Fundamentals of Programming

The Basics of BASIC

By

Samuel T. Scott

Fundamentals of Programming

The Basics of BASIC

By

Samuel T. Scott

The fundamentals of programming are not difficult to learn. You can create unique computer programs making the computer do exactly what you want it to do in a relatively short time. Essentially, learning to write computer programs is like learning another language, like Spanish, German, French, Korean, Chinese, etc. Instead of learning to speak the other language, you will only have to learn how to read/write in it.

Like all languages, people or computer, a programming language has rules for grammar and spelling. A computer language also has a lexicon of words reserved or "key" that have very specific uses as commands or instructions for the computer to execute. When you write computer programs and use these reserved or key words you create control structures, program lines of code for the computer to execute and perform a task.

Why would you want to learn a computer programming language? Even with all of the commercial off the shelf (COTS) application software available today, specific, concise, efficient programs need to be written for a myriad of tasks and projects. Quick BASIC (Beginner's All-purpose Symbolic Instruction Code) is just one of many programming languages available for you to create custom application software.

Since computers were created to help us perform repetitive tasks, programs follow the paradigm of sequence, selection, and iteration (repetition). You may think of writing a computer program in terms of writing a letter or a piece of music. All have a beginning, middle, and an end. All are creative. All must be well written in order to efficiently calculate numbers or sort files, in the case of a computer program; communicate information, in the case of a letter; or sound like music and not noise, in the case of a piece of music.

You do not simply sit down and write a computer program. Neither do you simply sit down and write a letter or a piece of music. None of these are created unless a need or task to be performed has been identified. In the case of a computer program, you cannot write a computer program to perform a task you cannot already perform manually. In other words, you cannot program a computer to solve a problem you don't already know how to solve. The reason for writing a computer program to solve a problem or perform a task is to get the task done more quickly, repetitively, and without error.

The next several pages will walk you through the basics of BASIC programming. In developing this tutorial on the fundamentals of programming, I assumed you have no prior knowledge or experience in structured programming. As appropriate, you will see screen captures of what you should be seeing on your computer. Since the Microsoft QBASIC interpreter and help files are only a combined 318K in size, they will easily fit on a 3.5-inch floppy disk with plenty of room for the programs you will need to write. You can run QBASIC from a FD. If you do, then you should be sure to copy all of the files either to another FD or to your hard drive (create a folder and call it QBASIC) to have a back up copy of all of your work. As you create and edit your QBASIC programs, you should always copy the files to another FD or your hard drive to preclude loss of work.

You will be able to write functional QBASIC programs using only a handful of key or reserved words/control structures. By the end of this course of instruction, you should be able to:

1. Write a functional QBASIC program solving a mathematical problem,
2. Recognize, identify, and describe various control structures to include predicting the computer's action,
3. Identify and differentiate between inputted and derived variables in a program.

Before you attempt to write a QBASIC program, you must correlate the general information processing cycle to specific programming tasks or problems. In general, the information processing cycle accepts input (data), processes it, and produces output (information). In a computer program we must also create an interface for the user allowing them to interact with the computer in order to effectively enter the input and receive the output.

From the beginning, let's use good programming practice in the design and development of even simple programs. We will apply the Program Development Life Cycle (PDLC) to help you establish good programming practices and habits. When we get to the point of actually writing the code, we'll use some general guidelines and models to simplify some of the administrative aspects of programming and help document some program facts.

The PDLC consists of 6 steps:

1. Analyze Problem
2. Design Programs
3. Code Programs
4. Test Programs
5. Formalize Solution
6. Maintain Programs

To get started, let's begin with a relatively simply task or problem to solve and create a rudimentary calculator. The task we have been given is to multiply any given number by 15.3%. In analyzing the problem (PDLC Phase 1), you can rewrite this task mathematically as a formula. In this case, the formula could be:

Answer = number * 0.153 (Note: For computers, we must use the asterisk (*) for the multiplication sign.)

In our analyzed restatement of the formula, the task to multiply by 15.3% needs to be re-written 0.153 since the computer needs to have values identified in decimal rather than percentage form.

It should also seem obvious to you the program will initially have two variables. One variable will be needed to represent the given number and the second variable will needed to represent the calculated number. What's a *variable*? A mathematical formula contains knowns and unknowns, or *constants* and *variables*. We usually represent the *variables* with letters, e.g. a + 3 = 5 where "a" is the variable and the "3 and 5" are constants. In programming, it can get vary confusing and difficult to remember which variables are represented by individual letters so we use words to describe the variable. Hence, the words "*answer*" and "*number*" can be used as the *variable names* in a program line of code. In this case, the variable "*number*" would be the input and the derived, calculated, or computed variable would be "*answer*". Almost any string of characters can be used for variable names except for Key or Reserved words that have specific functions in QBASIC. A partial listing of QBASIC Keywords is provided at the end of this document.

As we continue and enter Phase 2 of the PDLC, we need to lay out the design of the program and our approach to solving the problem and implementing the information cycle. Following the format of the Information Processing Cycle, we know we need user "inputs" before our formula in order to get the required "outputs". It is also a good idea to start the program code listing by documenting some administrative information to help us keep on track. Conceptually, our program could look something like this:

** Administrative Data Section

Program Name: Percentage Calculator *(or other name/description of your choosing)*
Program Version: 1.0 *(start with version 1.0, as we make minor changes we'll increment on the "tenths" side of the decimal, e.g. 1.2, 1.4, etc. Major changes will increment like 2.0, 3.0, etc.)*
Programmer: John Doe *(Type your name here.)*
Program Description: *(What will the program do? What task(s) will it perform? What does the user have to do? Etc.)*

** Variable Data Section

Variable Name	Variable Type	Description
Number	number	value inputted by user
Answer	number	value calculated for output

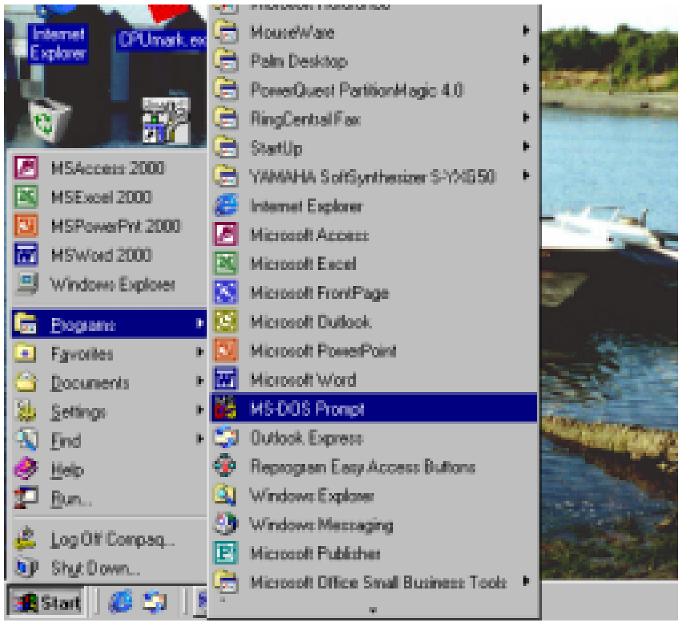
** Procedure Section

```
User inputs number
Answer = number * .153
Computer prints answer on screen
```

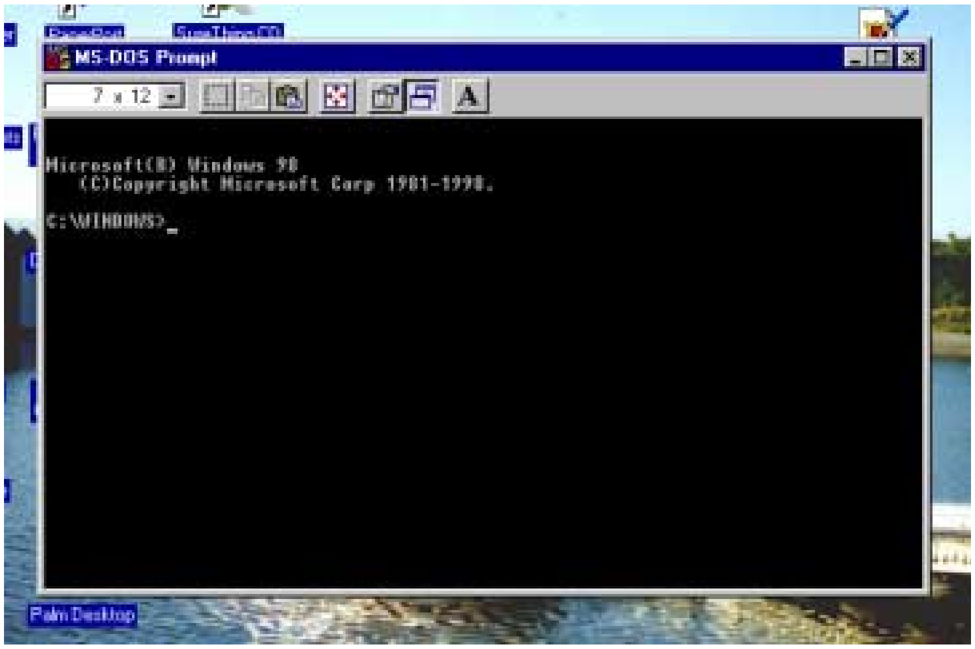
At this point you should refer to your listing of QBASIC Key (Reserved) words as we need to begin using programming terms as we move into Phase 3 of the PDLC. If you are going to run the Quick BASIC editor from a 3.5" FD, then you need to insert the disk into the 3.5" FDD. You now have a choice of either opening an MS-DOS window on your desktop to run QBASIC or using the full screen. If you open an MS-DOS window, then you will be able to easily switch to other programs or windows without closing QBASIC. If you want to run QBASIC using the full screen from the 3.5" FD, then <*double-click*> on the **My Computer** icon on your desktop and <*double-click*> on the **3 ½" Floppy** icon in the My Computer window and <*double-click*> on the QBASIC icon (not the QBASIC help icon).

I will illustrate running Quick BASIC from the 3.5" FD in a MS-DOS window.

<*click*> on <*start*>, <*programs*>, <*MS-DOS Prompt*>

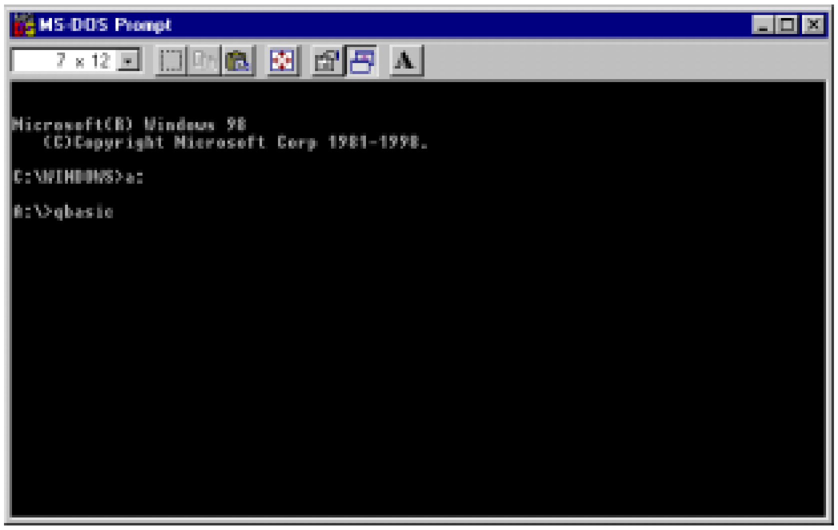


This window should appear on your screen.



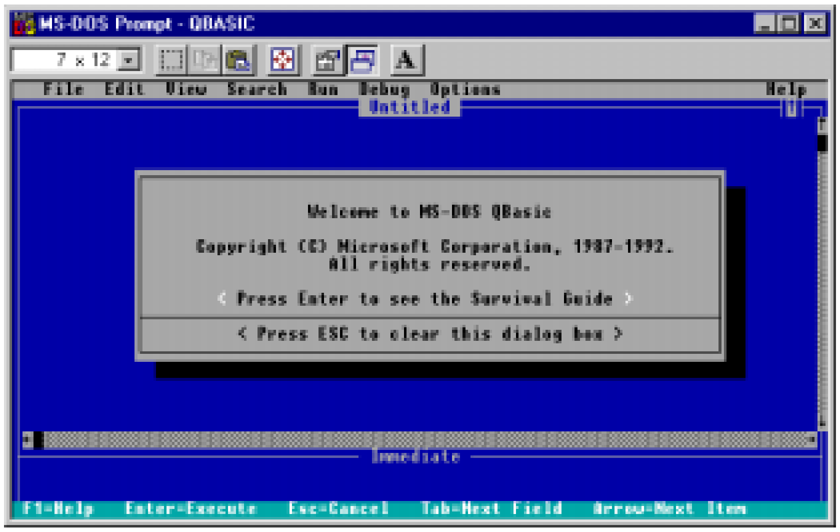
```
Microsoft(R) Windows 98
   (C)Copyright Microsoft Corp 1981-1998.

C:\WINDOWS>_
```

At the "C" prompt in the MS-DOS Window type **A:** and press the enter key. At the "A" prompt, type *qbasic* and press the enter key. This action will start the QBASIC editor.



```
Microsoft(R) Windows 98
   (C)Copyright Microsoft Corp 1981-1998.

C:\WINDOWS>a:

A:\>qbasic
```

You should now see the following QBASIC welcome screen. You will see this screen each time you run QBASIC. You can press <*Enter*> to see the Survival Guide or press <*esc*> to clear the dialog box so you can start typing your program.


MS-DOS Prompt - QBASIC
7 x 12
File Edit View Search Run Debug Options Help
Untitled
Immediate
F1=Help Enter=Execute Esc=Cancel Tab=Next Field Arrow=Next Item

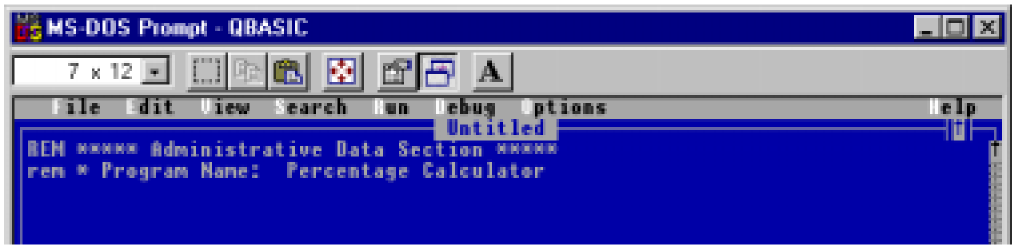
```
Welcome to MS-DOS QBasic

Copyright (C) Microsoft Corporation, 1987-1992.
All rights reserved.

< Press Enter to see the Survival Guide >

< Press ESC to clear this dialog box >
```

QBASIC is a command line interface program. Consistent and error free typing is essential. Your mouse is supported and will allow you to point and click to move the cursor and access the menu bar and its drop down menus. You can also activate and access the Menu bar and its drop down menus by pressing the <*alt*> key. You can then navigate across and up/down the menus by using the cursor control (arrow) keys on your keyboard.

Special Note: Each program line must either start with a Key word or be a mathematical/logical formula.

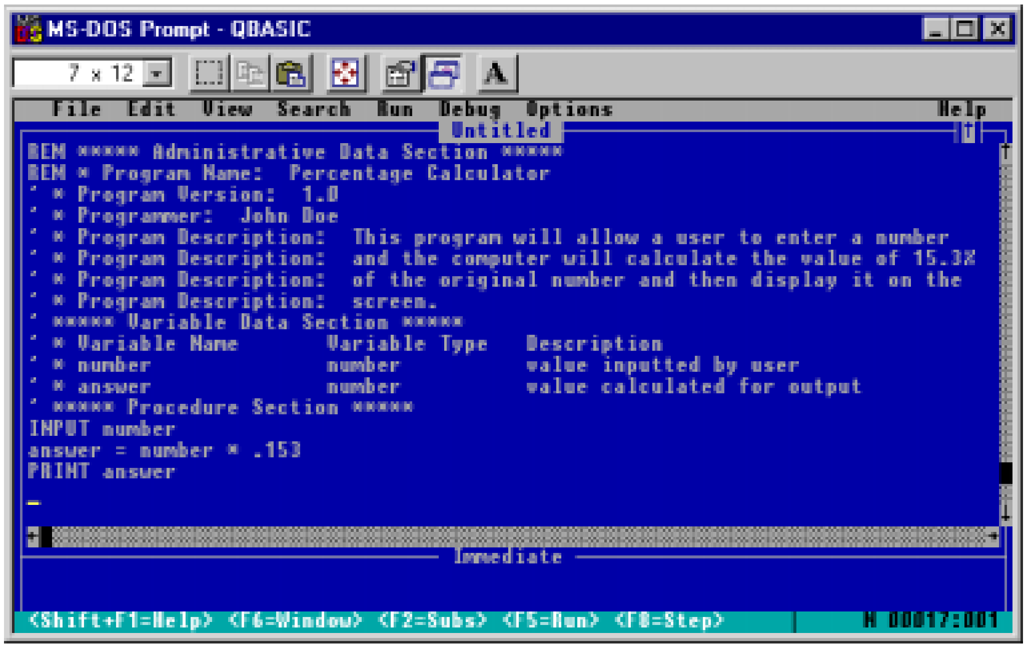
For this first program, we are only going to use three QBASIC Keywords. We will use the Keyword **REM** (an abbreviation for the command "remark") to enter our documentation information. Using **REM** is a way to put notes and comments into a program's code listing without affecting the execution of the program. When the computer "sees" the Keyword **REM**, it ignores everything after it on the line and moves on. If you want to be really lazy in your typing, you can even abbreviate the abbreviation for **REM** by using the *single* quote, that's the key next to the Enter key on the keyboard. We will use the Keyword **INPUT** to enable the computer to solicit input from the user. We will use the Keyword **PRINT** to display the answer on the screen.

When typing in the QBASIC editor, do NOT type with the CAPS LOCK on. Type primarily in lower case or use a combination of upper and lower case letters as appropriate. The QBASIC editor is programmed to look for strings of characters that match Keywords and when it finds them after you press the <*Enter*> key, it will capitalize them for you. This is a good way to catch spelling errors.



```
REM ***** Administrative Data Section *****
rem * Program Name:  Percentage Calculator
```

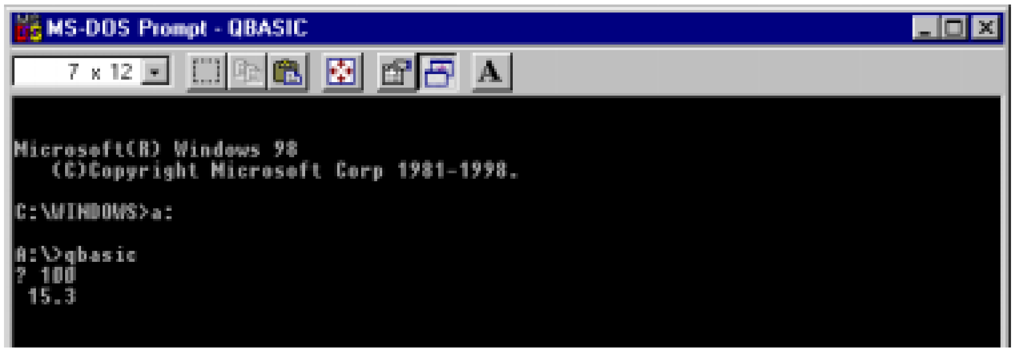
Notice when you type the Keyword **REM** in lower case and then press <*Enter*> at the end of the line how the editor finds the Keyword and coverts it to all caps. Since these lines are for documentation, you can use other keyboard symbols, like the asterisk to visually set them apart from other lines.


MS-DOS Prompt - QBASIC
7 x 12
File Edit View Search Run Debug Options Help
Untitled
Immediate
<Shift+F1=Help> <F6=Window> <F2=Subs> <F5=Run> <F8=Step>
N 00017:001

```
REM ***** Administrative Data Section *****
REM * Program Name:  Percentage Calculator
' * Program Version:  1.0
' * Programmer:  John Doe
' * Program Description:  This program will allow a user to enter a number
' * Program Description:  and the computer will calculate the value of 15.3%
' * Program Description:  of the original number and then display it on the
' * Program Description:  screen.
' ***** Variable Data Section *****
' * Variable Name        Variable Type     Description
' * number               number            value inputted by user
' * answer               number            value calculated for output
' ***** Procedure Section *****
INPUT number
answer = number * .153
PRINT answer
```

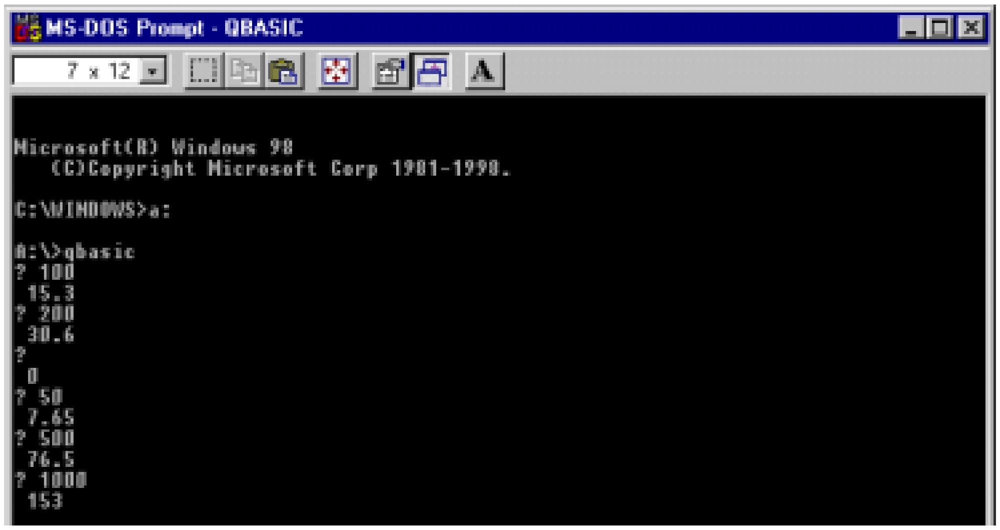
At this point you could initially "test" the program (PDLC Phase 4). You can "run" the program either by pressing function key <*F5*> or by clicking on <*Run*><*Start*> on the QBASIC menu bar. When you run the program to test it, use input values having easily known answers, e.g. input 100 and the answer should be 15.3.

You will notice the program will execute once. It will display a question mark "?" on the screen and after you enter a number, like 100, and press <*Enter*>, the answer 15.3 immediately displays. Pressing any key on the keyboard will return you to the QBASIC editor window.



```
Microsoft(R) Windows 98
   (C)Copyright Microsoft Corp 1981-1998.

C:\WINDOWS>a:

A:\>qbasic
? 100
 15.3
```

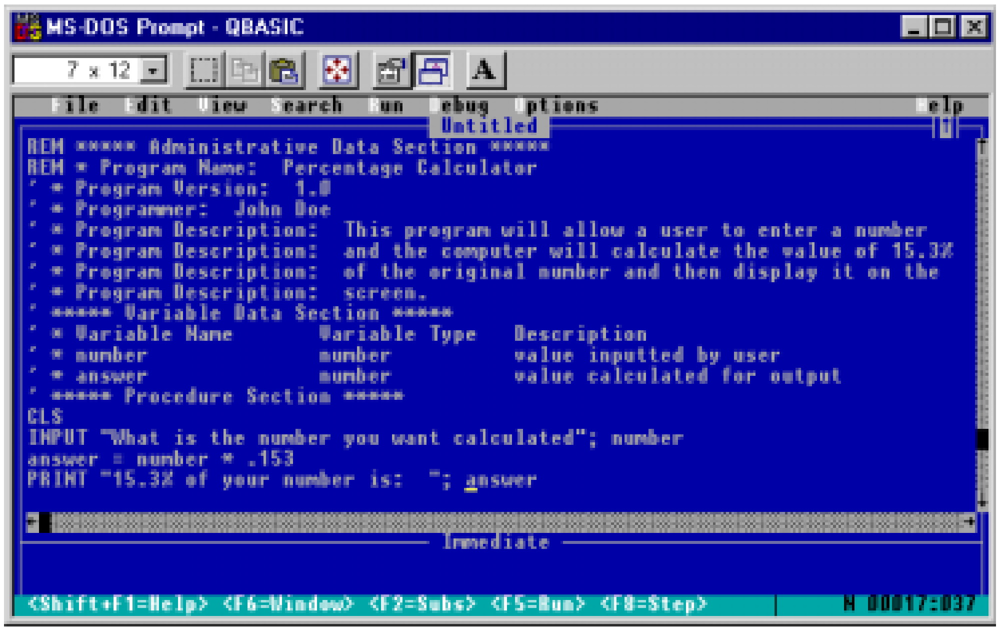
While functional, the program is very basic with virtually no interaction or interface with the user. Anyone other than the programmer would probably not be able to use this program. Also, you will note if you execute the program repeatedly, old inputs and outputs remain cluttering up the screen and need to be removed.



```
Microsoft(R) Windows 98
   (C)Copyright Microsoft Corp 1981-1998.

C:\WINDOWS>a:

A:\>qbasic
? 100
 15.3
? 200
 30.6
?
 0
? 50
 7.65
? 500
 76.5
? 1000
 153
```

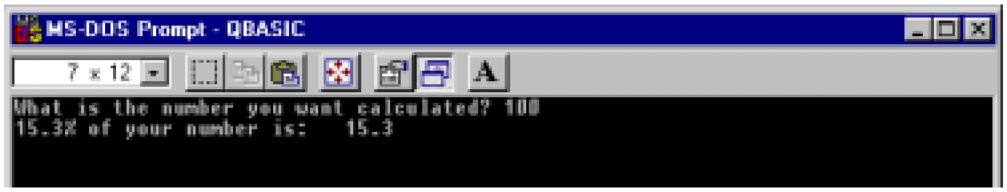
We need to add some user interaction and be more descriptive to enable virtually any user who needs to perform this task to do so easily. We will do this by editing our program lines and adding lines using additional Keywords to perform additional tasks. The next Keyword to include in our program is **CLS**. This Keyword is used to clear the screen. We use **CLS** to clear the screen of any previous data and usually use it at the beginning of a program or module. Additionally, we are going to make the computer display or print text messages or guidance to the user in conjunction with the **INPUT** and **PRINT** Keywords we are now using. QBASIC will print strings of characters on the screen exactly, or literally, as you type them, by putting the text inside double quotes. You will also have to separate the literal text string from the variable name with a semicolon. It's just a grammar rule we have to abide by.

In order to edit your program, you must move the cursor to the location in the program listing where you want to make the change. Move the cursor either by mouse or cursor control (arrow) keys. Make the editing changes you see in the next screen capture. QBASIC is very picky about punctuation so type carefully.


MS-DOS Prompt - QBASIC
7 x 12
File Edit View Search Run Debug Options Help
Untitled
Immediate
<Shift+F1=Help> <F6=Window> <F2=Subs> <F5=Run> <F8=Step>
N 00017:037

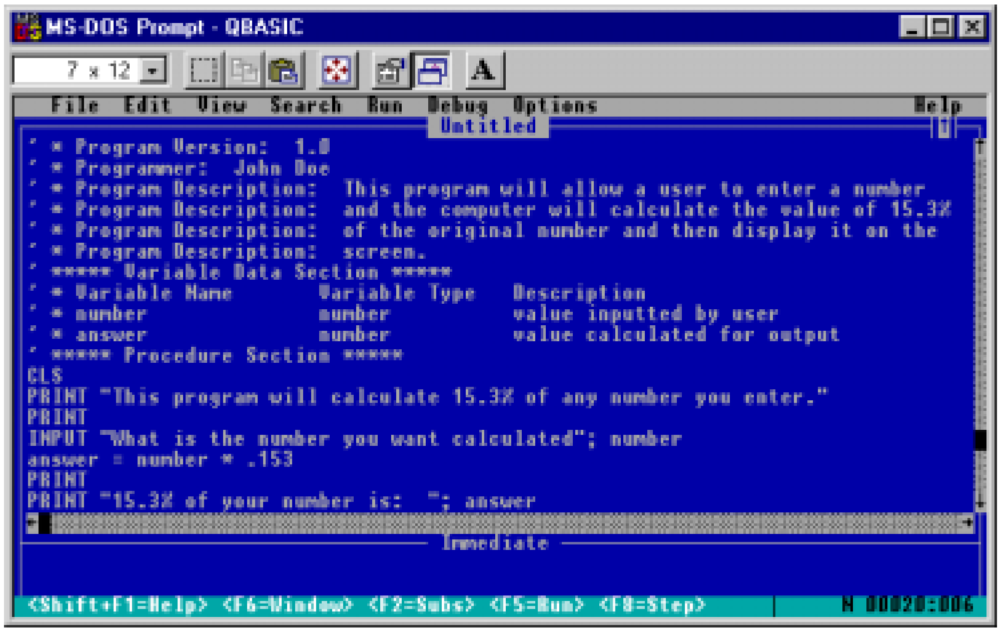
```
REM ***** Administrative Data Section *****
REM * Program Name:  Percentage Calculator
' * Program Version:  1.0
' * Programmer:  John Doe
' * Program Description:  This program will allow a user to enter a number
' * Program Description:  and the computer will calculate the value of 15.3%
' * Program Description:  of the original number and then display it on the
' * Program Description:  screen.
' ***** Variable Data Section *****
' * Variable Name        Variable Type    Description
' * number               number           value inputted by user
' * answer               number           value calculated for output
' ***** Procedure Section *****
CLS
INPUT "What is the number you want calculated"; number
answer = number * .153
PRINT "15.3% of your number is:  "; answer
```

While the execution of this slightly modified program is more interactive and descriptive, it hasn't changed the basic formula and we still get the same results. In order words, we have not changed the algorithm, which is/are the line/lines of code that actually "solve" the problem or tell the user it can't be solved. In this case, the algorithm for this program is the formula. You will also notice the screen is wiped "clean" each time you run the program and the user gets some description of what to type in and what they have received.



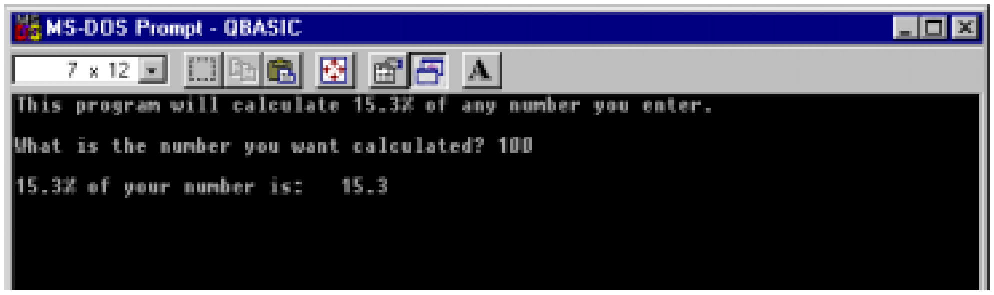
```
What is the number you want calculated? 100
15.3% of your number is:   15.3
```

We can also use the **PRINT** keyword to simply display literal strings of characters to the screen, or if nothing follows the keyword **PRINT**, to just print a blank line. The ability to print blank lines provides a means to format lines of text on the screen, making it easier on the user.


MS-DOS Prompt - QBASIC
7 x 12
File Edit View Search Run Debug Options Help
Untitled
Immediate
<Shift+F1=Help> <F6=Window> <F2=Subs> <F5=Run> <F8=Step>
N 00020:006

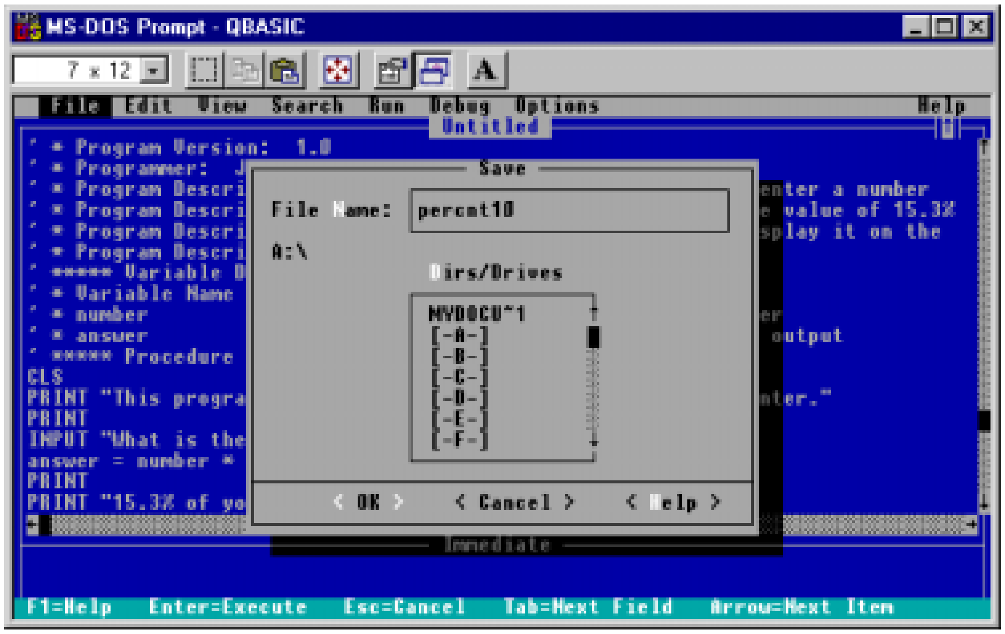
```
' * Program Version:  1.0
' * Programmer:  John Doe
' * Program Description:  This program will allow a user to enter a number
' * Program Description:  and the computer will calculate the value of 15.3%
' * Program Description:  of the original number and then display it on the
' * Program Description:  screen.
' ***** Variable Data Section *****
' * Variable Name       Variable Type    Description
' * number              number           value inputted by user
' * answer              number           value calculated for output
' ***** Procedure Section *****
CLS
PRINT "This program will calculate 15.3% of any number you enter."
PRINT
INPUT "What is the number you want calculated"; number
answer = number * .153
PRINT
PRINT "15.3% of your number is:  "; answer
```

With these latest changes, the program is reasonably interactive with the text printed on the screen in a simple, uncomplicated, and readable manner.



```
This program will calculate 15.3% of any number you enter.

What is the number you want calculated? 100

15.3% of your number is:   15.3
```

Now would be a good time to save your work. Either <*Click*> on <*File*><*Save*> or press <*Alt*><*F*><*S*> to open the QBASIC file save dialogue box. When you type the filename, limit it to 8 characters. QBASIC is a DOS program and does not support long file names. You will see under the File Name text in the Save dialogue box the letter A:\ indicating the computer is accessing the A drive and your file will be saved on the A drive. After you type in a filename (8 characters or less) then either <*Click*> on <*OK*> or press <*Enter*>.


MS-DOS Prompt - QBASIC
7 x 12
File Edit View Search Run Debug Options Help
Untitled
Save
Dirs/Drives
< OK > < Cancel > < Help >
Immediate
F1=Help Enter=Execute Esc=Cancel Tab=Next Field Arrow=Next Item

```
' * Program Version:  1.0
' * Programmer:  J
' * Program Descri
' * Program Descri
' * Program Descri
' * Program Descri
' ***** Variable D
' * Variable Name
' * number
' * answer
' ***** Procedure
CLS
PRINT "This progra
PRINT
INPUT "What is the
answer = number *
PRINT
PRINT "15.3% of yo
```

```
enter a number
e value of 15.3%
splay it on the
er
 output
nter."
```

```
File Name: percnt10
A:\
MYDOCU~1
[-A-]
[-B-]
[-C-]
[-D-]
[-E-]
[-F-]
```

You should hear your computer access your 3.5" FDD and the name you gave the file will appear in the top of the QBASIC window. At this point you could print the program listing and send it along with a memo to the problem initiator announcing your formal success (PDLC Phase 5). At this time you should exit QBASIC as you think you are finished. This would also be a good time to use Windows Explorer to make all necessary file/disk copier/backups.

Shortly after you send the info to the problem initiator, you will probably get back a memo saying "That's great!, Now, can you make it to do . . . .". Enter PDLC Phase 6, program maintenance.

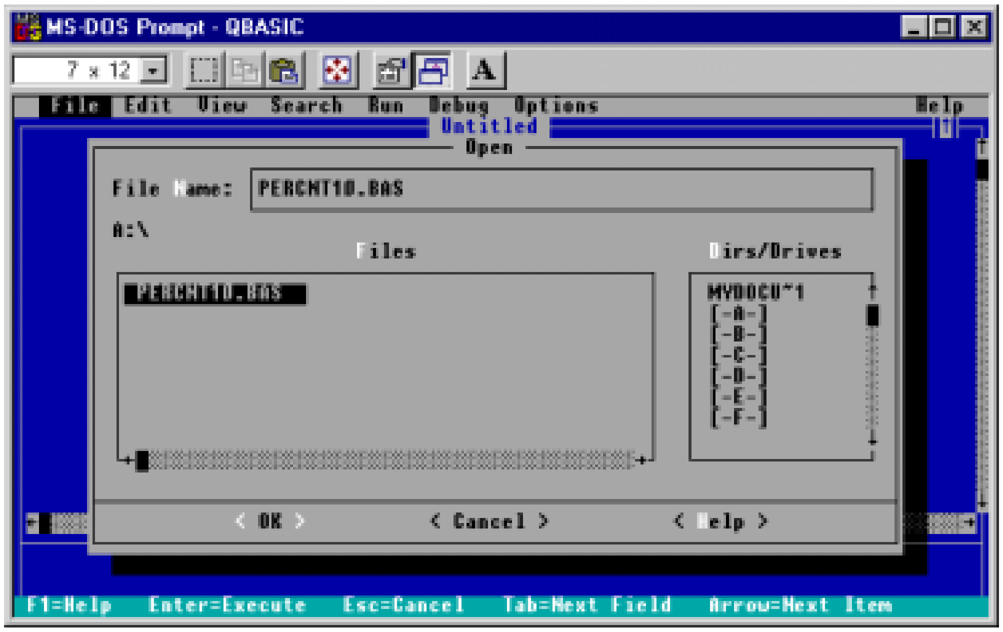
Now, the program needs to accept two different variables, i.e. the tasker wants to be able to change the percentage value as well as the number. Additionally, the tasker has multiple numbers to process at any one time and needs the program to iterate or repeat through the calculation procedure, something called looping for programmers.

Since we already have a solid, fundamental program, enhancing it to meet the new requirement will not be difficult. You will now need to employ multiple input variables, add a loop, and add a variable to capture a loop or even program ending condition.

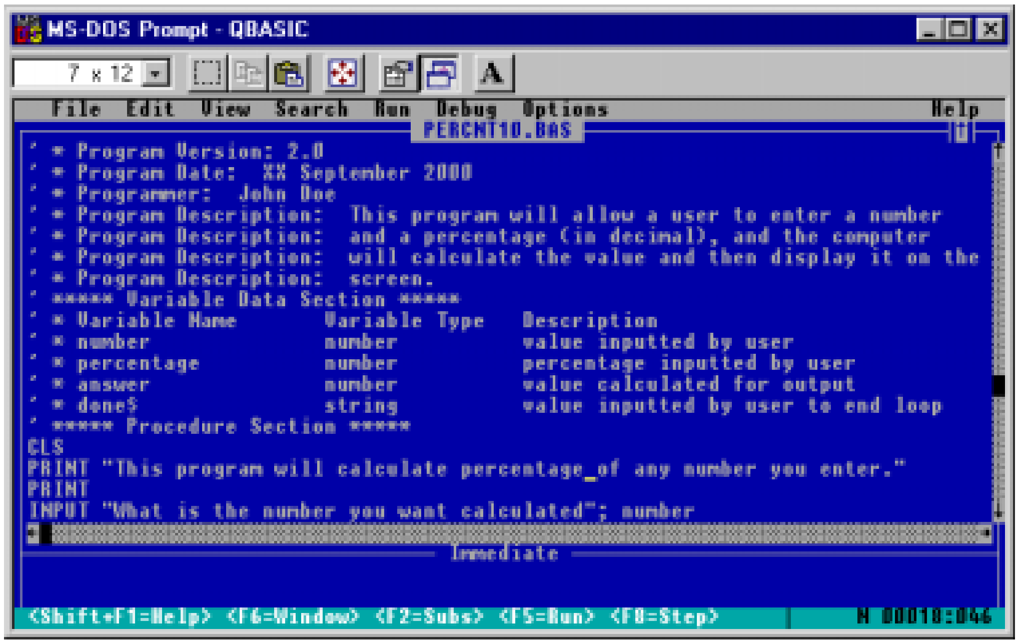
One of the new variables will need to be set to accept letters or numbers, what is called a character string or just string. String variables cannot be used for calculations but work well for accepting letters, names, etc., and then providing logical tests. You designate a variable name as a string variable by adding a dollar sign, the "$", to the end of the variable name.

An easy way to accommodate the requirement for repetition is by using the **DO UNTIL** / **LOOP** keyword combination. Set up properly, in this program, the procedure of input, processing, and output will continue over and over until the user tells the program to cease by typing in a specific string character. The specifics of the "DO" loop structure are outlined in the QBASIC Keyword sheet.

In order to begin revising and maintaining your program, you will once again have to open a DOS Window and start QBASIC. After clearing the initial dialogue box, you need to load your program into memory so you can modify it. Access the QBASIC menu bar *File* command and select the *Open* option. The Open dialogue box should either access your "A" drive or the folder on you hard drive where you have QBASIC loaded and store program files. The open dialogue box should show the filename for your PERCNT10.BAS file you previously finished and filed. Either use your mouse to point to the filename and <*double-click*> to load it or move from pane to pane by pressing the <*Tab*> key and press the <Up> or <Down> arrow keys to highlight the filename and then press <*Enter*>.



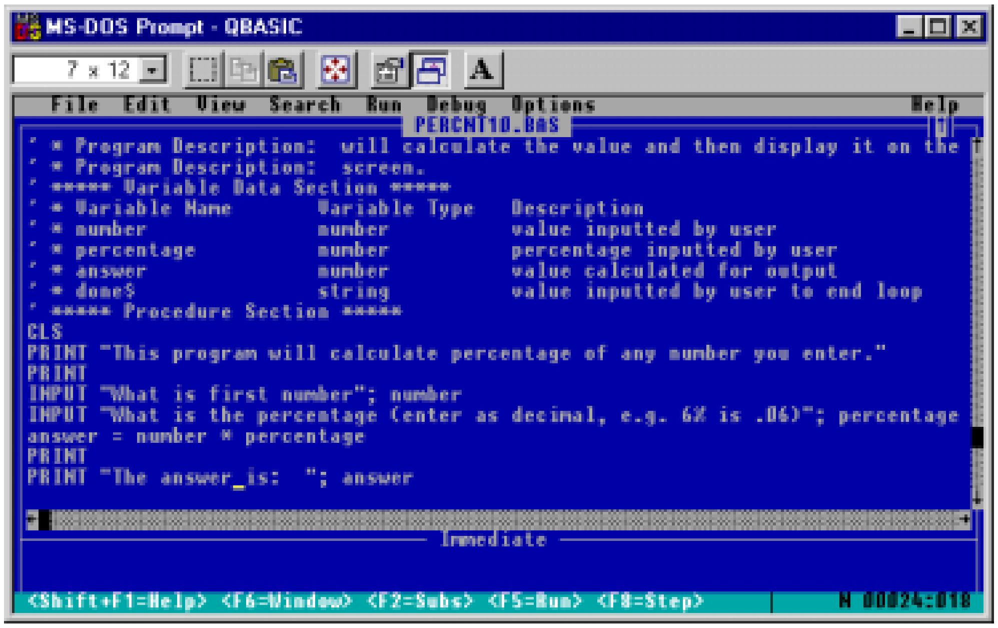
With the program loaded into memory we can now start modifying it to meet the new stated requirements. The additions will be rather extensive, so the program version number should probably be changed to 2.0, as it will be a major revision. In addition to the functional changes, you will need to edit the administrative data to reflect the current program.


MS-DOS Prompt - QBASIC
7 x 12
File Edit View Search Run Debug Options Help
PERCNT10.BAS
Immediate
<Shift+F1=Help> <F6=Window> <F2=Subs> <F5=Run> <F8=Step>
N 00018:046

```
' * Program Version: 2.0
' * Program Date:  XX September 2000
' * Programmer:  John Doe
' * Program Description:  This program will allow a user to enter a number
' * Program Description:  and a percentage (in decimal), and the computer
' * Program Description:  will calculate the value and then display it on the
' * Program Description:  screen.
' ***** Variable Data Section *****
' * Variable Name        Variable Type   Description
' * number               number          value inputted by user
' * percentage           number          percentage inputted by user
' * answer               number          value calculated for output
' * done$                string          value inputted by user to end loop
' ***** Procedure Section *****
CLS
PRINT "This program will calculate percentage of any number you enter."
PRINT
INPUT "What is the number you want calculated"; number
```

As you can see from the preceding screen capture, the version number was changed and some of the program description edited to reflect the new program capabilities. Also, two new variables were documented, one a number for the percentage and the other a string for the loop ending variable. Note the loop ending variable is designated as a character or string variable by the dollar sign at the end of the variable name.

Next, add another program line to allow the user to input the percentage. Be sure to notify the user to enter the percentage as a decimal number, e.g. 23% should be entered as .23. You will then also need to edit the algorithm and substitute the percentage variable name for the constant .153 value used before. After these changes are made and working, you can add the loop control structure.


MS-DOS Prompt - QBASIC
7 x 12
A
File Edit View Search Run Debug Options Help
PERCNT10.BAS
Immediate
<Shift+F1=Help> <F6=Window> <F2=Subs> <F5=Run> <F8=Step>
N 00024:018

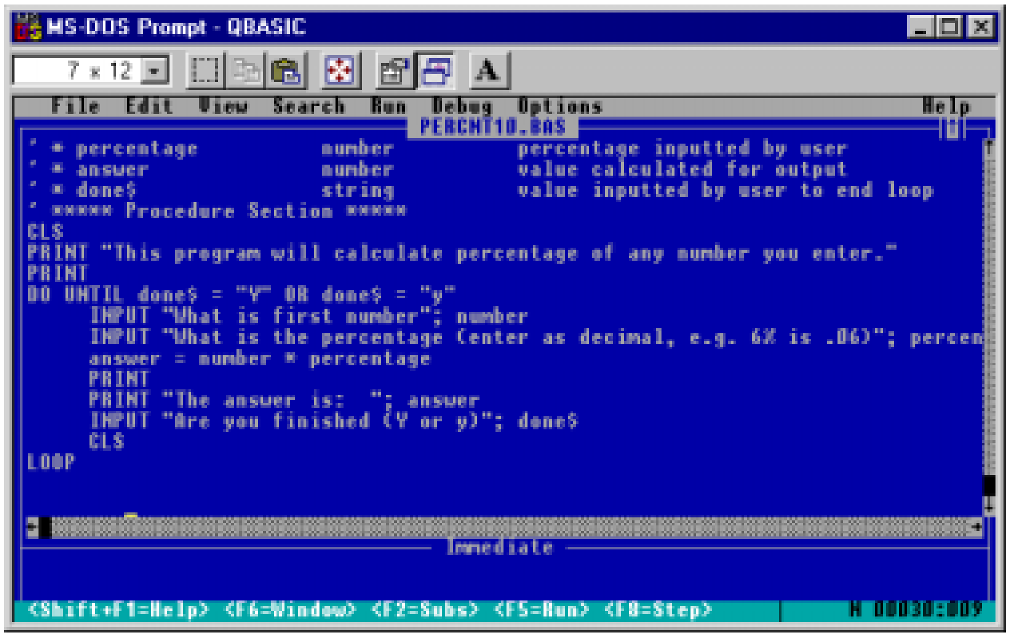
```
' * Program Description:  will calculate the value and then display it on the
' * Program Description:  screen.
' ***** Variable Data Section *****
' * Variable Name       Variable Type   Description
' * number              number          value inputted by user
' * percentage          number          percentage inputted by user
' * answer              number          value calculated for output
' * done$               string          value inputted by user to end loop
' ***** Procedure Section *****
CLS
PRINT "This program will calculate percentage of any number you enter."
PRINT
INPUT "What is first number"; number
INPUT "What is the percentage (enter as decimal, e.g. 6% is .06)"; percentage
answer = number * percentage
PRINT
PRINT "The answer is:  "; answer
```

The only portion of the program needing to repeat or loop are the input-processing-output program lines. We can easily put these lines inside a Do Loop control structure so that multiple iterations of processing can be performed without having to re-run the program.

The grammar and syntax for the Do Loop structure is to start the loop identifying an ending condition which will be false initially. Before the **LOOP** command at the end of the loop structure, you need to query the user for his input on whether or not he is finished with the program. The loop ending condition needs to check for the loop ending variable as either an upper or lower case letter, so the user doesn't have to be specific about anything but the letter itself.

```
Do Until done$ = "Y" or done$ = "y"
        Input-processing-output lines of code
        Input "Are you finished (Y or y)"; done$
Loop
```

The logic of the Do Until/Loop control structure is elegant and simple. On first pass through the program the value of the done$ variable is null so it is definitely not Y or y. Since it is null, the program execution proceeds to the next line and performs the input-processing-output lines of code. The user get the opportunity to enter a value for done$ when asked if finished. After the user types in a value for done$, the loop keyword returns execution back up to the Do Until line where the value of done$ is checked to see if it is Y or y. If the value of done$ is Y or y, then the program execution skips all lines of code between the Do Until and Loop keywords and executes the next program line after the Loop keyword. If there are no more program lines after the Loop keyword, then the program ends. If the value of done$ is anything other than Y or y, then the programs lines after the Do Until are executed again. By convention, all program lines between Do Until and Loop are indented for visual clarity and ease of reading for the programmer.


MS-DOS Prompt - QBASIC
7 x 12
File Edit View Search Run Debug Options Help
PERCNT10.BAS
Immediate
<Shift+F1=Help> <F6=Window> <F2=Subs> <F5=Run> <F8=Step>
N 00030:009

```
' * percentage            number          percentage inputted by user
' * answer                number          value calculated for output
' * done$                 string          value inputted by user to end loop
' ***** Procedure Section *****
CLS
PRINT "This program will calculate percentage of any number you enter."
PRINT
DO UNTIL done$ = "Y" OR done$ = "y"
     INPUT "What is first number"; number
     INPUT "What is the percentage (enter as decimal, e.g. 6% is .06)"; percen
     answer = number * percentage
     PRINT
     PRINT "The answer is:  "; answer
     INPUT "Are you finished (Y or y)"; done$
     CLS
LOOP
```

Placing a **CLS** keyword on the line before the **LOOP** keyword forces the program to clear the screen before each new set of inputs.

At this point you should save the revised program (<*File*><*Save* As>) as a new name so you will have both the original and your enhanced version 2.0 for comparison.

At this point, you have taken a user requirement or problem, run it through the PDLC and produced a working piece of custom written applications software. You should be able to apply these tools and techniques to other similar tasks or problems involving inputs, mathematical processing or formulas, and outputs, all within the iterative control structure of a "Do" loop.

Other problems, while seeming more complex, may simply require more than one formula line in order to provide some intermediate calculated values to be used by other formulas. The objective in these cases is to minimize to the greatest extent possible requiring the user to perform any manual calculations prior to inputting any values and make the computer manipulate all raw data into the desired information.

Remember the paradigm of sequence, selection, and iteration. Program lines of code have to be in the correct sequence in order to properly compute ultimate output. Computers don't make errors. Programmers make errors. Users make errors. Good programmers write code facilitating accuracy and minimizing the potential for user errors.

Even in this program there is one user input item that could be prone to error which, the result will not be catastrophic but will cause a waste of time. Potentially, if the user is required to convert a percentage to a decimal in order for the formula to work, the user will periodically enter a whole percentage number and not its decimal equivalent. You can program the computer to catch, or trap, the error. In programming the computer to trap or catch errors, you can choose to simply trap the error, report it to the user, and let them re-enter the data, or you can trap the error and convert it to the correct form.

In the interest of keeping with fundamentals, we'll just trap and report errors to the user here. The principle control structure and keywords used to trap errors is the **IF-THEN-ELSE** sequence. The **IF-THEN-ELSE** structure needs to be used in conjunction with a DO loop for efficiency. While we will have to add several new lines of code, no new variables are needed. The logic and syntax of the **IF-THEN-ELSE** control structure is the most complex thus far, but well within the realm of desirable fundamental tools.

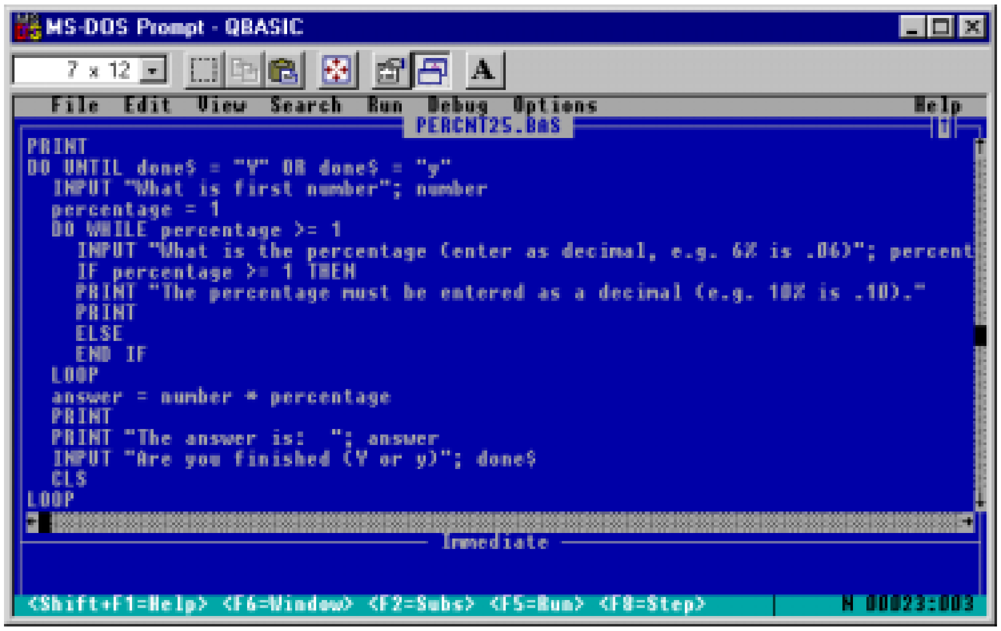
In conjunction with the **IF-THEN-ELSE** control structure we will need to use a **DO WHILE** initiated loop because we want a loop around a particular line(s) of code to continue until the user gets it right, in this case the percentage. In order to make it work, we need to initialize the variable *percentage* before the **DO WHILE** loop starts.

In a loop within a loop structure, the program lines within the nested loop are also indented to facilitate readability. Two program lines need to be inserted before the program line accepting user input for *percentage* and six lines after it need to be inserted, including the **IF-THEN-ELSE** control structure. In the **IF-THEN-ELSE** control structure, the **IF** keyword establishes a condition to be checked. If the condition is true, then the line(s) following the **THEN** are executed. If the condition is false, then the lines following the **ELSE** are executed. An **END IF** statement is needed to mark the end of the lines to be executed if the condition is false.

In our situation, the logic of the error trapping is as follows:

The *percentage* variable is initialized to be equal to 1 so the Do While loop will start because the first logical test of the Do While loop is whether or not *percentage* is greater than or equal to 1. Since *percentage* is set equal to 1 the loop executes. The user inputs a number for *percentage* and execution proceeds to the If-then line. Once again, the value of *percentage* is checked. If it is greater than or equal to 1, then a message is printed to the screen informing the user they made an error. If the initial *percentage* input is less than 1, essentially correctly inputted, then no message is displayed to the user and the error check has completed. At the loop keyword, program execution returns to the Do While line where the value of *percentage* is checked again. If the value of *percentage* is less than 1, then program execution jumps to the formula (algorithm) line and continues until completion.

With the error trapping routine completed, edit the version line to reflect version 2.5 since this is a relatively minor change to the program and save this 3rd version as a new filename for future comparison with the previous programs.


MS-DOS Prompt - QBASIC
7 x 12
File Edit View Search Run Debug Options Help
PERCNT25.BAS

```
PRINT
DO UNTIL done$ = "Y" OR done$ = "y"
  INPUT "What is first number"; number
  percentage = 1
  DO WHILE percentage >= 1
    INPUT "What is the percentage (enter as decimal, e.g. 6% is .06)"; percent
    IF percentage >= 1 THEN
    PRINT "The percentage must be entered as a decimal (e.g. 10% is .10)."
    PRINT
    ELSE
    END IF
  LOOP
  answer = number * percentage
  PRINT
  PRINT "The answer is:  "; answer
  INPUT "Are you finished (Y or y)"; done$
  CLS
LOOP
```

Immediate
<Shift+F1=Help> <F6=Window> <F2=Subs> <F5=Run> <F8=Step>
N 00023:003

Loop within a loop error checking/trapping logic is relatively straightforward and follows linear execution fairly clearly.

At this time you should be able to apply these fundamental error detecting/trapping routines and logic to other similar tasks or problems in developing new QBASIC programs.

Programming can be fun, challenging, and rewarding (and frustrating if your attention to detail and typing skills are poor). Fundamentals are straightforward and elegant, useful programs can be created using only a handful of key or reserved words, structures, and techniques.

QBASIC Reserved (Key) Words

Reserved (Key) Word	Description/Useage
CLS	Clears the screen
REM (or abbreviated as a single quote ')	Remark - computer does not execute, used to add notes, etc into source code for explanations, visual separation of different sections, modules, etc.
END	Terminates program execution
Input/Output	
INPUT *variablename*	Prints a "?" to the screen and waits for keyboard input for the variable
INPUT *"text"; variablename*	Prints the text inside the double quotes to the screen and a "?" and waits for keyboard input of variable. You must separate the "text" and variable name with punctuation, either a ";" or a ",". The semicolon places the "?" in the next position following the text. The period tabs the "?" 5 spaces to the right.
PRINT	Prints a blank line to screen
PRINT *"text"*	Prints the text inside the double quotes to the screen
PRINT TAB (xx) *"text"*	Tabs the start of printing "xx" spaces from the left margin of the screen and prints the text inside the double quotes to the screen
PRINT USING "$$##,###.##"; *variablename*	Used to format number input to the style indicated between the double quotes, in this case with a $ sign, comma at thousands, and 2 decimal places. You must separate the "format" and variable name with punctuation, a semicolon so the program will print the value of the variable indicated in the format indicated.
LPRINT	Same as PRINT only to a printer
LPRINT *"text"*	Same as PRINT "text" only to a printer
LPRINT TAB (xx) "*text*"	Same as PRINT TAB only to a printer
LPRINT USING "$$##,###.##"; variablename	Same as PRINT USING only to a printer
Loops and Decisions	
DO UNTIL *variable condition*	Starts a loop which executes until the variable condition is true, must use LOOP to return to DO UNTIL program line
DO WHILE *variable condition*	Starts a loop which executes while a variable condition is true, must use LOOP to return to DO WHILE program line
LOOP	Returns execution of a series of program lines to the beginning of a DO UNTIL or DO WHILE loop
IF *variable condition* **THEN** *statement* **ELSE** *statement*	Provides means to branch execution of program, trap errors, make decisions. If the variable condition is *true*, the *then* statement is executed, otherwise the *else* statement is executed. The *else* may be omitted if you want the program line following the IF-THEN to be executed. If a program line(s) follows the ELSE that is not simply a continuation of program execution but special actions you desire to be executed if the variable condition is false, you must use an END IF to mark the last line that would be executed if the variable condition is false.
Variable Types	
variablename	Default single precision number (calculated to 8 decimal places)
variablename!	Specifies single precision number
variablename#	Specifies double precision number (calculated to 16 decimal places)
variablename%	Specifies integer, no decimal places
variablename$	String, i.e. lables or names, a combination of letters and numbers

Note: variablenames must be continuous characters, no spaces allowed.