT iCount2 NEXT iCount1 </pre>	

Variable Name	Data Type	Description

Exercise 2.12 Write the psuedocode to produce the following output and fill in the variable table.

Psuedocode	What the User Sees
	LAB TABLE ASSIGNMENTS Table 1: Student 1 Table 1: Student 2 Table 1: Student 3 Table 1: Student 4 Table 2: Student 1 Table 2: Student 2 Table 2: Student 3 Table 2: Student 4 Table 3: Student 1 Table 3: Student 2 Table 3: Student 3 Table 3: Student 4

Variable Name	Data Type	Description

Lesson 6 - Case

When we looked at the IF statement to make decisions, there were only two possible outcomes – having more than two outcomes required nested loops, which can get messy fast. A CASE statement (sometimes also called a SWITCH) provides lots of possibilities.

When Goldilocks wrote her oatmeal temperature program, it looked like this:

```
DECLARE OatTemp AS REAL
INPUT "Please input oatmeal temp", OatTemp
IF OatTemp > 60 THEN
  IF OatTemp > 175 THEN
    PRINT "Too hot"
  ELSE
    PRINT "Just right"
  END IF
ELSE
  PRINT "Too cold"
END IF
```

If she had only known about CASE statements...

```
DECLARE OatTemp AS REAL
INPUT "Please input oatmeal temp", OatTemp
CASE
  OatTemp <= 60
    PRINT "Too cold"

  OatTemp = 82
    PRINT "Ahhh, perfect!"

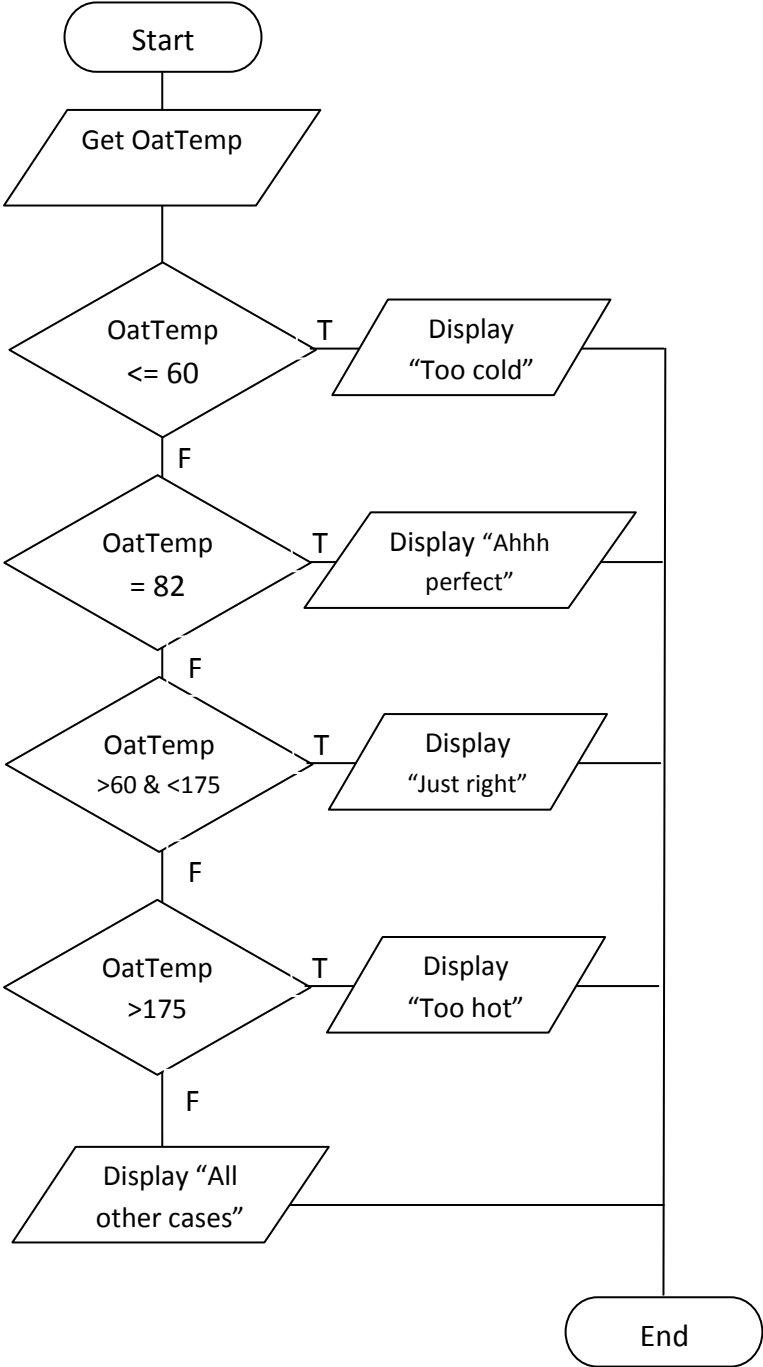
  OatTemp > 60 and OatTemp < 175
    PRINT "Just right"

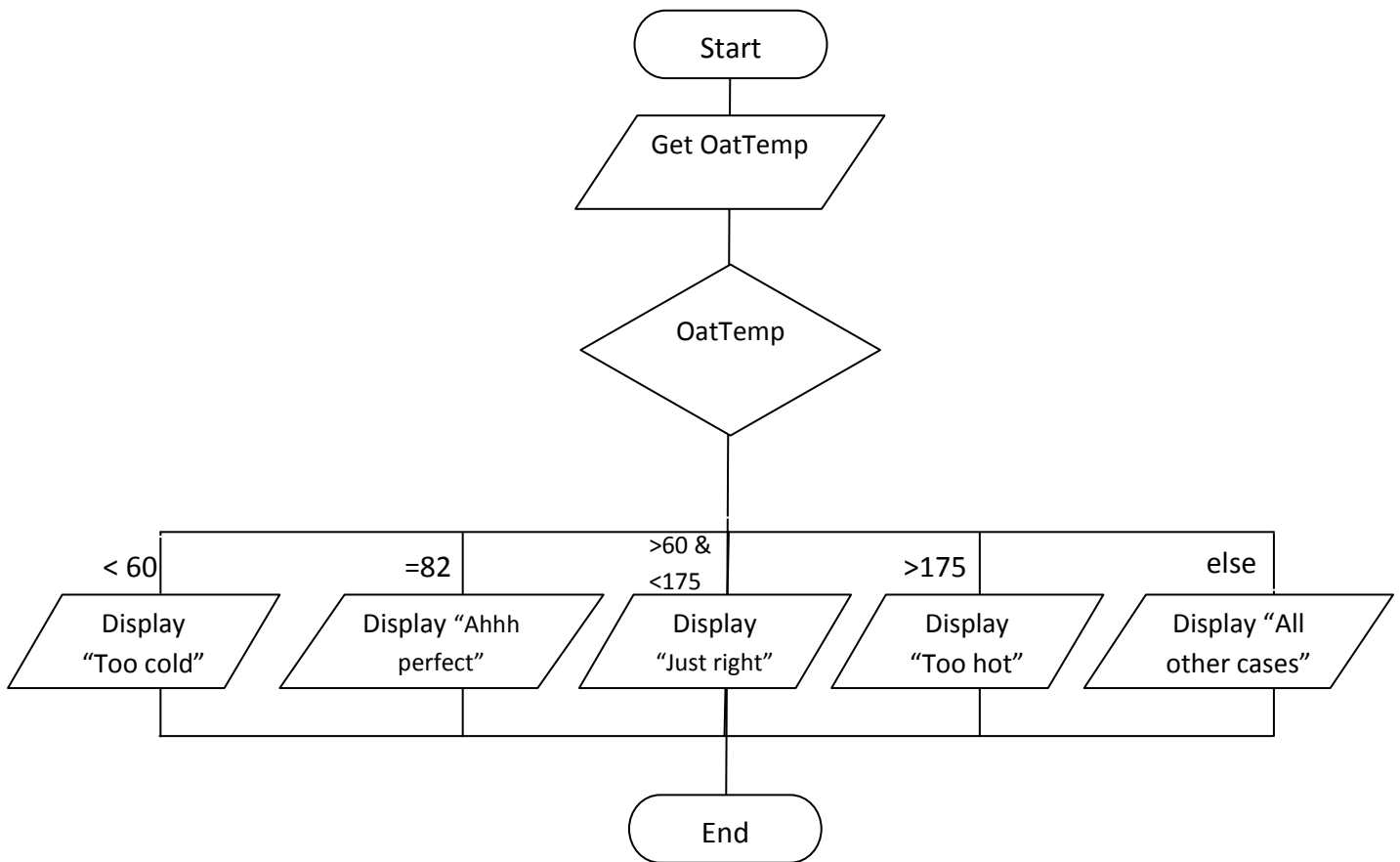
  OatTemp >= 175
    PRINT "Too hot"

  ELSE
    PRINT "This covers all over cases"

END CASE
```

There are a couple of ways to create flowcharts of CASE statements.





The first of these probably better illustrates what is happening from the computer's point of view, but the second seems easier to interpret.

Let's pretend we're writing a system to sort voters into four lines – they'll step up and give their last name and we'll tell them which line to get into.

```

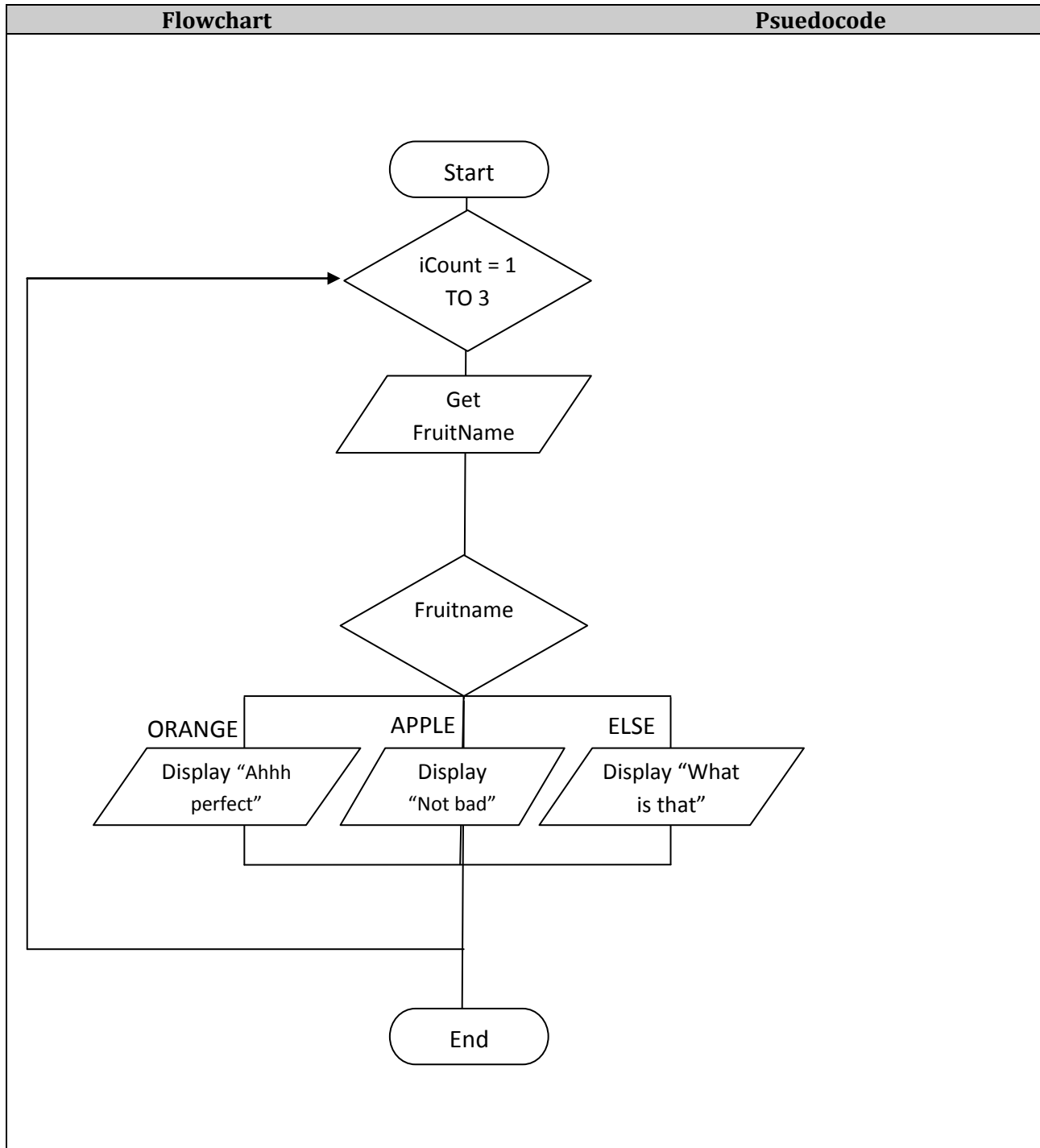
DECLARE LastName AS STRING
DECLARE LineNum AS INT
DECLARE FirstChar AS CHAR
//Get the first letter of their last name
FirstChar = LEFT(LastName,1)
//Assign lines
CASE
  FirstChar = A TO E
    PRINT "Please go to line 1"
  FirstChar = F TO J
    PRINT "Please go to line 2"
  FirstChar = K TO Q
    PRINT "Please go to line 3"
  ELSE
    PRINT "Please go to line 4"
END CASE
  
```

Exercise 2.13 Show the output of the following program and fill in the variable table.

Pseudocode	What the User Sees
<pre> DECLARE iCount AS INT FOR iCount = 1 to 4 CASE iCount = 1 PRINT "First time" iCount = 2 To 4 PRINT "Somewhere in the middle" iCount = 5 PRINT "Last time, bye" NEXT iCount1 </pre>	

Variable Name	Data Type	Description

Exercise 2.14 Write the psuedocode to produce the following output and fill in the variable table.



Lesson 7 – Decision Tables

```
<html>
<body>
<script type="text/javascript">

Purchase=prompt("Please enter the purchase price");
LastPay=prompt("Please enter number of days since last payment");
Balance=prompt("Please enter the account balance");
CredStat = ""

if (Purchase < 100)
{
  if (LastPay <30)
  {
    if (Balance<1000)
    {
      CredStat = "Credit OK"
    }
    else
    {
      CredStat = "Refer to credit Dept"
    }
  }
  else
  {
    if (Balance<1000)
    {
      CredStat = "Credit OK"
    }
    else
    {
      CredStat = "Credit denied"
    }
  }
}
else
{
  if (LastPay <30)
  {
    if (Balance<1000)
    {
      CredStat = "Refer to credit dept"
    }
  }
}
```

```
    else
    {
        CredStat = "Credit denied"
    }
}
else
{
    if (Balance<1000)
    {
        CredStat = "Refer to credit dept"
    }
    else
    {
        CredStat = "Credit denied"
    }
}
}
```

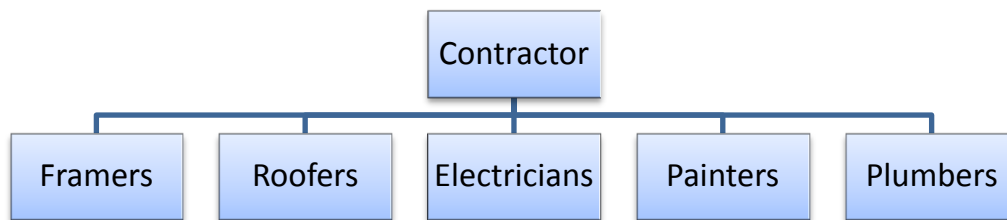
```
document.write(CredStat);
</script>
</body>
```

Module 3 - Modular Program Development

Lesson 1 - Terminology & Problem Organization

If we were building a new house, there would be a whole team of people all working together to accomplish the task. A contractor would be in charge and would direct the work of foundation guys, framers, roofers, electricians, plumbers, sheet rocks guys, painters and others.

A organizational chart of the crew might look like the following:



Each task is broken off and handled separately as directed by the Contractor. He makes sure all the other workers do their part and that everything happens in the right order.

We do the same thing in computer programming. As problems and programs become more complex, it helps the organization and solution if we break our problem into smaller pieces. The contractor is replaced by the “control program”; the subs are called modules.

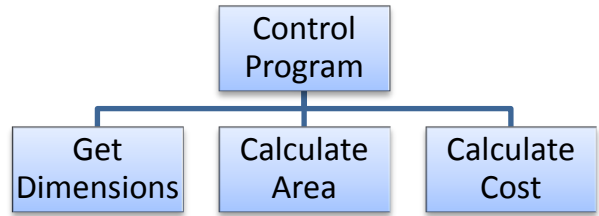
For example, a while back we wrote a program to calculate the cost to carpet a room. The individual steps were:

- Get the length of the room in feet
- Get the width of the room in feet
- Calculate the area of the room by multiplying length x width
- Convert this area to square yards by dividing by 9
- Get the cost of carpet per square yard
- Calculate the cost of the carpet by multiplying number of square yards x cost per square yard

Our psuedocode was

```
DECLARE RoomLength, RoomWidth, RoomArea, CostPerYard, Cost AS REAL
INPUT "Please input room length", RoomLength
INPUT "Please input room width", RoomWidth
RoomArea = RoomLength * RoomWidth
RoomArea = RoomArea /9
INPUT "Please input the cost per sq.yd", CostPerYd
Cost = RoomArea * CostPerYd
PRINT Cost
```

While this is a very simple program, we could regroup it into three modules as shown in the “module diagram” at the right:



- **Get the dimensions** of the room
 - Get the length of the room in feet
 - Get the width of the room in feet
- **Calculate the area** of the room
 - Multiply length x width
 - Convert this area to square yards by dividing by 9
- **Calculate the cost** of the carpet
 - Get the cost of carpet per square yard
 - Multiply number of square yards x cost per square yard

Cohesion - Each of these three modules need to be independent and be able to do its own part of the problem. This is referred to as “cohesion”. Since cohesion means “holds together”, Wikipedia’s definition makes sense – “In computer programming, cohesion is a measure of how strongly-related or focused the responsibilities of a single module are.” Each task should be tightly focused on a particular chore.


Coupling - You remember that we had several variables:

- RoomLength
- RoomWidth
- RoomArea
- CostPerYd
- Cost

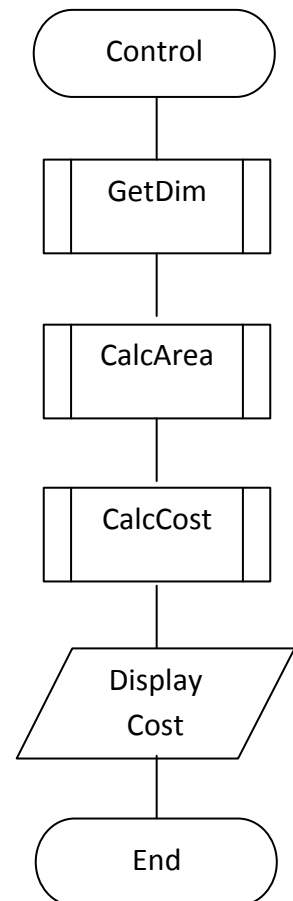
You can see that the Calculate Area Module will need to know the RoomLength and RoomWidth values from the Get Dimensions Module. Being able to pass these variables back and forth is referred to as “coupling”. Wikipedia puts it like this – “In computer science, **coupling** or **dependency** is the degree to which each program module relies on each one of the other modules.”

Using cohesion and coupling makes your program work and makes it easier to read and maintain.

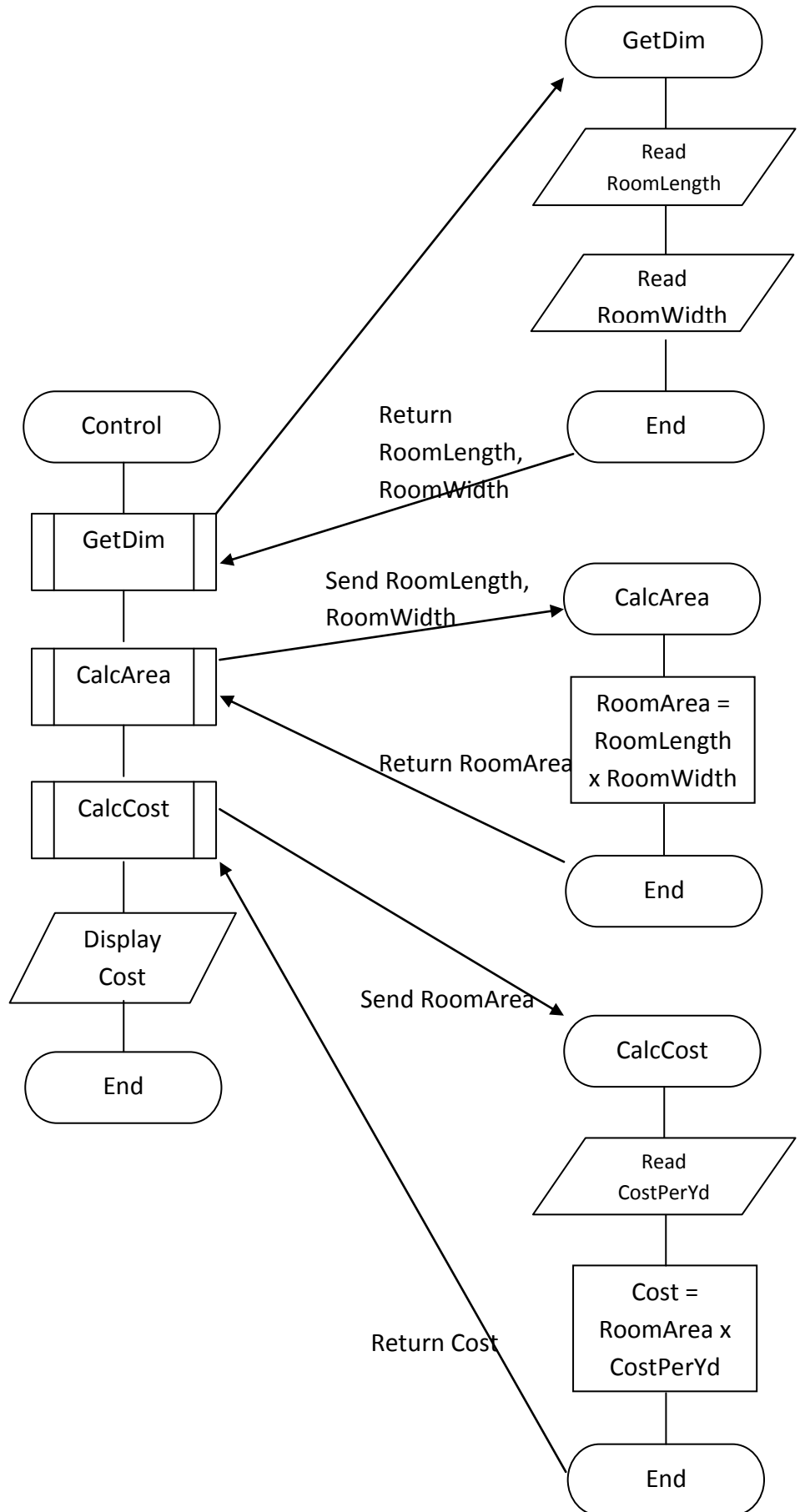
The flowchart symbol for a module is shown.

Flowchart Symbol	Purpose
	Module – a symbol within the Control flowchart to call a module

The control program would look like the figure at the left. You can see that Control first moves to GetDim, then calls CalcArea and CalcCost



before it finally displays the cost to the user, either on the screen or to a printer. Here you can see the control program and each of the modules. The lines running back and forth between control and the modules show the order, as well as what data needs to be passed.



The carpet cost pseudocode might look something like the following:

```
//Control Program
// 17 Mar 2009 – John Billingsworth

// Define Variables
DECLARE RoomLength as Real
DECLARE RoomWidth as Real
DECLARE RoomArea as Real
DECLARE CostPerYd as Real
DECLARE Cost as Real

//Call GetDim – return RoomLength, RoomWidth
CALL GetDim(RoomLength, RoomWidth)

//Call CalcArea – send RoomLength, RoomWidth, return RoomArea
CALL CalcArea(RoomLength, RoomWidth, RoomArea)

//Call CalcArea – send RoomArea return Cost
CALL CalcCost(RoomArea, CostPerYd, Cost)
PRINT Cost
END

*****
GetDim(RoomLength, RoomWidth)
//Get the Room size
INPUT “What is the room Length”, RoomLength
INPUT “What is the room Width”, RoomWidth
RETURN RoomLength, RoomWidth

*****
CalcArea(RoomLength, RoomWidth, RoomArea)
//Calc the area in sq.yds
RoomArea = RoomLength * RoomWidth/9
RETURN RoomArea

*****
CalcCost(RoomArea, CostPerYd, Cost)
INPUT “What is the cost per sq.yd”, CostPerYd
Cost = RoomArea * CostPerYd
RETURN Cost
```

While this seems like more work, it makes bigger problems much easier to solve. Additionally, you can often “recycle” code – a module can contain code that does the same thing to lots of sets of data without programming each one. Simply recall the module and pass the new data to it.

In JavaScript, there are several ways to do this – the easiest is probably the one you see here:

```
<html>
<head>
<script type="text/javascript">

function GetDim(){
RoomLength=prompt("Please enter the Room Length");
RoomWidth=prompt("Please enter the Room Width");
}

function CalcArea(){
RoomArea = RoomWidth*RoomLength;
document.write("The room area is ",RoomArea, " square feet<p>");
RoomArea=RoomArea/9;
document.write("The room area is ",RoomArea, " square yards <p>");
}

function CalcCost(){
CostPerYd=prompt("Please enter the cost per sq.yd");
Cost= CostPerYd*RoomArea;
}

</script>
</head>

<body>
<script type="text/javascript">
GetDim()
CalcArea()
CalcCost()
document.write("The cost for your carpet is ",Cost)
</script>
</body>
</html>
```

3 modules (or functions) in <head> instead of <body>

Call 3 modules (or functions)

As another example, let's write a program to convert money from one currency to another. This program has a WHILE loop that will do as many currency conversion as you need to do – to exit, simply put a zero for the amount of money to be converted.

```

//Currency Converter
//Lee Craig – 10 Feb 2011

// Define Variables
DECLARE StartMoney, EndMoney as Real
DECLARE StartCurrency, EndCurrency, ConvFactor as Real

StartMoney = 1
WHILE StartMoney > 0
    INPUT "How much money would you like to convert?", StartMoney
    IF StartMoney = 0
        END
    END IF
    INPUT "Initial currency? (d=dollars, e=euros, p=pesos)", StartCurrency
    INPUT "Final currency? (d=dollars, e=euros, p=pesos)", EndCurrency
    CALL ChangeMoney(StartMoney, StartCurrency, EndCurrency)
WHILE END
END

//*****
ChangeMoney(StartMoney, StartCurrency, EndCurrency)
CASE
    StartCurrency = "d" AND EndCurrency = "e"
        ConvFactor = 0.7330

    StartCurrency = "d" AND EndCurrency = "p"
        ConvFactor = 12.10

    StartCurrency = "e" AND EndCurrency = "d"
        ConvFactor = 1.36

    StartCurrency = "e" AND EndCurrency = "p"
        ConvFactor = 16.47

    StartCurrency = "p" AND EndCurrency = "d"
        ConvFactor = 0.08

    StartCurrency = "p" AND EndCurrency = "e"
        ConvFactor = 0.06

    ELSE
        PRINT "Please try again"
        RETURN
END CASE
EndMoney = StartMoney * ConvFactor
PRINT StartMoney, StartCurrency, " = ", EndMoney, EndCurrency
RETURN

```

Exercise 3.1 Draw a module diagram and a flowchart of the Currency Converter program we just did.

Exercise 3.2 Give the output of the following code if the user inputs “GrapeJello” when prompted. Draw a module diagram below.

Psuedocode	What the User Sees
<pre> //Control Program DECLARE FavFood AS STRING DECLARE NumChars, CharCTR as INT INPUT "What's your favorite food?", FavFood NumChars = LENGTH(FavFood) FOR CharCTR = 1 to NumChars CALL NextChar(FavFood, CharCTR, NumChars) NEXT CharCTR LastTime() END //***** NextChar(FavFood, CharCTR, NumChars) DECLARE ThisChar AS CHAR IF CharCTR = 1 THEN CALL FirstTime() ENDIF ThisChar = MID(UPPER(FavFood), NumChars-CharCTR+1,1) PRINT ThisChar RETURN //***** FirstTime() PRINT "I bet you really like " RETURN //***** LastTime() PRINT " don't you?" RETURN </pre>	

Exercise 3.3 Write a calculator program that asks the user for two numbers and whether to add or multiply the two numbers. The addition or multiplication should be done by separate modules. Show the output for the numbers 4, 17 and multiply. Draw a module diagram below.

Psuedocode	What the User Sees
<pre> //Control Program //***** AddNum() RETURN //***** MultNum() RETURN </pre>	

Lesson 2 - Scope

In the last lesson, when we wrote the JavaScript program using modules to calculate the cost of carpet in a room, all of the variables could be seen by the control program, as well as each of the modules. Variables that can be seen everywhere, like these, are referred to as “**global variables**”. On the other hand, if a variable is visible and usable only to a particular module, it is called a “**local variable**”. The concept of **scope** describes whether a variable is local or global.

As someone once said, “The advantages of a global variable are easy: You can access it from anywhere in your program. The disadvantages are also easy: You can access it from anywhere in your program.”

Let’s say we have a large, involved program with 216 modules which can each be called from the control program, or from the other modules. Two of these modules have a variable called `LogInState`. In one of the modules this variable is supposed to track whether a user has successfully logged in, but the other is determining which US state a user is logged in from. Declaring `LogInState` as a global variable means that one module may overwrite the variable with “Nebraska” instead of “Valid” – this can clearly lead to all sorts of messes.

On the other hand, if a user logs in to a system, and his username needs to be available throughout the program, a global variable may be just the thing. Most programmers agree that nearly all variables should be local variables, the exception being “constants”. This forces you, as the programmer, to pass variables back and forth – but it also guarantees that you have control over the data, and therefore the logic of the program. The extra work is worth the effort if it means you know what you’re dealing with.

To declare `NumCars` as global, simply put
GLOBAL DECLARE `NumCars` AS INT

To declare `NumCars` as a local variable only visible to the code it’s within, simply put
DECLARE `NumCars` AS INT

```
<html>
<head>
<script type="text/javascript">

function GetDim(){
RoomLength=prompt("Please enter the Room Length");
RoomWidth=prompt("Please enter the Room Width");
}

function CalcArea(){
RoomArea = RoomWidth*RoomLength;
document.write("The room area is ",RoomArea, " square feet<p>");
RoomArea=RoomArea/9;
document.write("The room area is ",RoomArea, " square yards <p>");
}

function CalcCost(){
CostPerYd=prompt("Please enter the cost per sq.yd");
Cost= CostPerYd*RoomArea;
}

</script>
</head>

<body>
<script type="text/javascript">
GetDim()
CalcArea()
CalcCost()
document.write("The cost for your carpet is ",Cost)
</script>
</body>
</html>
```

To call a module called AddPhone that uses global variables, since the module can already see and use all of the variables, write

```
CALL AddPhone()
```

However, to call a module called AddPhone that uses local variables you must pass those variables to the function. Some languages give you a choice of two methods that we refer to as

- **“Call by value”** - sends the value of a variable to a module, but the module cannot overwrite the original variable. For example, if a variable iOutTemp = 63 is passed by value, the number 63 would be sent to the module.

```
CALL AddPhone(iNumUsers, NewPhoneNum)
```

- **“Call by reference (or address)”** – sends the memory address of the variable to the module, and the module can overwrite the original variable. For example, if a variable iOutTemp = 63 is passed by reference, the computer memory address of the original variable would be sent to the module and it could read, use, overwrite that value. Adding an ampersand accomplishes this.

```
CALL AddPhone(&iNumUsers, &NewPhoneNum)
```

Anytime variables are passed, the module needs to be ready to “catch” what is tossed to it. The function name must include the same number of variables and the same type of variables to receive these variables. The variables names in the modules may be the same or may be different as the original variables.

```
AddPhone(iNumUsers, NewPhoneNum)
```

Exercise 3.4 List the variables that are global and those local to each module.

Pseudocode	Variables
<pre>//Control Program GLOBAL DECLARE FavFood AS STRING DECLARE NumChars, CharCTR as INT INPUT "What's your favorite food?", FavFood NumChars = LENGTH(FavFood) FOR CharCTR = 1 to NumChars CALL NextChar(CharCTR, NumChars) NEXT CharCTR LastTime() END</pre>	<p>Global</p> <p>Local to Control</p>
<pre>//***** NextChar(CharCTR, NumChars) DECLARE ThisChar AS CHAR IF CharCTR = 1 THEN CALL FirstTime() ENDIF ThisChar = MID(UPPER(FavFood), NumChars-CharCTR+1,1) PRINT ThisChar RETURN</pre>	<p>Local to NextChar</p> <p>Local to FirstTime</p>
<pre>//***** FirstTime() PRINT "I bet you really like " RETURN //***** LastTime() PRINT " don't you?" RETURN</pre>	<p>Local to LastTime</p>

Lesson 3 – Bullet Proofing



It is the responsibility of the programmer to try to anticipate and prevent everything that can go wrong in the program.



When I did programming for control/monitoring systems for textile mills, if a program crashed and couldn't be restarted, I would get a call in the middle of the night and have to drive a couple hundred miles to the mill and stay there until it was fixed. Sometimes, it was an unexpected input or a poorly entered piece of data. Sometimes it was a hardware problem. Sometimes a couple of the mill workers, having nothing but a loom to stare at for twelve hour shifts thought it was more fun to fiddle with the computer, and see if they could crash it. Our motto was "Trust no one!" Preventing crashes can literally help get you a good night's sleep!



For example, in module 2, lesson 3 we wrote this program for billing doctor's appointments:

```
DECLARE Fee, AmtDue AS REAL
DECLARE InsYesNo AS LOGICAL
INPUT "Please input today's fee", Fee
INPUT "Does the patient have insurance?" ,InsYesNo
IF InsYesNo = 1 THEN
    AmtDue = .20*Fee
    PRINT "Today's fee is $",AmtDue
ELSE // InsYesNo = 0
    AmtDue = Fee
    INPUT "Is the patient paying by credit card or cash", PayMeth
    IF PayMeth = "card" THEN
        INPUT "Mastercard or VISA?",CredCard
        PRINT "$",AmtDue, " has been charged to ", CredCard
    ELSE //pay Cash
        PRINT "Thank you for paying $", AmtDue, " by cash"
    END IF
END IF
```

Here there are a number of places where a distracted (or malicious) secretary might type in the wrong thing and crash your program. For example, two lines above read

```

INPUT "Is the patient paying by credit card or cash"
IF PayMeth = "card" THEN

```

What happens if someone types in "CC" or "Card" or "CARD" or "756" instead of "card"?

Here's a safer, more ironclad version of the same program, it tries to guess what the user means by an input, including

- Making the input all caps (or all lower case) to prevent "Card", "CARD" or "card" from being read as different responses. The UPPER() function makes it all caps.
- Prompting the user to enter a specific input – for example

```
INPUT "Does the patient have insurance? (y or n)", InsuranceStatus
```
- Accepting several responses as valid – these lines:

```
InsurStat = UPPER(LEFT(STRING(InsuranceStatus),1))
IF InsurStat = "Y" OR InsurStat = "1"
```

accept any of the following input of YES, Yes, yes, yEs, Y,y, ye, YE, etc or 1 as the same thing and all evaluate as true in the IF statement.

```

DECLARE Fee, AmtDue AS REAL
DECLARE InsuranceStatus AS STRING
DECLARE InsurStat AS CHAR
INPUT "Please input today's fee", Fee

//Check to see if patient has insurance
INPUT "Does the patient have insurance? (y or n)", InsuranceStatus
//Make sure InsuranceStatus is a string, Get the first letter of the string,
//make it upper case
InsurStat = UPPER(LEFT(STRING(InsuranceStatus),1))
IF InsurStat = "Y" OR InsurStat = "1"
    AmtDue = .20*Fee
    PRINT "Today's fee is $",AmtDue
ELSE // InsYesNo = 0
    AmtDue = Fee
    INPUT "Is the patient paying by credit card or cash", PayMeth
    PayMeth = UPPER(LEFT(STRING(PayMeth),2))
    IF PayMeth = "CR" OR PayMeth = "CC" THEN
        INPUT "Mastercard or VISA?",CredCard
        PRINT "$",AmtDue, " has been charged to ", CredCard
    ELSE //pay Cash
        PRINT "Thank you for paying $", AmtDue, " by cash"
    END IF
END IF
END IF

```

But there are still places that bad data could sneak in. The old computer adage “Garbage In = Garbage Out” holds here. Prompting the user for a specific input will help but in practice, **ALL information you get from the user should be verified by the program.** If that data is not right or not understood, your program should ask the user again or take another path that does not jeopardize the integrity of the data, the logic or the output. This strengthened portion of the code uses a loop to **force the user to enter acceptable data** regarding their insurance status:

```
//Check to see if patient has insurance
DECLARE InsOK = 0
WHILE InsOK = 0
    INPUT “Does the patient have insurance? (y or n)” , InsuranceStatus
    //Make sure InsuranceStatus is a string, Get the first letter of the string,
    //make it upper case
    InsurStat = UPPER(LEFT(STRING(InsuranceStatus),1))
    IF InsurStat = “Y” OR InsurStat = “1” OR InsurStat = “N” OR InsurStat = “0”
        InsOK = 1
    END IF
WHILE END

IF InsurStat = “Y” OR InsurStat = “1”
    AmtDue = .20*Fee
    PRINT “Today’s fee is $”,AmtDue
ELSE    // InsYesNo = 0
    AmtDue = Fee
    INPUT “Is the patient paying by credit card or cash”, PayMeth
    PayMeth = UPPER(LEFT(STRING(PayMeth),2))
    IF PayMeth = “CR” OR PayMeth = “CC” THEN
        INPUT “Mastercard or VISA?”,CredCard
        PRINT “$”,AmtDue, “ has been charged to ”, CredCard
    ELSE    //pay Cash
        PRINT “Thank you for paying $”, AmtDue, “ by cash”
    END IF
END IF
```

This is a lot of extra work, but it means that you have control of the data that your software is using, and that you and your customer can rely on that software.

Exercise 3.5 Circle all the places that need to be “bulletproofed”. Briefly describe the changes that would make the data and the run “safer”.

Psuedocode
<pre>//Control Program // 17 Mar 2009 – John Billingsworth // Define Variables DECLARE RoomLength as Real DECLARE RoomWidth as Real DECLARE RoomArea as Real DECLARE CostPerYd as Real DECLARE Cost as Real CALL GetDim(RoomLength, RoomWidth) CALL CalcArea(RoomLength, RoomWidth, RoomArea) CALL CalcCost(RoomArea, CostPerYd, Cost) PRINT Cost END ***** GetDim(RoomLength, RoomWidth) //Get the Room size INPUT “What is the room Length”, RoomLength INPUT “What is the room Width”, RoomWidth RETURN RoomLength, RoomWidth ***** CalcArea(RoomLength, RoomWidth, RoomArea) //Calc the area in sq.yds RoomArea = RoomLength * RoomWidth/9 RETURN RoomArea ***** CalcCost(RoomArea, CostPerYd, Cost) INPUT “What is the cost per sq.yd”, CostPerYd Cost = RoomArea * CostPerYd RETURN Cost</pre>

Lesson 4 – Documentation

There are three types of documentation, two of which we have already seen and used extensively:

- “**Internal**” documentation – notes to ourselves inside the code that only the programmer(s) can see, but that the computer ignores.
- “**External**” documentation – paper forms, tables, flowcharts, variable lists that help the programmer(s) to plan and write the code.
- “**User**” documentation – information that is provided to the user to help them use the software.

Let’s think about each of these.

Internal documentation is easy to build in and so very important to guide the programmer in both developing the software and especially months or years later to fix and update the software.

I once started working at a software company and inherited the development and maintenance of a large and complicated windows-based database system that tracked the life history of rolls of cloth made in a textile factory: the yardage, when it was produced, if it was joined with other rolls or split into new rolls and where all the imperfections were as the roll was stretched. There were thousands of pages of code, hundreds of variables, dozens of different reports the users could run- it was a mess. But having access to the thoughts of the original programmer AS HE WROTE the software was invaluable – why this variable was a certain type, or what a particular report did.

We already know that a comment is as easy to put in as simply typing
// this report gives the yardage of rolls from longest to shortest
// only visible to management and warehouse

Most software companies will have a basic layout for comments beginning a piece of code that tells what the code does, who wrote it, when it was written and other useful information, for example:

```
//*****  
//*****  
//*** STRETCHING YARDAGE REPORT  
//*** gives the new yardage of faults in a stretched roll  
// *** written by Tom Josefson  
// *** 17 April 1994  
//*****  
//*** updated 4 May 1997 by Sue Glasco  
//*** added graphing functionality
```

Since the computer doesn't read the comments, put in everything you think will be useful down the road. You never know who else will have to fix your stuff, it might be you or someone else but years go by quickly and so does your memory of why you wrote the code a particular way.

External documentation is important as you plan and develop software. It is also invaluable years later as it may give lists of all the variables, or database fields, or how the data is used. The external information we have seen and used includes the following:

- Description of problem
- List of steps
- List of variable names and types
- Flowchart
- Psuedocode

User documentation is important and has become a whole industry in itself. There are three main sources of documentation o help learn and use software – books, the internet, and help menus built into the software.

Thousands of books are written every year helping users learn the ins and outs of different software packages. The people that produce these guides are referred to as technical writers, and there's a lot of good money in technical writing. Help books are usually logically organized and well illustrated with screen shots and tables.



Internet websites run by software producers serve the same purpose as the books, but the people involved must be technical writers and have good web publishing skills.

Most software packages also have a help menu that was built contemporaneously with the software. A “Help” item on the toolbar, or sometimes pressing F1 activates the help menu. Generally these are searchable by keyword, and have topics listed in an index. Sometimes there is also a list of FAQ – Frequently Asked Questions to make finding the answer to a particular question easier. Help menus can usually be built in and by the same “development environment” that was used to produce the software.

Whatever the delivery format, the Help documentation must be prepared by someone who is intimately familiar with the software, is well organized and thinks objectively and logically – sometimes that means you, the programmer. In the database program mentioned at the beginning of this lesson, in addition to performing all of the maintenance and updating of the software, I was also responsible for writing a system help menu and a 30 page booklet about how to use the software. Be prepared!

Lesson 5 – Introduction to Object Oriented Programming/Visual BASIC

Developing software for Windows based systems has changed the way programs are structured. In the programs we have looked at so far, a user is forced to take a single path through the logic. They might be asked for several inputs and the program takes care of everything else by following the sequential programming in the code. Loops, decisions and modules all occur, but in a very predictable manner.

In a Windows environment, the user is faced with a form that allows much more flexibility. This form is referred to as a “**Graphical User Interface**” or **GUI**. Each item has its own responsibilities and powers derived from the code that is written into each component, or object, of the program. **Essentially, each object is its own module**, independent of the other controls. This type of programming is called “**Object Oriented Programming**”.

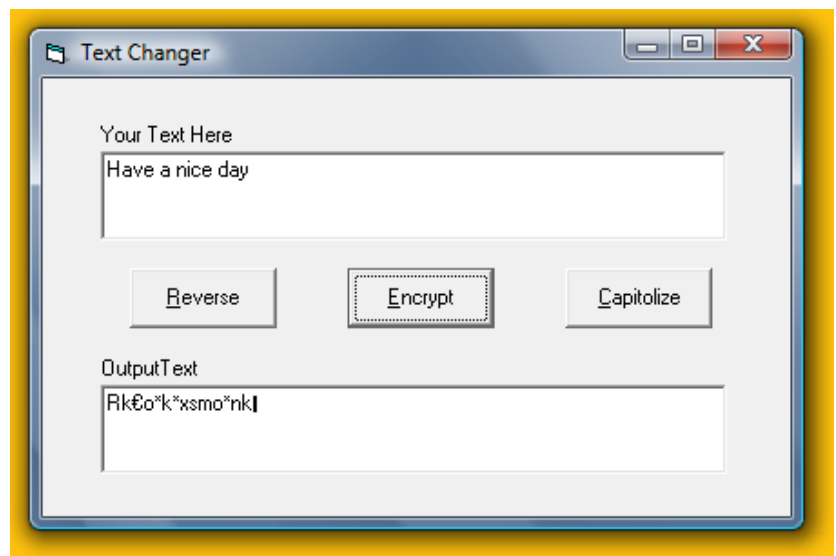
As an example, here is a program that was written in Visual BASIC. The user is asked for a string. In this case they input “Have a nice day”. Depending on which button they push the program outputs

- yad ecin a evaH
- Rk€o*k*xsmo*nkf
- HAVE A NICE DAY

Here is the code for each of the three buttons:

- Reverse Button

```
Private Sub cmdReverse_Click()  
    strText = Text1.Text  
    strNewText = ""  
    iLengthText = Len(strText)  
    For iCTR = iLengthText To 1 Step -1  
        cNextChar = Mid(strText, iCTR, 1)  
        strNewText = strNewText + cNextChar  
    Next iCTR  
    Text2.Text = strNewText  
End Sub
```



- Encrypt Button

```

Private Sub cmdEncrypt_Click()
strText = Text1.Text
strNewText = ""
iLengthText = Len(strText)
For iCTR = 1 To iLengthText
    cNextChar = Mid(strText, iCTR, 1)
    iNextChar = Asc(cNextChar) + 10
    cEncryptNextChar = Chr(iNextChar)
    strNewText = strNewText + cEncryptNextChar
Next iCTR
Text2.Text = strNewText
End Sub

```

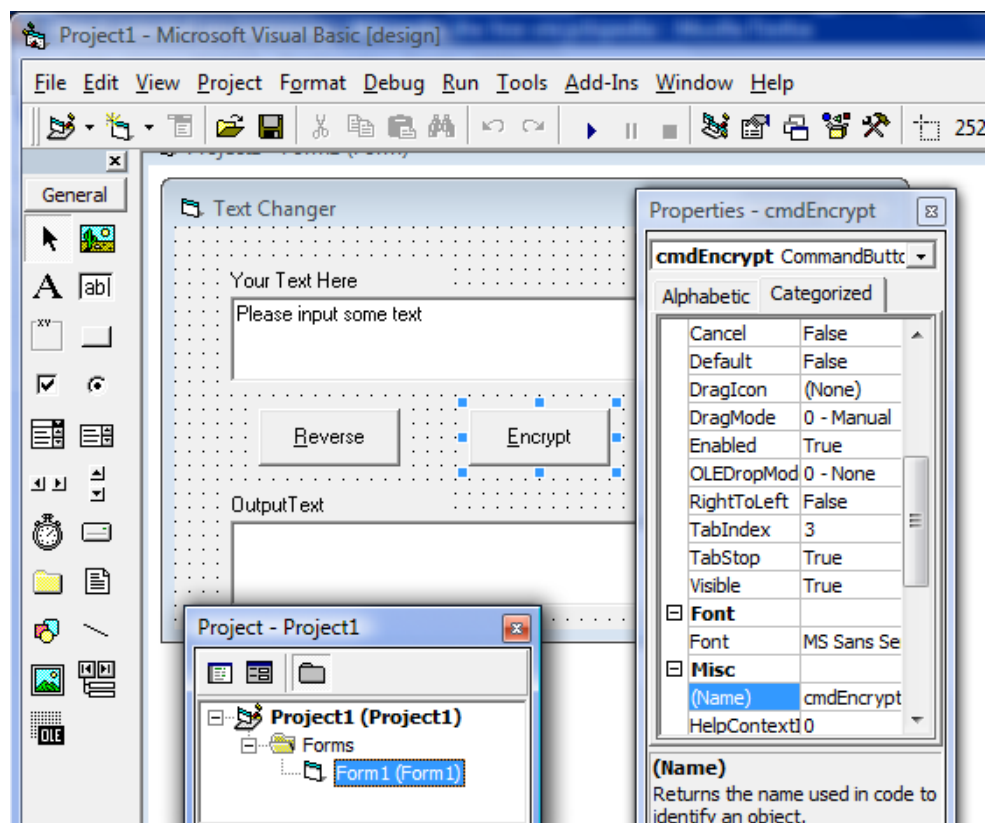
- Capitalize Button

```

Private Sub cmdCapitol_Click()
strText = Text1.Text
strNewText = UCase(strText)
Text2.Text = strNewText
End Sub

```

To build the GUI, there is a “Development Environment” that has a toolbar with the different controls that can be dropped onto the form. Each object has a name and a list of properties that can be changed by the programmer. Code is entered by double clicking the control.



Module 4 – Arrays & File Access

Lesson 1- Arrays & Parallel Arrays

Variables hold a single piece of data. Sometimes, however, we wish to store several pieces of related data – for example the phone numbers of all the students in the class. For this purpose we use an **array**, which can be thought of as a block of variables. The array name is the same for all the pieces of data, and a number is used to keep track of which particular piece we want to work with. This number is referred to as an **index**. Since most languages start numbering at zero, the index of each **element** an array with five boxes would be 0,1,2,3,4.

In this example there are five phone numbers stored in an array called PhoneNum. Asking the computer to read PhoneNum(3) returns “984-2356”. To write to an array, simply put

PhoneNum(3) = “982-7352”

PhoneNum(0)	981-2566
PhoneNum(1)	407-5589
PhoneNum(2)	287-4587
PhoneNum(3)	984-2356
PhoneNum(4)	981-6263

and it will replace the existing data stored there with the new phone number.

If you need to keep up with two (or more) sets of data that are interrelated, an array for each set of data is constructed, and the pair of arrays is referred to as a **parallel array**. Here the names and phone numbers of five individuals are stored in two arrays.

PhoneNum(0)	981-2566	LastName(0)	Jones
PhoneNum(1)	407-5589	LastName (1)	Smith
PhoneNum(2)	287-4587	LastName (2)	Barber
PhoneNum(3)	984-2356	LastName (3)	Wilson
PhoneNum(4)	981-6263	LastName (4)	Ericson

If you want information about a particular individual, use the same index for both arrays.

Pseudocode	User Sees/Enters
//Show the name and phone # INPUT “Pick number from 0 to 4”, NumPick PRINT “Name is”, LastName(NumPick) PRINT “Phone is”, PhoneNum(NumPick)	Pick number from 0 to 4 >3 Name is Wilson Phone is 984-2356

To declare an array, it’s almost the same as declaring a variable. The command

DECLARE PhoneNum(5) AS INTEGER

would set up an array with five elements, each of which could hold an integer. All other data types can also be stored in arrays.

You can setup up as many elements in an array as you want. In **static** arrays the array size, or **dimension**, cannot be changed once declared. In **dynamic** arrays, you can increase or decrease the number of array elements after the array is declared, as the program is running. Some programming languages allow dynamic arrays, others only use static arrays. If you are using a static array, over guessing the size you need is better than running out of room.

To fill arrays we could use a FOR loop.

Pseudocode	User Sees/Enters	Array					
<pre>//Ask a user for five names and write them to an array DECLARE FiveNames(5) AS STRING FOR iCTR = 0 to 4 INPUT "Name ", iCTR," = ", Name FiveNames(iCTR) = Name NEXT iCTR</pre>	<pre>Name 0 = >Joe Name 1 = >Sam Name 2 = >Bill Name 3 = >Bob Name 4 = >Kelly</pre>	<pre>FiveNames(0) FiveNames(1) FiveNames(2)</pre> <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>Joe</td></tr> <tr><td>Sam</td></tr> <tr><td>Bill</td></tr> <tr><td>Bob</td></tr> <tr><td>Kelly</td></tr> </table>	Joe	Sam	Bill	Bob	Kelly
Joe							
Sam							
Bill							
Bob							
Kelly							

Later, we could also use a FOR loop to display the values from that same array.

Pseudocode	User Sees
<pre>//Print five names from the previous array PRINT "The array contains:" FOR iCTR = 0 to 4 PRINT FiveNames(iCTR) NEXT iCTR</pre>	<pre>The array contains: Joe Sam Bill Bob Kelly</pre>

Warning: Since an array is essentially a block of variables, it no longer exists when the program finishes running. If you want to “permanently” store the data, an array is no good – a data file or database is what you want. These will be discussed later.

Exercise 4.1 Draw the array that gets created and populated and show the output of program.

Pseudocode	Array	Output
<pre>DECLARE FavColor(4) AS STRING FavColor(0) = "green" FavColor(1) = "pink" FavColor(2) = "red" FavColor(3) = "blue" PRINT FavColor(2)</pre>		

Exercise 4.2 Write psuedocode to collect two pieces of information on 30 students. The first piece of information will be student name and should get stored in an array called StudNames(). The other piece of information is the number of classes each student is taking and this gets stored in StudNumClass(). Use a FOR loop to get the user to INPUT the data. Then read back through the data and PRINT the names of students taking more than 4 classes.

Psuedocode

Lesson 2 - Double Scripted Arrays

In the first lesson we stored the names and phone numbers of five individuals in two arrays.

PhoneNum(0)	981-2566	LastName(0)	Jones
PhoneNum(1)	407-5589	LastName (1)	Smith
PhoneNum(2)	287-4587	LastName (2)	Barber
PhoneNum(3)	984-2356	LastName (3)	Wilson
PhoneNum(4)	981-6263	LastName (4)	Ericson

Another option is to create a two dimensional array, with two columns and five rows. We would need two index numbers to keep up with where each piece of data is located. The indices give the row number followed by the column number. Here the same data is stored in a two dimensional array called PhoneData.

Below you can see they addressing of the two dimensional array.

PhoneData(0,0)	PhoneData(0,1)
PhoneData(1,0)	PhoneData(1,1)
PhoneData(2,0)	PhoneData(2,1)
PhoneData(3,0)	PhoneData(3,1)
PhoneData(4,0)	PhoneData(4,1)

Jones	981-2566
Smith	407-5589
Barber	287-4587
Wilson	984-2356
Ericson	981-6263

To declare a two dimensional array put
DECLARE PhoneData(5,2) AS STRING

To read the data in the fourth row and first column into a variable called NewName
NewName = PhoneData(3,0)

To change Mr. Barber's phone number we would write
PhoneData(2,1) = "684-5233"

We could have a two dimensional array that was much larger than 5x2 – you can easily imagine a set of data 600 rows by 42 columns. It is also possible however to create and use arrays that are three or more dimensional. Multi dimensional arrays are hard to picture, but imagine a data set that contains medical information on all the patients in a hospital. Perhaps the rows represent individual patients, the first column stores their name, the second column their age, the third column might be the physicians name, etc. “Behind that” we might have a record of their blood pressure as checked four times a day, their preferences for meals and of course the expenses incurred. Each of these is information going off in another “direction” or dimension. These five dimensions are impossible to draw a picture of, and difficult to even imagine, but are all critical to proper data management. Here's the “front” of the data set – you'll have to figure out where the other three directions go!

	Name	Age	Physician	Room#
Row for each patient	(0,0,0,0,0)	(0,1,0,0,0)	(0,2,0,0,0)	(0,3,0,0,0)
	(1,0,0,0,0)	(1,1,0,0,0)	(1,2,0,0,0)	(1,3,0,0,0)
	(2,0,0,0,0)	(2,1,0,0,0)	(2,2,0,0,0)	(2,3,0,0,0)
	(3,0,0,0,0)	(3,1,0,0,0)	(3,2,0,0,0)	(3,3,0,0,0)
	(4,0,0,0,0)	(4,1,0,0,0)	(4,2,0,0,0)	(4,3,0,0,0)
	(5,0,0,0,0)	(5,1,0,0,0)	(5,2,0,0,0)	(5,3,0,0,0)

Exercise 4.3 Rewrite the code for Exercise 4.2 so that it uses a two-dimensional array instead of two one dimensional arrays. At the same time, break the program into a Main program that CALLs two modules called FillArrays and HeavyClassLoad. FillArray uses a FOR loop to get the user to INPUT and store the data. HeavyClassLoad reads the array and PRINTS the names of all students taking more than 4 classes.

Pseudocode

Exercise 4.4 Write a program that will create a double-scripted array that holds the name and 5 test grades for 20 students. Write a module that will determine the number of students who did better on the second test than they did on their first test. Draw the array that gets created and populated.

Pseudocode	Array

Exercise 4.5 Write a program for a small local business that prompts the user to enter the number of customers served for each day of a month (hold the input in an array). Write a second module that determines the number of days the business served 0-4, 5-9, 10-14, 15-19, 20-24, 25-29, or 30-34 customers (hold these values in a second 7 element array).

Pseudocode	Array

Lesson 3- Pointer Technique & Searching Arrays

When information is stored in arrays, it is important to be able to find, retrieve and use that data. There are different ways to search an array to find the desired information.

At the right is the array of names and phone numbers from the last lesson. Let's say we need to retrieve Ms. Ericson's phone number. The only problem is we don't which line her information is on.

Jones	981-2566
Smith	407-5589
Barber	287-4587
Wilson	984-2356
Ericson	981-6263

The easiest thing is to start at the top and read each name in the first column until we find "Ericson". This is called a **Linear Search Algorithm**. Once we find the right row, then, using the same index number, we could get the phone number. Here's a piece of pseudocode that could be used for this. Assume the array is called PhoneData and we know the dimensions of the array to be 5x2.

```
//set up a variable to store the index #, but not a valid array address
DECLARE iIndex = -1
FOR iCTR = 0 to 4
    IF PhoneData(iCTR,0) = "Ericson"
        iIndex = iCTR
        //if you find Ericson, print the phone #
        PRINT PhoneData(iIndex,1)
    END
ENDIF
NEXT iCTR

IF iIndex = -1
    PRINT "Record not found"
END IF
```

This works great, unless the array is pretty big. Then you have to search every line of the array until you find the desired record, which may take a lot of time and computer memory.

Using a **Binary Search Algorithm** is faster and more efficient, but requires that the information be sorted alphabetically or numerically, depending on the data type.

To do a Binary Search we check the middle record first. If this isn't what we want, we then decide if we need to look in the first or second half of the array – that is, did we go too far or not far enough. Go to the middle of the half that has the needed data and check that record. If that isn't the desired record we keep jumping to the middle of the remainder of the array that has our information. Effectively we cut the array in half with every search.

Barber	287-4587
Ericson	981-6263
Jones	981-2566
Smith	407-5589
Wilson	984-2356

Here's the same data sorted alphabetically by name. To do a binary search we jump to the middle which in this case is "Jones". Since "Jones" is after "Ericson" we went too far and need to concentrate on the first half of the array. Next, go to the middle of the first half of the array, and the record reads "Ericson". We're done.

In this contrived example, we got the right record in only two steps when the linear search took five, but the efficiency of the system is rock solid. In an array of 10,000 records, a linear search could very well require up to 10,000 searches whereas using a binary search guarantees that you will have your information within 14 searches. With a million records, a linear search algorithm on average takes 500,000 searches, while the binary search requires 20 or less.

Here's an example of how to do a binary search.

```
//Array is called YourArray()
//FirstPt is the index of the first element of the piece we want to search
//LastPt is the index of the last element of the piece we want to search
//MidPt is the index of the element in the middle - the one we check
//SearchItem is the value we are searching for

FirstPt = 0
LastPt = ArraySize

WHILE FirstPt <= LastPt
    // compute midpoint and get the integer part
    MidPt = INT((FirstPt + LastPt) / 2)
    CASE
        SearchItem < YourArray[MidPt]
            // repeat search in first half.
            LastPt = MidPt - 1
        SearchItem > YourArray [MidPt]
            // repeat search in second half.
            FirstPt = MidPt + 1
    ELSE
        // found it. return position
        RETURN MidPt
    END CASE
WHILE END
```

Another technique to increase the speed of finding data is to use **pointers**. A pointer is a variable type that simply stores a memory address. You could also create an array of pointers each of which points to a memory location. This way, you don't have to look up anything, you just follow the pointer. This is used extensively in the C programming languages and some others.

To define and use a pointer, first define a variable and then create a pointer to it.

```
DECLARE NumBooks AS INT //create a integer variable
DECLARE * PointNumBooks // pointer to that variable, the * indicates pointer

NumBooks = 47 //give NumBooks a value
PointNumBooks = & NumBooks //point PointNumBooks to NumBooks

PRINT NumBooks //prints the number 47
PRINT *PointNumBooks //”dereferences” or follows the address to
//NumBooks and prints the number 47
```

Solve problems using the pointer technique, linear search binary search

Exercise 4.6 Write a LINEAR search to find who is bringing applesauce to the potluck.

Psuedocode

Ashley	Chicken
Beth	Cole Slaw
Chris	Cake
Dave	Cake
Doug	Potatoes
Eric	Pie
Fred	Ham
Ginger	Mac Cheese
Hepzibah	Applesauce
Isa	Green Beans
Jud	Rolls
Mark	Brownies
Sally	Tea

Exercise 4.7 Write a BINARY search to find what Mark is bringing to the potluck. How many steps will it take to find Mark?

Pseudocode

Ashley	Chicken
Beth	Cole Slaw
Chris	Cake
Dave	Cake
Doug	Potatoes
Eric	Pie
Fred	Ham
Ginger	Mac Cheese
Hepzibah	Applesauce
Isa	Green Beans
Jud	Rolls
Mark	Brownies
Sally	Tea

Lesson 4 - File Access

All of the data that we have learned to store in arrays disappears the instant the program stops running. When we want to “permanently” store data it needs to be written to an external file of some type.

One choice is a text file. Data is simply written one line at a time. You would open the file, telling the computer the **file mode** (whether you want to write to or read from the file) and creating a number reference to the file.

Maybe we want to keep a list of all the phone numbers of people that call an office. A text file is created, usually by launching a word editor like MS Notepad and saving an empty file with the filename and location we want. Perhaps we create a file called “PhoneNum.dat”. The .dat extension tells us the file stores data, but .txt, or anything else will work too.

To write data to a file,

- Open the file using the OPEN command. You will need to specify the file name, file mode (OUTPUT), and file number. The file number is then a reference to the file.
- Use PRINT, followed by the file number and the data you want to write.
- Close the file using the CLOSE command.

For example, the following pseudocode opens a file using mode OUTPUT and number 1, and then saves the data to the file. If there is existing data in the file, the new data is written to the next empty line.

```
OPEN "testfile.dat" FOR OUTPUT AS #1
PRINT #1, "487-3766"
CLOSE #1
```

To read data from a file,

- Open the file using the OPEN command. You will need to specify the file name, file mode (INPUT), and file number.
- Use INPUT, followed by the file number and the data you want to write.
- Close the file using the CLOSE command.

Here we read a phone number

```
OPEN "testfile.dat" FOR INPUT AS #1
INPUT #1, text$
CLOSE #1
```

The data we read is stored in our program as text\$, and we can then print the data or do anything we usually do with user inputted data. Or maybe our program is connect to a phone system and could automatically call all the numbers in the file.

We can also store more than a single piece of data on each line. Let's say we want to store the first name, last name, phone number, city and state of every caller. These five pieces of data could all be stored in our data file in one of three ways:

- Sequential files
- Delimited (or Comma Separated Value) files
- Fixed Width files

In **Sequential files**, we could write each piece of data on a separate line, as shown to the right. If you wanted Bill Walsh's phone number, you'd have to go to the second set of data and the phone number is the third piece of data in the second set or the 8th line.

Both of the other options involve **parsing** – that is cutting a longer string into pieces to get the specific data you want.

```
Joe
Asburger
403-8927
Cincinnati
Ohio
Bill
Walsh
608-7756
Topeka
Kansas
... and so on
```

The second option would be to write a set of data on a single line and use some kind of marker to separate the pieces of data. These files are referred to **Delimited files**, since the marker delimits or marks where the next bit of data starts. Very often the marker is a comma, so we talk about **Comma Separated Values files** or **CSV files**. The same data is shown below in CSV format. To get Bill Walsh's phone number here, read the second line of data into a variable. In that variable find the second comma, and the phone number will be the next eight characters.

```
Joe,Asburger,403-8927,Cincinnati,Ohio
Bill,Walsh,608-7756,Topeka,Kansas
... and so on
```

A third option, called **Fixed Width files**, involves setting-up a certain amount of room for each piece of data. Maybe the longest first name you can imagine is 15 characters, last names might need 20

characters, the phone number is 8 (unless you want the area code), city might require 20 and the state 20. Then

each line of this data file always has 83 characters and the phone number

```
Joe      Asburger  403-8927  Cincinnati  Ohio
Bill     Walsh     608-7756  Topeka      Kansas
... and so on
```

always starts on character 36. MID(string, 36,8) would give you the phone number.

Exercise 4.8 Write a FOR loop to read each line of the file and print just the phone numbers. File format: firstname = 12 characters, lastname = 15, phone = 13, reservation = 3

Pseudocode			

Bob	Smith	472-5685	no
Jill	Jones	325-8554	yes
Ed	Moore	854-8989	no

Exercise 4.9 Write a FOR loop to read each line of the CSV file and print just the phone numbers. Use the FIND(string, search character, incidence) function to return the position of the second comma. For example StartPos = FIND("running fast", "n", 3) finds the third "n" and stores the position as StartPos.

Pseudocode

Bob,Smith,472-5685,no
Jill,Jones,325-8554,yes
Ed,Moore,854-8989,no

Lesson 5 - Introduction to SQL

The best way to “permanently” store and use data that is outside of your program is a **database**. Lets say that I want to keep up with textbook sales in the bookstore. For each book sold, I want to record the date of purchase, the name of the textbook, the selling price and the salesperson. For each book sold, that’s four pieces of information that gets stored in a **table** like the one below.

Sales Table

DateTime	TextName	SellPrice	Employee
01/14/2011 8:36 AM	Applied Physics	\$16.95	Johnson
01/14/2011 8:45 AM	Modern Psychology	\$145.83	Johnson
01/14/2011 2:15 PM	English Today	\$127.33	Waites
01/15/2011 10.15 AM	Modern Psychology	\$145.83	Fredrickson
01/17/2011 4:25 PM	Art History	\$156.04	Waites

Each sale generates a new row in the table – this row is referred to as a **record**. Each record can have many **fields**. Here we see four fields in each record, and we have five records in our table.

There may be a second table that contains the Employee’s name and home phone number and when they started working at the bookstore.

EmpPhone Table

StartDate	Employee	HomePhone
01/14/1987	Johnson	125-7845
01/14/2005	Fredrickson	354-8896
01/14/2010	Waites	407-8566

Here’s a third table that gives the names of the textbooks and publisher info

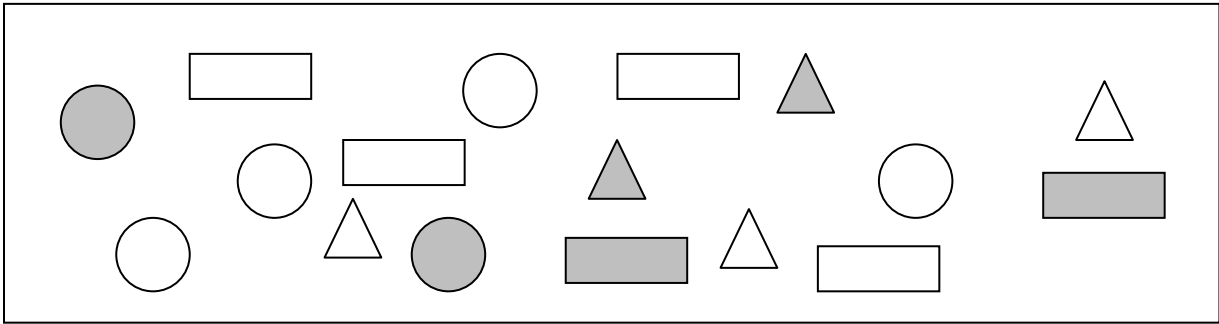
Books Table

TextName	BuyPrice	Publisher	BookRepPhone
Applied Physics	\$9.95	McGraw-Hill	(652)253-6659
Modern Psychology	\$82.47	Addison	(809)872-8966
English Today	\$64.17	McGraw-Hill	(652)253-6659
Art History	\$73.28	Prentice Hall	(287)665-4289

All the tables together make up the **Database**.

In order to search the database we use SQL – Structured Query Language. SQL simply says

go pick “this stuff” from “location” where “this is true” and put it “here”



In the picture above there is a collection of shapes. If we were to reach into this box we could scoop up items that had similar properties. Maybe we want to choose all the triangles, or just the shaded triangles, or maybe all the shaded shapes.

In the same way, SQL might look at the first table and choose all the books that were sold by Waites and put that information into a new table called WaitesSales:

```
SELECT TextName FROM Sales WHERE Employee = "Waites" INTO TABLE WaitesSales
```

WaitesSales Table

TextName
English Today
Art History

Since we only asked for the one field TextName, that's the only field in the new table.

We could have entered the following query and would have had that information too.

```
SELECT TextName, DateTime FROM Sales WHERE Employee = "Waites" INTO TABLE WaitesSales2
```

WaitesSales2 Table

TextName	DateTime
English Today	01/14/2011 2:15 PM
Art History	01/17/2011 4:25 PM

We could also pull information from more than one table. If we wanted the TextName, the SellPrice and the BuyPrice of all the books that were sold by Waites we would have to go to the Sales and Books tables.

```
SELECT Sales.TextName, Sales.SellPrice, Books.BuyPrice FROM Sales, Books WHERE Sales.TextName = Books.TextName and Sales.Employee = "Waites" INTO TABLE WaitesSales3
```

WaitesSales3 Table

TextName	SellPrice	BuyPrice
English Today	\$127.33	\$64.17
Art History	\$156.04	\$73.28

Index

accumulator	37	JavaScript	14
Algorithmic solutions.....	3	Linear Search Algorithm.....	80
AND logical operator.....	7	local variable.....	62
array	73	Logical operators.....	7
Binary Search Algorithm.....	80	module diagram.....	54
Call by reference.....	63	modules.....	53
Call by value.....	63	Multi dimensional arrays.....	76
CASE statement	46	nested loop.....	42
Cohesion.....	54	nesting decisions	32
Comma Separated Values files	85	Object Oriented Programming	71
constant.....	5	Operators	6
control program	53	OR logical operator.....	7
counter	37	order of operations	6
Coupling	54	parallel array	73
CSV files.....	85	parsing.....	85
data type.....	4	pointers	81
database	87	psuedocode	22
Delimited files	85	record	87
Development Environment	72	scope.....	62
dimension	74	Sequential files.....	85
dynamic arrays	74	SQL.....	87
element.....	73	static arrays	74
External documentation	70	Structured Query Language.....	87
field.....	87	syntax	14
file mode.....	84	table	87
Fixed Width files	85	tags.....	14
Flowcharts	20	text file.....	84
FOR loop.....	36	truth table	7
global variables.....	62	two dimensional array	76
Graphical User Interface	71	UNTIL loop.....	37
GUI	71	User documentation	70
Heuristic solutions	3	variable.....	3
HTML.....	14	Variable names	4
IF statement.....	28	Visual BASIC.....	71
index	73	WHILE loop	37
Internal documentation.....	69		