BASIC Softips(tm)
Entire contents copyrighted, 1990
Issue number 2
12/01/1990
6

TOC
TOC TOC TOC TOC TOC TOC TOC TOC TOC TOC TOC TOC TOC TOC TOC TOC TOC TOC TOC TO

Welcome to BASIC Softips for December 1990. Thank you downloading it and good reading! Use the up/down arrows, pageup/pagedown, control pageup/pagedown and home and end keys to navigate through an article or section. Press the ALT key or click the mouse on the top line of the screen for a menu.

√ Inside this issue get five FREE working programs!
√ Increase display speed by 50%
√ A file information program
√ Read & Change file attributes
√ Binary and BCD conversions & manipulations!
√ Reading the CURRENT drive & directory using dos
√ Draw boxes anywhere!
√ Business qaulity graphing with GRAPHIT.BAS
√ Smaller exectable files
√ How to determine if a color monitor is installed

This months segments include:

END TOC
ADD 1

Q&A Q&A
Q&A Q&A Q&A Q&A Q&A Q&A Q&A Q&A Q&A Q&A Q&A Q&A Q&A Q&A Q&A Q&A Q&A Q&A Q&A

The Q&A section is for any questions you have regarding anything, except maybe the meaning of life.

Q: How can I make BASIC print faster to the screen?

A: Well, of course printing is one of the most important aspect of your program. Printing to the screen is almost always needed to output data to the user. To understand how to print faster, we must first understand printing in general. BIOS is the Basic Input Output System ROM of your PC. ROM BIOS printing is notoriously slow. One of the biggest problems which we face is that printing is not only for characters. There are also control codes which can be printed! For example, the bell character, the tab character, the backspace character and others. These special characters do not print per se on the screen. Instead, BIOS examines the item to be printed. If it is printable then it gets printed. If it is a control character, then the appropriate action is taken. This examination of the characters to be printed takes time. Unless we write a special routine to directly address the video memory, we just have to live with the PRINT command.

How do you get around this?

Well, there are a number of things we can do to speed up our printing. Let's use the following example code fragment. Often we use the LOCATE command to position the cursor start point then we use PRINT.

```
 DEFINT A-Z

 FOR X = 1 TO 100
   LOCATE 5, 5, 1
   PRINT "HELLO WORLD."
 NEXT
```

This code will print the "HELLO WORLD" message 100 times at screen row 5 column 5. This code takes an average of .61 seconds to run on my machine. Lets see what we can do with PRINT. When PRINT is used the item is printed to the screen. PRINT supports many useful features, not all of which are discussed here. Of note though is the fact the by default PRINT always prints a carriage return and line feed after the last item. This again, takes time to address ROM, to scroll the screen etc. Luckily we can turn off this feature by using a semi-colon at the end of the PRINT statement. This alone can shave 40%-50% off the time needed to print. I next re-wrote the code as shown below.

```
 DEFINT A-Z
 FOR X = 1 TO 100
   LOCATE 5, 5, 1
   PRINT "HELLO WORLD.";
 NEXT 'X
```

This code runs in approximately .49 seconds! A savings of 20%! If you can, use a semi-colon after each print statement.

Using LOCATE 5, 5, 1 position us at row 5, column 5 and turns on the cursor. Turning on the cursor takes time - so does updating it's position after each character is printed. Lets take a look at LOCATE to see if we can save some time there too. LOCATE takes 5 options as shown below.

LOCATE ROW, COLUMN, CURSOR, START, STOP

ROW   - is the screen row (from 1 to 60, depending on video mode set by
        WIDTH command)
COLUMN - is the screen column (from 1 to 80, depending on video mode set
        by WIDTH command)
CURSOR - 0= turn visible cursor off, 1=turn it on
START - Starting scan line of cursor
STOP   - Stop scan line of cursor

We don't need a cursor while we are printing, so lets turn it off to save those dreaded BIOS calls! I re-wrote the code as shown below.

```
DEFINT A-Z

LOCATE , , 0
```

```
 FOR X = 1 TO 100
   LOCATE 5, 5
   PRINT "HELLO WORLD.";
 NEXT 'X
```

I turned OFF the cursor first. Why do we need a cursor when we are printing? The above loop runs in approximately .39 seconds! A savings of another 20%! So turn OFF the cursor before beginning a print routine.

Finally, a removed the literal string "HELLO WORLD." and replaced it with a variable. I also replaced the LOCATE 5, 5 with LOCATE X, Y. Contrary to what is says in the QB manuals, using a CONSTANT in place of an actual number makes BASIC operate much faster.

```
 DEFINT A-Z

 LOCATE , , 0
 X = 5
 Y = 5
 A$ = "HELLO WORLD."

 FOR X = 1 TO 100
   LOCATE X, Y
   PRINT A$;
 NEXT 'X
```

This time the loop ran in .34 average seconds! That's another 14% reduction in printing time! Using a string literal in BASIC means that the string must be examined before it can be printed. BASIC must pass the entire string to the print routines. Worse, BASIC must make a copy of the string first - this means finding and allocating memory and all that. Using a variable lets BASIC pass the position in memory of the string - not the actual string. This too saves precious time.

All the changes I made gave me a reduction from .61 seconds to .34 seconds for this loop to run. That's better than a 50% decrease in print time! Just like last month we see again that many times it isn't the language - it's our use of it that makes for slow programs.

Q: How can I make my programs smaller?

A: BASIC typically does not produce small executables. The reason for this is that QB includes EVERY possible routine in the finished program - even if you are not using them all! That's just how BASIC works. BUT, The MS PDS BASIC 7.0 and 7.1 use another approach that only includes a SUBSET of the entire language library in the program. Better still, Cresenct softwares BASIC replacement library PDQ only includes those routines actually used. Whats all that mean?

This months PROJECT has a program named BOX. I compiled BOX using QB 4.5, PDS 7.0 and then using PDQ. The program sizes are shown below.

BOX.OBJ linked using QB 4.5	23,146 bytes
BOX.OBJ linked using PDS 7.0	18,282 bytes

BOX.OBJ linked using PDQ 2.2   6,468 bytes

That's what all that means! A real difference in code size. If you don't
have the luxury of having alternate libraries like PDQ there are still
some things that you can do to reduce code size of the disk. Unfortuately
some of these measures might require a differnt version of BC.EXE than
you have. But...

1) Make reusable sub routines that can be shared by several procedures.
  Reuseable code is the easiest way to reduce EXE size.

2) Link using the /EX option - this packs the executable into the smallest
  size possible - but programs will load slower.

3) Compile using the /FPA and/or /OT options - this will reduce code size
  also (if your version of BC supports this).

One sure way to reduce code size is to use NO ON ERROR or any EVENT trapping
link ON KEY or ON ... GOSUB etc., There are many other ways around ANY
programming situation where you can check for validity first, then enter
a routine. I feel that using ON ERROR is best left alone. Why? BASIC
adds FOUR (count 'em 1234 - 4) bytes PER LINE OF CODE if you use any
event trapping or ON ERROR or related error handling code! Imagine how
bigggg and ssssslow this makes your code! Maybe next month I'll dicuss
ways out of using ON ERROR and all that - but for the mean time I'll
just say that I haven't used ON ERROR for at least 5 years! -HM

If any of you have similar experiments or questions, please send them in!

Please submit any questions, problems, corrections or
comments to :

Electronic (preferred):
Editor
BASIC Softips
CompuServe 76120, 2413

Paper (if you must):
Editor
BASIC Softips
135 Chestnut Street
Bridgewater NJ 08807

END Q&A Q&A
FORUM FORUM
FORUM FORUM FORUM FORUM FORUM FORUM FORUM FORUM FORUM FORUM FORUM FORUM
FORUM

We had over 70 downloads last month. Not bad at all! Talk it up folks!
I received two letters and a phone call with the first issue. I only
uploaded it to CIS, MSLANG forum. I would like to get wider distribution.
Anyone can send this to another BBS or distribute it to your friends
or schools. If you have any ideas about spreading the word please drop
me a line. To Use CompuServes EMAIL, type GO EMAIL at any prompt and
send it to:

Editor
BASIC Softips
CompuServe 76120, 2413

This month the READER.EXE has been updated - READER now FULLY supports B&W or mono adapters. Start up READER with a /B to see it in B&W. I also added a new option called Maximum Lines - this option displays in the most lines your system can suppoty, as follows:

ADAPTER TYPE	MAXIMUM LINES DISPLAYED
B&W (no graphics adapter)	25 lines
CGA	25 lines
EGA	43 lines
VGA	50 lines

The startup default is 25 lines. Maximum Lines is a toggle and will let you change to or from the high resolution mode.

You can add this type of display to your programs using the WIDTH statement in BASIC as follows:

```
WIDTH 40/80, 25/30/43/50/60
```

30 and 60 line displays are ONLY supported in graphics modes, which READER does not use.

Also, the screen may be re-sized using the mouse at any time. Just grab the lower right corner of the screen box or the left side and move it where you want.

I accidentally erased the first letter while logged onto CIS, but the sender found some typos in the program! Thanks for the feedback - and keep on reading!

The second sender is below.

From: csri.toronto.edu!tmsoft!masnet!canremote!brent.ashley
Subject: SOFTIP
To: 76120.2413@compuserve.com
X-Mailer: MaS-Relayer Usenet/Internet/Fidonet/PCBoard Gateway

Hank;

I just dloaded a copy of your SofTip magazine. Very impressive, indeed! I have recently become moderator for the BASICs conference on the international NorthAmeriNet BBS network, and my first announcement was to recommend that everyone have a look at your new mag. Good luck in this - I hope to see more of the same!

```
-----
Brent
-----

brent.ashley@canremote.uucp  |
                         | Canada Remote Systems,Toronto, Ontario
                         | International NANET Host.
```

Thanks Brent! There's a WHOLE lot more a 'comin!

Heres a freebie! The BEST way to see if the system has a color
monitor installed is show below -HM

```
DEF SEG = 0
IF PEEK(&H463) = &HB4 THEN
'mono
ELSE
'color
END IF
```

END FORUM FORUM

PROJECT OF THE MONTH PROJECT OF THE MONTH

PROJECT OF THE MONTH PROJECT OF THE MONTH PROJECT OF THE MONTH PROJECT OF THE

This months project is a multiple purpose graphing and box drawing routine, written entirely in BASIC. If will operate under QB or the PDS.

This month some concepts developed are of proportionality and scale. The graphing program below can automatically resize itself to fit inside the bounding box you define. It also takes care of all the nagging little details like positioning the text, creating the bars, positioning the labels and it even selects the colors! All you do is define some simple variables and give the routine the data to graph - graphit does the rest.

It is actually two programs - GRAPHIT.BAS and BOX.BAS, when you load it into the QB environment, use the cut and save options to make it back into two programs, then use the MOVE subroutine command to put SUB BOX back into BOX.BAS. You could also leave it alone - it will run as is in QB or PDS.

Within GRAPHIT there is a routine to print text either centered, left or right justified or vertically. You can use it in you programs as it.

BOX.BAS supports 5 different box types and will even optionally fill in a box. Or you can have box draw boxes around items on the screen and not fill it the box - the choice is yours.

Use the Cut segment command from the main utilities menu to save this file to disk. Give it a name like GRAPH1.BAS so you can keep them straight each month as we add features. When you load this into BASIC, delete all of the text lines above. GRAPHIT has a self running demo of the routines. Just load it and run it!

```
'Start of program-------------------------------------------------------
'
'(C)Copyright 1990 Marquis Computing Inc. All rights reserved.
'You may use this program for anything or any purpose including inclusion
'into programs you write BUT you cannot sell this source code. Written by
'Hank Marquis. revised 10/18/90.
DEFINT A-Z

DECLARE FUNCTION trim$ (totrim$)

DECLARE SUB PrintLine (Lrc%, text$, wid%, Hpos%, Vpos%, Clr)
DECLARE SUB Box (Ulr%, Ulc%, Lrr%, Lrc%, Fore%, Back%, Fill, LineType)
DECLARE SUB Graph (DataToGraph%(), DataLabels$(), Graf AS ANY)


 'The following is a demo showing all of the types of graphs supported.

 SCREEN , , 0, 0
 CLS
 SCREEN , , 1, 0
 CLS

 TYPE Gt
   Title AS STRING * 80           'main graph title
   titlejust AS INTEGER           'left, right or justify the title
   TitleColor AS INTEGER          'color of the title
   SubTitle AS STRING * 80        'a sub title for the graph
   Fore AS INTEGER                          'foreground color
   Back AS INTEGER                          'background color
   Gtype AS INTEGER               'graph type, 1 or 2 Vert or Horiz
   Box AS INTEGER                           'bounding box on or off (0 or 1)
   scale AS INTEGER               'what scale to use
   UpperLeftRow    AS INTEGER          '
   UpperLeftCol    AS INTEGER ' setup for graph corners
   LowerRightRow    AS INTEGER         '
   LowerRightCol    AS INTEGER         '
 END TYPE

 DIM Graf AS Gt

 Items = 6
 REDIM DataToGraph(Items), DataLabels$(Items)

 'For every label, there is a corresponding value, as shown below.

 DataLabels$(1) = "Pens"
 DataToGraph(1) = 24

 DataLabels$(2) = "Pencils"
 DataToGraph(2) = 10

 DataLabels$(3) = "Pads"
 DataToGraph(3) = 51

 DataLabels$(4) = "Erasers"
```

```
DataToGraph(4) = 46

DataLabels$(5) = "Glue"
DataToGraph(5) = 31

DataLabels$(6) = "Paper clips"
DataToGraph(6) = 63

'These are the controling element of the graphing process. You can
' add more by adding them to the Type...EndType above. Then define them
' in the routines. for example, you might add a variable to set the color
' of the SubTitle to somthing other than the default.

Graf.Title$ = "Sales For November"
Graf.SubTitle$ = "Cases Sold"
Graf.Fore = 15        'fore ground
Graf.Back = 0         'background
Graf.Box = 1          '1 = border, 0 = no border
Graf.scale = 0        '0 = use highest item as 100%
Graf.titlejust = 2     '1=left, 2=center, 3=left, 4=vertical
Graf.TitleColor = 14   'foreground color of title

'----------------------------------------------------
'This is the start of demo I put together for you. It uses
' both graph types. This first one is a plain bar graph.

Graf.UpperLeftRow = 1
Graf.UpperLeftCol = 2
Graf.LowerRightRow = 20  'maximum is 24 using type 1 graph
Graf.LowerRightCol = 60
Graf.Gtype = 2
Graph DataToGraph(), DataLabels$(), Graf
SLEEP

'------------------------------------------------------------
'This one it the same as above, only smaller. Notice how the position
'  of the bars is automatic, as is the size of the bars. The first
'  time it is displayed, it has no bounding box, the second time it does.

CLS
Graf.Box = 0                  'turn frame off
Graf.UpperLeftRow = 1
Graf.UpperLeftCol = 2
Graf.LowerRightRow = 15       'maximum is 24 using type 1 graph
Graf.LowerRightCol = 30
Graf.Gtype = 2                 'bar graph
Graph DataToGraph(), DataLabels$(), Graf
SLEEP

Graf.Box = 1                  'turn frame back on
Graph DataToGraph(), DataLabels$(), Graf
SLEEP

'----------------------------------------------------------------
'To have more than one graph on the screen at once, just don't erase
' the last one!
```

```
Graf.Fore = 0
Graf.Back = 7
Graf.titlejust = 1           '1=left, 2=center, 3=right
Graf.UpperLeftRow = 7
Graf.UpperLeftCol = 35
Graf.LowerRightRow = 24  'maximum is 24 using type 1 graph
Graf.LowerRightCol = 80
Graf.Gtype = 1
Graph DataToGraph(), DataLabels$(), Graf
SLEEP

Graf.titlejust = 3           '1=left, 2=center, 3=right
Graph DataToGraph(), DataLabels$(), Graf
SLEEP

'--------------------------------------------------------------------
'
COLOR 7, 0
CLS
Graf.Fore = 15
Graf.Back = 1
Graf.titlejust = 2           '1=left, 2=center, 3=right
Graf.UpperLeftRow = 1
Graf.UpperLeftCol = 1
Graf.LowerRightRow = 20       'maximum is 24 using type 1 graph
Graf.LowerRightCol = 80
Graf.Gtype = 1
Graph DataToGraph(), DataLabels$(), Graf
SLEEP

Graf.scale = 100              'set 100% as scale
Graph DataToGraph(), DataLabels$(), Graf
SLEEP

Graf.scale = 50               'set 50% as scale
Graph DataToGraph(), DataLabels$(), Graf
SLEEP



'--------------------------------------------------------------------
'This next demo shows how the graphs make use of relative position
' markers. I have added more items to display. Notice how the size of
' the bars as well as the location & position of the labels for the
' bars change as the graph definition changes.

COLOR 7, 0
CLS

Items = 10
REDIM DataToGraph(Items), DataLabels$(Items)

DataToGraph(1) = 24
DataToGraph(2) = 10
DataToGraph(3) = 51
DataToGraph(4) = 46
DataToGraph(5) = 31
DataToGraph(6) = 63
DataToGraph(7) = 21
DataToGraph(8) = 15
DataToGraph(9) = 41
DataToGraph(10) = 36

DataLabels$(1) = "Pens"
DataLabels$(2) = "Pencils"
DataLabels$(3) = "Pads"
DataLabels$(4) = "Erasers"
DataLabels$(5) = "Glue"
DataLabels$(6) = "Clips"
DataLabels$(7) = "Paper"
DataLabels$(8) = "Mrkers"
DataLabels$(9) = "Tape"
DataLabels$(10) = "Pins"

Graf.Gtype = 1
Graf.Fore = 0
Graf.Back = 7
Graf.UpperLeftRow = 2
Graf.UpperLeftCol = 2

FOR X = 5 TO 23
  Graf.LowerRightRow = X
  Graf.LowerRightCol = 55 + X
  CLS
  Graph DataToGraph(), DataLabels$(), Graf
NEXT
SLEEP

'---------------------------------

Graf.Gtype = 2
Graf.Fore = 15
Graf.Back = 0
Graf.scale = 0                'set scale as highest data value
Graf.UpperLeftRow = 2
Graf.UpperLeftCol = 2
Graf.LowerRightRow = 20       'maximum is 24 using type 1 graph

FOR X = 30 TO 80
  Graf.LowerRightCol = X
  CLS
  Graph DataToGraph(), DataLabels$(), Graf
NEXT
SLEEP


'---------------
Items = 8
REDIM DataToGraph(Items), DataLabels$(Items)

DataToGraph(1) = 24
DataToGraph(2) = 10
DataToGraph(3) = 51
DataToGraph(4) = 46
DataToGraph(5) = 31
DataToGraph(6) = 63
DataToGraph(7) = 21
DataToGraph(8) = 15

DataLabels$(1) = "Pens"
DataLabels$(2) = "Pencils"
DataLabels$(3) = "Pads"
DataLabels$(4) = "Erasers"
DataLabels$(5) = "Glue"
DataLabels$(6) = "Clips"
DataLabels$(7) = "Paper"
DataLabels$(8) = "Mrkers"

Graf.LowerRightCol = 80
CLS
Graph DataToGraph(), DataLabels$(), Graf

SLEEP

'------------------------------------------------------------------------
'this demo is to show how Graphit automatically adjusts the bar width to
' the number of items to graph.

Items = 5
REDIM DataToGraph(Items), DataLabels$(Items)

DataToGraph(1) = 24
DataToGraph(2) = 10
DataToGraph(3) = 51
DataToGraph(4) = 46
DataToGraph(5) = 31

DataLabels$(1) = "Pens"
DataLabels$(2) = "Pencils"
DataLabels$(3) = "Pads"
DataLabels$(4) = "Erasers"
DataLabels$(5) = "Glue"

CLS
Graph DataToGraph(), DataLabels$(), Graf

SLEEP
SCREEN 0, 0, 0
CLS

Ulr = 8        'Upper Left Row
Ulc = 20       'Upper Left Column
Lrr = 15       'Lower Right Row
Lrc = 55       'Lower Right Column
Fill = 1       '1=filled in box, 0=framed outline only
LineType = 1   'line types are 1 to 5
Fore = 9      'foreground color
Back = 1       'background color
```

```
 'draw boxes, filled in
 FOR X = 1 TO 5
   Box Ulr, Ulc, Lrr, Lrc, Fore, Back, Fill, X
   Fore = Fore + 1
   LOCATE Ulr + 2, Ulc + 2, 0
   PRINT "Hope you liked the demo! Check"
   LOCATE , Ulc + 2, 0
   PRINT "out the attached program - BOXES!"
   LOCATE , Ulc + 2, 0
   PRINT "It's a complete box drawing "
   LOCATE , Ulc + 2, 0
   PRINT "program. Ready to use!"
   SLEEP 1
 NEXT

 SLEEP

'------------------------------------------------------------------
'Cut this out is QB and make a new module, then use the F2, ALT+M
'commands to put the sub BOX back into this program.

  '---------------------------------------------------------------
  'This sub draws a box anywhere on the screen and optionally
  ' fills it in with a color.
  '
  ' (C)Marquis Computing 1990
  ' Written by Hank Marquis
  ' You can use this routine BUT you can't sell this source code.

'DECLARE SUB Box (Ulr, Ulc, Lrr, Lrc, Fore, Back, Fill, LineType)   '

COLOR 0, 7
CLS
PRINT STRING$(2000, CHR$(249));

Ulr = 8        'Upper Left Row
Ulc = 20       'Upper Left Column
Lrr = 15       'Lower Right Row
Lrc = 55       'Lower Right Column
Fill = 1       '1=filled in box, 0=framed outline only
LineType = 1   'line types are 1 to 5
Fore = 15      'foreground color
Back = 1       'background color

'draw boxes, filled in
FOR X = 1 TO 5
  Box Ulr, Ulc, Lrr, Lrc, Fore, Back, Fill, X
  LOCATE Ulr + 3, Ulc + 7, 0
  PRINT "This is box style #"; LTRIM$(RTRIM$(STR$(X))); "."
  SLEEP
NEXT


SUB Box (Ulr, Ulc, Lrr, Lrc, Fore, Back, Fill, LineType)
```

```
 '---------------------------------------------------------------
 ' here we set up all the variables we will use. Keep all non-variant
 ' calulations out of the main routine. Do all your math & string
 ' creation first - this will allow your sub routine to draw the box
 ' as fast as it can.

 COLOR Fore, Back

 wid = Lrc - Ulc                        'the width of the box
 startdraw = wid - Ulc                      'where to start filling from
 start = Ulr + 1                        'where to start drawing sides
 count = Lrr - 1                        'how many lines to draw

 IF Fill = 1 THEN Fill$ = SPACE$(wid) 'if we are filling make a fill line

 IF LineType > 5 OR LineType < 1 THEN LineType = 1    'five types - 1 to 5

 '----------------------------------------------------------------
 'You could put other characters in here if you want to!

 Ulc$ = MID$("┌╔╒╓█", LineType, 1)    'select the correct drawing set
 urc$ = MID$("┐╗╕╖█", LineType, 1)                   ''
 llc$ = MID$("└╚╘╙█", LineType, 1)                   ''
 Lrc$ = MID$("┘╝╛╜█", LineType, 1)                   ''
side$ = MID$("│║│║█", LineType, 1)                   ''
 top$ = MID$("─══─█", LineType, 1)                   ''

 bar$ = STRING$(wid, top$)            'make the top & bottom lines

 '---------------------------------------------------------------
 ' Here we draw the box & fill it if the fill flag is on.

 LOCATE Ulr, Ulc, 0   'only turn of the cursor once - this saves time
  PRINT bar$;
 LOCATE Ulr, Ulc
  PRINT Ulc$;         'a ; after each PRINT saves about .05 seconds/print!
 LOCATE Ulr, Lrc
  PRINT urc$;
 LOCATE Lrr, Ulc
  PRINT bar$;
 LOCATE Lrr, Ulc
  PRINT llc$;
 LOCATE Lrr, Lrc
  PRINT Lrc$;

 'draw the sides ...
 '               ... and optionally fill in the box if fill$ <> ""

 FOR X = start TO count
  LOCATE X, Ulc               'draw left side and...
   PRINT side$ + Fill$;       ' ... filling print the line of spaces
  LOCATE X, Lrc
   PRINT side$;               'draw the right side
 NEXT

END SUB
```

```
SUB Graph (DataToGraph(), DataLabels$(), Graf AS Gt)

'
'The elements used in Graf as shown below
'
'  Graf.Title$    title of this graph
'  Graf.SubTitle$ a sub title
'  Graf.Fore      foreground color
'  Graf.Back       background color
'  Graf.Box       1 = border, 0 = no border
'  Graf.scale     0 = use highest item as 100% - this makes the graph
'                 automatically scale itself to fit in the bounding box
'                 and sets the scale where 100% = highest data value
'
'                 any other value = the scale to use. Use 100 for a scale
'                 of 100, 50 for 50% etc.
'
'  Graf.titlejust  1=left, 2=center, 3=left, 4=vertical
'  Graf.TitleColor foreground color of title
'
'  Graf.UpperLeftRow  1 to (maximum lines on screen -2)
'  Graf.UpperLeftCol  1 to (maximum columns on screen -2)
'  Graf.LowerRightRow  maximum is 24 using type 1 graph
'  Graf.LowerRightCol
'
'  Graf.Gtype  1=a vertical bar graph, 2=a horizontal bar graph
'
'This is the main graphing sub. It takes in raw data and
' graph style information and then makes the graph.
'


  SCREEN , , 1, 0              'we work on screen 0 while showing 1

  Ulr = Graf.UpperLeftRow      ' setup the coordinates for the graph
  Ulc = Graf.UpperLeftCol      '               "
  Lrr = Graf.LowerRightRow     '               "
  Lrc = Graf.LowerRightCol     '               "
  wid = Lrc - Ulc + 1          '               "

  '---build a bounding box------------------------------------------
  IF Graf.Box THEN
    Fill = 1           'fill in the box
    LineType = 1       'use line type 1
    Fore = Graf.Fore   'use colors passed in Graf Type
    Back = Graf.Back   '              "
    Box Ulr, Ulc, Lrr, Lrc, Fore, Back, Fill, LineType
  END IF

  '---print graph title------------------------------------------
  IF LEN(trim$(Graf.Title$)) THEN
    Hpos = Ulr + 1                     'position the title bar
    Vpos = Ulc + 1                     '          "
    text$ = trim$(Graf.Title$)         'strip any blanks
    winwid = Lrc
  'setup colors for title
  IF Graf.TitleColor = 0 THEN Clr = Fore ELSE Clr = Graf.TitleColor
  IF Graf.titlejust = 0 THEN
     titlejust = 2
  ELSE
     titlejust = Graf.titlejust
  END IF

  'print the title
  PrintLine titlejust, text$, winwid, Hpos, Vpos, Clr
END IF

'determine scale
H = UBOUND(DataToGraph)

FOR X = 1 TO H
  'sorts through array of values & gets the highest value
  IF DataToGraph(X) > Hi THEN Hi = DataToGraph(X)
NEXT

bar$ = SPACE$(Lrc - Ulc - 2)

'if we are using the high data value as scale or using
' a different scale
IF Graf.scale = 0 THEN scale = Hi ELSE scale = Graf.scale

SELECT CASE Graf.Gtype

  CASE 1   'horizontal bar graph
  IF LEN(trim(Graf.SubTitle)) THEN
     'this section determines the position of the scale markers used
     ' and then prints the scale bar with percentage numbers
     COLOR Fore
     REDIM ScaleBar$(1)
     ScaleBar$(1) = STRING$(wid - 2, "┴")
     LOCATE Lrr, Ulc + 1

     'This next section looks imposing - but it is just choping up the
     ' scale bar into four even sections and labling them in quarters
     ' based on the scale of this graph.
     MID$(ScaleBar$(1), 1, 1) = "1"
     qty = LEN(trim(STR$(scale)))
     MID$(ScaleBar$(1), wid - 2 - qty, qty) = trim(STR$(scale))
     mid = ((wid - 2) \ 2)
     MID$(ScaleBar$(1), mid - qty, qty) = trim(STR$(scale \ 2))
     mid1 = ((wid - 2) \ 2) \ 2
     MID$(ScaleBar$(1), mid1 - qty, qty) = trim(STR$(scale \ 2 \ 2))
     mid2 = ((wid - 2) \ 2) * 1.5
     MID$(ScaleBar$(1), mid2 - qty, qty) = trim(STR$(scale * .75))
     PRINT ScaleBar$(1);

     IF Graf.titlejust = 0 THEN
       titlejust = 2
     ELSE
       titlejust = Graf.titlejust
     END IF
```

```
   text$ = trim(Graf.SubTitle)
   Hpos = Lrr + 1
   Vpos = Ulc + 1
   Clr = Fore
   LOCATE Lrr + 1, Ulc, 0
   PRINT SPACE$(wid);

   'use print line to do the dirty work...
   PrintLine titlejust, text$, wid, Hpos, Vpos, Clr

 END IF

 hgt = Lrr - Ulr                  'determine hieght of graph
 top = Graf.UpperLeftRow + 3       '  "    top
 H = H * 2                        'setup qty of items
 IF H > hgt THEN H = hgt \ 2       'if there are more items than space
 IF H MOD 2 THEN H = H + 1         'make up for odd sizes

 FOR X = 1 TO H STEP 2
  IF X + top + 1 >= Lrr THEN EXIT FOR           'if we run out of room
  P = DataToGraph(X \ 2 + 1)                    'determine value
  Position = CINT(((P + 1&) / scale) * wid) - 1&'adjust bar to scale
  LSET bar$ = STRING$(Position, "█")            'make a bar
  Clr = X                                      'use the next color
  IF Clr = Back THEN Clr = Clr + 1              'if this color is same
  COLOR Clr                                     'as Back then change it
  LOCATE top + X, Ulc + 1                       'Print the bar
  PRINT bar$;
  LOCATE top + X + 1, Ulc + 1                   'print the label
  PRINT DataLabels$(X \ 2 + 1); " -"; STR$(DataToGraph(X \ 2 + 1))
NEXT

CASE 2   'vertical bar graph
'same as above but this time graph goes from left to right - not
' top to bottom.
'
' XXXXXXXXXXXXXX
' ZZZ
' YYYYYYYYY
'


IF LEN(trim(Graf.SubTitle)) THEN

   COLOR Fore                 'set color
   hgt = Lrr - Ulr - 2    'determine maximum hieght
   REDIM ScaleBar$(hgt)       'make a scale bar

   LOCATE ,,0                 'turn of the cursor - saves time

   FOR X = 1 TO hgt                          'print out the
     LOCATE Ulr + 1 + X, Ulc         'scale marker
     PRINT "├";
   NEXT

   'here we determine the positioning of the sub title
   titlejust = 4
  text$ = trim(Graf.SubTitle)
  hgt = (LEN(text$) \ 2)
  Hpos = Ulr + (((Lrr - Ulr) \ 2) - hgt)
  Vpos = Ulc - 1
  Clr = Fore
  PrintLine titlejust, text$, wid, Hpos, Vpos, Clr

  'here, as above in graph type 1, we are building a proportional
  '  scale line
  LOCATE Ulr + 2, Ulc
  PRINT trim(STR$(scale));
  LOCATE Lrr - 1, Ulc
  PRINT trim(STR$(1))
  offset = ((Lrr - Ulr) \ 2) + Ulr
  mid = offset
  LOCATE mid, Ulc
  PRINT trim(STR$(CINT(scale * .5)))
  mid2 = offset * .75
  LOCATE mid2, Ulc
  PRINT trim(STR$(CINT(scale * .75)))
  mid1 = offset * 1.25
  LOCATE mid1, Ulc
  PRINT trim(STR$(CINT(scale * .25)))
END IF

'This is where the difference between the graph types comes in.

hgt = Lrr - Ulr - 1
bot = Lrr
linewid = wid \ H
H = H * linewid
bar$ = STRING$(linewid - 1, "█")
IF (H \ linewid) * linewid > wid THEN H = H MOD wid
cfooter = Ulc
lfooter = bot + 1
count = UBOUND(DataLabels$)

FOR X = 1 TO count
 Z = LEN(trim(DataLabels$(X)))
 IF Z > labelwid THEN
   labelwid = Z
 END IF
NEXT

labelwid = labelwid + linewid

FOR X = 1 TO H STEP linewid

 'this is a scaling algorithm - it determines scale for each item
 ' to graph
 P = DataToGraph(X \ linewid + 1)       'chop this value by physical size
                                  ' of graph

 'determine postion by scaling to 'scale'
 Position = CINT(((P + 1&) / scale) * hgt) - 1&
```

```
      'choose a new color for this item - but not current Back color
           Clr = X \ linewid + 1
      IF Clr = Back THEN Clr = Clr + 1
      COLOR Clr

      'do the bar
      FOR Z = 1 TO Position
       LOCATE bot - Z, Ulc + X + 2
       PRINT bar$;
      NEXT

      'make sure there is enough room for the label on this row - if
      ' not then adjust the positioning & bump up lfooter (row)
      cfooter = foot + Ulc
      IF cfooter + labelwid > wid + Ulc THEN
       cfooter = Ulc
       foot = 0
       lfooter = lfooter + 1
      END IF

      'do the label
      LOCATE lfooter, cfooter
      PRINT bar$; " "; DataLabels$(X \ linewid + 1);
      foot = foot + labelwid
    NEXT

  END SELECT

  PCOPY 1, 0
END SUB

SUB PrintLine (Lrc, text$, wid, Hpos, Vpos, Clr)

  'This sub prints a line of text based on the value of the
  ' variable 'lrc' 'L'eft 'R'ight 'C'entered, where:
  '
  ' LRC = 1 = Right justified    │       XY│
  ' LRC = 2 = Centered           │   XY    │
  ' LRC = 3 = Left justified     │XY       │
  ' LRC = 4 = Vertical           │X        │
  '                              │Y        │


  'set color to print
  COLOR Clr

  'locate & turn of cursor
  LOCATE Hpos, , 0
  'chop text down to fit in given window
  IF LEN(text$) > wid THEN text$ = LEFT$(text$, wid - 2)

  SELECT CASE Lrc

    CASE 1 'right
      LOCATE , Vpos
    CASE 2 'centered
      linewid = wid \ 2
      Vpos = linewid - (LEN(text$) \ 2)
      LOCATE , Vpos + 1

    CASE 3 'left
      Vpos = wid - LEN(text$)
      LOCATE , Vpos

    CASE 4 'vertical
      count = LEN(text$)
      FOR X = 1 TO count
        LOCATE Hpos + X, Vpos, 0
        PRINT MID$(text$, X, 1);
      NEXT
      EXIT SUB

  END SELECT

  PRINT text$;


END SUB

FUNCTION trim$ (totrim$)

     'Quicky function to remove spaces - leading & trailing.

     trim$ = LTRIM$(RTRIM$(totrim$))

END FUNCTION
```

END PROJECT OF THE MONTH PROJECT OF THE MONTH
LONG TERM LONG TERM
LONG TERM PROJECT LONG TERM PROJECT LONG TERM PROJECT LONG TERM PROJECT LONG T

Last month, we developed the skeleton of a working, learning expert system. This month we are going to add a new feature. We will add the capability to process using NODES. A NODE is a processing element. Our expert, from last months issue, has only a single node. It processes and then exits. This is fine for simple operations, but for more complex actions adding another node may be in order. For example, if you are developing a diagnostic expert system to troubleshoot PC or LAN problems, then a single node system might be all you need. For inputs, give the expert the reported problems. For outputs give the expert the symptoms. Now train your expert, and let it go. It would then be capable of reading a symptom, say - strange characters on the screen. Then provide an answer, say - different DOS versions. But, if you want the expert to then recommend another course of action what do you do? Add another node! Using the above example, a two node LAN troubleshooter would use the first nodes answer, different DOS versions, as the input to another node. Maybe this second node would be to recommend a repair activity, say - change PC's DOS to current revision. We are going to modify last months expert to contain multiple nodes. Finally this month, we are going to give our expert a simple user interface - a menu system.

For an example lets make our expert a floppy disk "guru" that recommends which NORTON Utility to use for a given problem.

Build a two node system with node 1 having 6 inputs and 5 outputs. For inputs on the first node use the following.
1) Disk makes grinding noise and says "sector error"
2) Disk makes grinding noise but won't read floppy
3) Error message "Sector not found error" displayed
4) Light doesn't light up on drive
5) DOS says "disk not formatted", but you know it is
6) DOS says "file not found"

For results on the first node and INPUTS to the seconds node, use these. The results of node 1 will be the inputs to node 2. Node 2 then makes it's decisions based on node 1's decisions! Be sure to enter them EXACTLY THE SAME - INCLUDING CAPITALIZATION, SPELLING AND PUNCTUATION.

1) Boot sector damaged
2) Diskette damage
3) File erased
4) Disk drive not connected or broken
5) Disk unformatted

On node two use for the INPUTS, the above 5. For OUTPUTS use these:

1) Use NORTON DISK DOCTOR (NDD)
2) Use NORTON DISK TEST (DT /M)
3) Use NORTON Safe Format (SF)
4) Use NORTON Quick Unerase (QU)
5) Check disk drive cabling

Exercise the expert until it now recognizes and responds correctly. Then you will have an expert system that given some set of inputs, produces not only a conclusion but provides remedial action also! You can add as many nodes as you want (limited by memory) and cross-link as many inputs and outputs as you want. I have added code to make the expert skip asking any question twice, this is nice for a 'smarter looking' expert.
Have fun!

After all that work entering variables and training the expert it's sad to lose all that data! So next month we are going to add a save to disk or load from disk option. This save to disk option will be in the form a file composed of all the arrays and variables of the system. The code to do this is of use not only in this expert system, but also in any other program where you want to save arrays to disk.

```
' -----------------------------------------------------------------
'
'(C)Copyright 1990 Marquis Computing Inc. All rights reserved.
'You may use this program for anything or any purpose including inclusion
'into programs you write BUT you cannot sell this source code. Written by
'Hank Marquis.

DEFINT A-Z
```

```
CLS

' To make this a little easier for you, below I have stated the name &
' function of each array and variable .
'
'  VARIABLE              NAME                     FUNCTION
'  --------              ----                     --------
'  MN           NUMBER OF NODES          THE MAXIMUM NUMBER OF NODES
'  MV           NUMBER OF VARIABLES      THE MAX NUMBER OF VARIABLES
'  MR           NUMBER OF RESULTS        THE MAX NUMBER OF RESULTS
'  HI           HIGHEST RESULT           POINTS TO THE MOST LIKELY NR()
'  D            HIGHEST RESULT           HOLDS LAST BEST GUESS
'
'  FV(X,Y)      FLAG REGISTER            FOR THE CURRENT (X,Y) THIS
'                                        REGISTER HOLDS WHETHER OR NOT
'                                        THIS VARIABLE IS USED ON ANOTHER
'                                        NODE.  IF SO, THE EXPERT WILL
'                                        ONLY QUERY FOR IT ONCE.
'
'  MV(X)        MAX VARIABLE REGISTER    HOLDS THE MAX VARIABLES FOR X.
'  MR(X)        MAX RESULTS REGISTER     HOLDS THE MAX RESULTS FOR X.
'  NR$(X,Y)     NUMBER OF RESULTS        HOLDS THE CURRENT RESULT FOR
'                                        (X,Y).
'  R(X,Y,Z)     RULE ARRAY               HOLDS THE RULE FOR EACH (X,Y,Z).
'  D(X,Y)       DECISION ARRAY           HOLDS THE CURRENT BEST GUESS
'  NV(X,Y)      NUMBER OF VARIABLES      POINTS TO THE VARIABLE FOR CASE
'               POINTER.                 (X,Y).
'  NV$(X,Y)     RESULT REGISTER          FOR (X,Y) HOLDS THE ACTUAL RESULT.
'
' A little menu action

CLS
PRINT "Multi-nodal neural network - version 1.0  (C)1990 Marquis Computing"
PRINT STRING$(80, "-");
PRINT
PRINT " 0 - Define expert - create or change expert system paramaters"
PRINT " 1 - Display rules - show neural network matrix"
PRINT " 2 - Exit program  - quit expert"
PRINT " 3 - Run expert    - cycle the expert system"
PRINT STRING$(80, "-");
VIEW PRINT 10 TO 25

'------------------------------ MENU -------------------------------------

DO

CLS
a$ = UCASE$(INPUT$(1))
X = ASC(a$)

IF X > 48 AND NN = 0 AND X <> 50 AND X <> 52 THEN X = 255
'if we havent defined an expert yet it is kind of hard to use it!
' Set X = 255 to invoke a simple alarm

SELECT CASE X
```

```
CASE 48 'general information collection.
 PRINT

 INPUT " How many nodes"; NN
 INPUT " Maximum number of variables"; NV
 INPUT " Maximum number of results"; NR

 REDIM F(NV, NN), MV(NN), MR(NN), NR$(NR, NN), R(NV, NR, NN), D(NR, NN)
 REDIM NV(NV, NN), NV$(NV, NN)

 FOR H = 1 TO NN

 CLS
 PRINT "How many variables at node"; H;
 INPUT ; MV(H)
   IF MV(H) > NV THEN
     'a little bounds checking action to save
     ' grief later on.
     PRINT " <-entry out of range, setting to"; NV
     BEEP
     MV(H) = NV
   END IF
 PRINT
 FOR I = 1 TO MV(H)
   PRINT "    Enter Node"; H; "variable"; I; ": ";
   LINE INPUT ""; NV$(I, H)
 NEXT 'I

 CLS
 PRINT "How many results at node"; H;
 INPUT ; MR(H)
   IF MR(H) > NR THEN
     'a little bounds checking action to save
     ' grief later on.
     PRINT " <-entry out of range, setting to"; NR
     BEEP
     MR(H) = NR
   END IF
 PRINT
 FOR I = 1 TO MR(H)
   PRINT "    Enter node"; H; "result"; I; ": ";
   LINE INPUT ""; NR$(I, H)
  NEXT 'I

NEXT  ' H

CASE 49
 PRINT
 FOR H = 1 TO NN
   PRINT "Node"; H
   FOR I = 1 TO NV
    FOR J = 1 TO NR
     PRINT R(I, J, H); " ";
    NEXT 'J
   PRINT
   NEXT  'I
   PRINT
  NEXT    'H

  PRINT "Touch any key to continue."
  SLEEP

 CASE 50
  END
 CASE 51
  CLS
  D = 0
  FOR H = 1 TO NN
   FOR I = 1 TO MR(H)
    F(I, H) = 0
    D(I, H) = 0
   NEXT ' I
  NEXT  ' H
  GOSUB Expert

 CASE 52


 CASE 255 'alarm
   PRINT
   PRINT "Can't execute command."
   PRINT
   PRINT "You must define an expert before you can use it."
   PRINT "Select option 0 to define an expert."
   PRINT
   PRINT "Touch any key to continue."
   BEEP
   SLEEP
   dummy$ = INKEY$

END SELECT

LOOP

END

Expert: 'START OF ENGINE ------------------------------------------------

'Here is the code to support multiple nodes. It is used in combination
' with changing the loops to support an added dimension.

FOR H = 1 TO NN           'Repeat for Number of Node - NN

'MAIN DATA ENTRY LOOP  ------------------------------------------------

FOR I = 1 TO MV(H)        'Repeat for MV times, for node H

IF F(I, H) = 1 THEN

 'I have added a Flag register. If this variable is present on another node
 ' then F(I,H) = 1. If it is '1' then we skip asking about it. This makes
 ' our expert more 'intelligent'. If flag register is '1' then skip as '1'
 ' means variable NV$(x,xx) exists on another node and we don't need to ask
 ' about is every time.

ELSE 'ask about the variable

 'if this is the first time for a variable or it is unique (i.e., not a
 'variable on any other node) then ask if it is true or false

 Done = 0 'Done is the loop exit flag. When we are done with this loop,
          ' we set Done to a one. (no pun intended)

 DO

  'get user input. If out of range then repeat. If done, set loop exit
  ' flag Done to 1.

  PRINT "Is node"; H; "variable"; I; "'"; NV$(I, H); "' [T]rue or [F]alse"
  YN$ = UCASE$(INPUT$(1))

  SELECT CASE YN$
    CASE "T"
      NV(I, H) = 1
      Done = 1
    CASE "F"
      NV(I, H) = 0
      Done = 1
    CASE ELSE
      BEEP
      PRINT " Please enter [T] or [F]alse"
      Done = 0
  END SELECT

 'share the input with all other nodes, if another node uses this variable.
 ' This is why we don't need to ask for the same variable on multiple nodes

 FOR HH = H + 1 TO NN              'from this node to the last node
  FOR II = 1 TO MV(HH)             'for all variables for node
   IF NV$(I, H) = NV$(II, HH) THEN 'if variable NV$(x,xx) is on any other
     NV(II, HH) = NV(I, H)         '  node, set the flag register F(x,xx)
     F(II, HH) = 1                 '  and NV(x,xx) to true (1)
   END IF
  NEXT 'II
  NEXT 'HH

LOOP UNTIL Done = 1    'main data input loop

END IF 'end of...asking about variables



NEXT 'I

'BUILD A RULE -------------------------------------------------------

CLS
PRINT "Working..."
```

```
FOR I = 1 TO MV(H)              'for maximum number of variable for node H
 FOR J = 1 TO MR(H)             'for maximum number or results at node H
  D(J, H) = D(J, H) + NV(I, H) * R(I, J, H)    'perform rule
 NEXT 'J
NEXT 'I

'MAKE AN EDUCATED GUESS! ---------------------------------------------

FOR I = 1 TO MR(H)                   'for maximum number or results at node H
 IF D(I, H) > D OR D(I, H) = D THEN '  is D(x,y) = to 1 or -1 ?
   D = D(I, H)                      '  if 1 OR -1 then it is best guess
   HI = I
 END IF
NEXT 'I

'ASK IF IT'S A CORRECT ASSUMPTION -------------------------------------

CLS
PRINT "Is the answer "; NR$(HI, H); "? [Y]es or [N]o"
a$ = UCASE$(INPUT$(1))

IF a$ = "Y" THEN             'we got it right! hooray!
FOR HH = H + 1 TO NN        'share the positive result with all other nodes,
 FOR II = 1 TO MV(HH)       ' if the result is an input on any other node.
  IF NR$(HI, H) = NV$(II, HH) THEN
    NV(II, HH) = 1
    F(II, HH) = 1
  END IF
 NEXT 'II
NEXT  ' HH

ELSE 'we got it wrong, sigh, lets adjust the rules (learn)---------------

FOR I = 1 TO MR(H)              'DISPLAY ALL THE POSSIBLE RESULTS
  PRINT I; " "; NR$(I, H)
NEXT 'I

PRINT "Which result number was it";    ' SELF EXPLANATORY
B = VAL(INPUT$(1))
PRINT

FOR I = 1 TO MR(H)
 IF D(I, H) > D OR D(I, H) = D AND I <> B THEN
   FOR J = 1 TO MV(H)
     R(J, I, H) = R(J, I, H) - NV(J, H)
   NEXT 'J
 END IF
NEXT 'I

FOR J = 1 TO MV(H)
  R(J, B, H) = R(J, B, H) + NV(J, H)
NEXT 'J

END IF 'end of...we got it wrong or right

'loop for next node ------------------------------------------
```

```
NEXT  'H

'return when finished ----------------------------------------

RETURN
```

END LONG TERM LONG TERM
THE BASICS THE BASICS
THE BASICS THE BASICS THE BASICS THE BASICS THE BASICS THE BASICS THE BASICS

This month we are going to talk about how BASIC and DOS stores numbers. The reason for this is three-fold. First, I never have seen a decent explanation of BASIC numbering or DOS numbering. Second, the theme this month seems to be on numeric manipulation (GRAPHING, Binary and all that). Third, we need to understand this stuff. BASIC uses several number formats. In BASIC you can have integers, long integers, single precision and double precision numbers. An example BASIC number representation follows.

X% integer
X& long-integer
X# double precision
X! single precision

Each of these number has it's purposes. An integer is a single 16 bit number in the range of -32768 to +32767. BASIC designers felt that it was better to give us a range of negative to positive than to limit us to positive numbers. Internally, QuickBASIC stores numbers as binary two's complemented format. Don't be scared! This just means that the most significant or high order bit is a 1 for negative numbers and a 0 for positive numbers! In truth there are really still 65536 possible numbers.

Simple integers like A% and B%, are stored this way. If you try to assign an integer more than it's highest or lowest value you will cause a BASIC error and stop your program cold. There are other types of numbers available to us in BASIC - these are called floating point (single precision and double precision) and long-integers. Long integers (X&) are much like integers in that they are stored in memory in binary two's complement format. The difference is that they are 32 digits long - thus allowing a much greater range of numbers. In either case integers are WHOLE numbers - that's the key. If you want to do decimals of fractions then you must get creative with integers or switch to floating point math.

Now, floating point math.

Floating point numbers (can) have a decimal part to them. Floating point numbers are referenced in BASIC by X# or Y!. The # and the ! symbol indicate a floating point number. Double precision uses the # and single precision uses the ! symbol. What are they? Well, on the surface they are BIGGER number holders! Internally, QB stored single precision number with a SIGN, a MANTISSA and an EXPONENT. QB uses the IEEE format for storing these numbers whereby single precision (X!) uses 4 bytes and double precision (X#) uses 8 bytes. That is why last months LOOP optimization examples showed such a dramatic increase in performance. To manipulate

X# (double precision floating point) QB needs to operate on 8 bytes - to operate on X% (integer, 1 byte) QB only manipulates 1 byte.

Below is a chart showing the ranges supported by QB.

TYPE	SYMBOL	RANGE	DECIMAL POINTS
Integer	%	-32,768 to +32,767	N/A
Long Integer	&	-2,147,483,648 to +2,147,483,647	N/A
Single Precision	!	-3.402823^+38 to +3.402823^+38	7
Double Precision	#	-4.940655^-324 to +4.940655^+324	15 or 16

Just remember - if you can use an integer or a long integer your program will be VERY much quicker and can compile out smaller. At times it's is trying to write code with this in mind, but trust me, in the long run your program will be smaller and faster - and isn't that what it's all about!

Now lets talk about something called BCD - Binary Coded Decimal. That is what DOS stores most numeric data as.

What is it?
Well, for example lets take the time entry from the BPB (see last months issue for detail on the BPB). DOS stores the time as a two character string, lets say characters AB. AB are really two ASCII characters - each of which has a number representing it. Example A=65, B=66. This is called Binary Coded Decimal. DOS would store the time as AB - meaning 65,66.

```
   [    2nd byte 'B'    ]  [    1st byte 'A'    ]
   15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0
   h  h  h  h  h  m  m  m  m  m  m  xx xx xx xx xx

   hh=binary number of hours (0 to 23)
   mm=binary number of minutes (0 to 59)
   xx=binary number of 2 second increments
```

To read the time DOS stores (or any other similarly encoded data) we first read the data into a string of the correct length:

```
 TimeString$ = SPACE$(2)
```

or extract it from the file directory entry

```
 TimeString$ = MID$(entry$, 23, 2)
```

Now we have a 2 character string which represents our number. Lets say the string is 'AB'. This is really:

```
  ASCII value of B * 1 + ASCII value of A * 256
```

In BASIC we could write

```
 byte1# = ASC(MID$(TimeString$, 1, 1))
 byte2# = ASC(MID$(TimeString$, 2, 1))
```

Then we add them up

```
 Num# = (byte2# * 256) + Byte1#
```

Num# now holds the NUMBER that was coded into TimeString$! Check out the code for BCD conversion in the program below. In general we could continue with our TimeString$ by convert it from BCD into binary and then extracting the hours, minutes and seconds. The routine BCDtoNum takes up to a 4 character BCD string and returns it's value - which then may be passed to ToBin - which takes a number and returns a binary string. It works just like the above code fragment.

Also included this month are GetBit returns the value of a single bit in a 16 bit number, SetBit turns any bit in an 8 or 16 bit number on or off. See the code for more information & comments on these routines. Also see this months ADVANCED section.

Not only does DOS do this, but dBASE and most other database package store numbers in this fashion! If you know the record length offset into a database file you can use these routines to return record count, length and all that good stuff - we'll save dBASE access for later. In short, you just wont find NUMBERS on DOS disk - you will find characters representing numbers though! (QB however does store floating point numeric data as all 4 or 8 bytes.) When you get into DOS the operating system though, things change. DOS likes integers specifically! Most DOS calls you see me writing in this mag use integers. And that brings up an interesting point - if DOS likes to talk in integers (-32K to +32K) then how do you indicate an number higher than 32,767?! Well, you make it a negative number! As the following code shows.

```
DEFINT A-Z
IF X& > 32,767 THEN
  X = X& - 65536
END IF
```

Now that means that -32768 is really 32,768, -32,767 is really 32,769 and so on! Neat huh? But you what? That's how you really do it! That's all for this month, and that's how BASIC and DOS do numbers -HM

END THE BASICS THE BASICS

ADVANCED BASIC ADVANCED BASIC
ADVANCED BASIC ADVANCED BASIC ADVANCED BASIC ADVANCED BASIC ADVANCED BASIC ADV

This segment is dedicated to an in-depth study and application of an advanced programming topic. Last month we built a routine to return the Bios Parameter Base (BPB) from DOS. GetDOSBoot told us all about the disk drive and the file allocation system. This month we are going to read a files directory entry to get vital information. GetFileInfo will return the files date, time size and attributes. Then, SetFileAttr will let us change the files attributes of Hidden, System, Archive or Read-Only. Again, these routines are part of a larger file unerase program which we are building. Also included this month are two utilities to get the current drive and disk from DOS. These are used by the SetFileAttr sub program.

This month some adidtional routines needed are introduced. These routines are needed to aid us in binary manipulation and Binary Coded Decimal (BCD)

operations. Six routines ToBin, ToNum, BCDtoNum, NumtoBCD, SetBit and GetBit
allow the basic programmer total control over bit level and BCD type
operations.

For information on BCD see this months BASICS section.
In any event, these routines perform bit level manipulation and
are going to be needed by our unerase program. So we are introducing
them here. This program is useful in it's own right for other purposes
so feel free to 'cut-and-paste'!

FILEINFO.BAS is the program for this month. It demonstrates setting up
a Disk Transfer Area, pointing it to DOS and also accessing DOS services.
It also introduces vital low-level numeric manipulation processes -
Binary Coded Decimal and bit manipulation. The number manipulation routines
are valuable in other programs as well.

Use the Cut segment command from the main utilities menu to save this
file to disk. Give it a name like FILEINFO.BAS so you can keep them
straight. When you load this into BASIC, delete all of the text lines
above. FILEINFO has a self running demo of the routines. Just load it
and run it!

To run this program start QB like this

QB /l QB - this loads the library supporting the call interrupt
function that we are using.

The program starts immediately below

```
'Start of program----------------------------------------------------
'
'(C)Copyright 1990 Marquis Computing Inc. All rights reserved.
'You may use this program for anything or any purpose including inclusion
'into programs you write BUT you cannot sell this source code. Written by
'Hank Marquis. revised 10/18/90.


DEFINT A-Z

'build InterruptX call type
TYPE RegTypeX
 AX AS INTEGER
 BX AS INTEGER
 CX AS INTEGER
 DX AS INTEGER
 BP AS INTEGER
 SI AS INTEGER
 DI AS INTEGER
 Flags AS INTEGER
 ES AS INTEGER
 DS AS INTEGER
END TYPE

'build file info type
TYPE FileType
 Nam AS STRING * 8
 Ext AS STRING * 3
 FullName AS STRING * 13
 attr AS INTEGER
 TimeStamp AS STRING * 8
 DateStamp AS STRING * 8
 size AS DOUBLE
 label AS INTEGER
 subdir AS INTEGER
 readonly AS INTEGER
 Hidden AS INTEGER
 Sys AS INTEGER
 archive AS INTEGER
END TYPE


DIM SHARED Regs AS RegTypeX, OutRegs AS RegTypeX
DIM SHARED FileI AS FileType

DECLARE SUB Interrupt (h, InRegs AS ANY, OutRegs AS ANY)


'These two are used to Get/Set the attributes of a file
DECLARE SUB GetFileInfo (FileName$)
DECLARE SUB SetFileAttr (FileName$)

'these two are used to convert from/to a binary number
DECLARE FUNCTION ToBin$ (NumToChange&)
DECLARE FUNCTION ToNum& (B$)

'these two convert to/from BCD
DECLARE FUNCTION BCDtoNum# (BCD$)
DECLARE FUNCTION NumToBCD$ (Num#)

'auxiallry routines used by this progam, interesting and useful in thier
' own right! These two get the default drive & directory from DOS
DECLARE FUNCTION GetDrive$ ()
DECLARE FUNCTION GetDir$ (Drive$)

'these two let you turn any bit on or off in any 16 bit long integer or
' any eight bit integer
DECLARE FUNCTION GetBit (Byte&, Bit%)
DECLARE SUB SetBit (Byte&, Bit, Value)

'inverts a number from 0 to 1 or 1 to 0
DECLARE FUNCTION INV (Num)

CLS
 '--------------------------------------------------------------
 'this demo shows you the use of the GetBit and SetBit
 ' routines.

 CLS

 DO

  LOCATE 25, 1:
```

```
 PRINT "Enter -1 to exit.";

 LOCATE 5, 6
 PRINT ToBin(ByteArray&)
 PRINT
 PRINT "     number ="; ByteArray&; "              ";
 PRINT

 'get a bit to turn on or off
 LOCATE 1, 30
 PRINT "   "
 LOCATE 1, 1
 INPUT "Enter a number from 0 to 15: ", Bit
 IF Bit < 0 THEN EXIT DO      'book on -1
 IF Bit > 15 THEN Bit = 15    'fix up so we don't go over the top

 'setup print state of 'Bit'
 LOCATE 3, 6
 PRINT "Bit ";
 PRINT USING "##"; Bit;
 PRINT " is now : ";

 'get & print the state of 'Bit'
 IF GetBit(ByteArray&, Bit) = 0 THEN
   PRINT "on "        'use on as this is a toggle - it WILL be ON now
   Value = 0          ' even though RIGHT now it is off!
 ELSE
   PRINT "off"        'use on as this is a toggle - it WILL be OFF now
   Value = 1          ' even though RIGHT now it is ON!
 END IF

 'turn it on/off - a toggle
 SetBit ByteArray&, Bit, INV(Value)



LOOP

DO: LOOP UNTIL INKEY$ = ""  'cheap way of blowing out the keyboard buffer
'-----------------------------------------------------------
'this demo shows you the use of the low level NumToBCD and
' BCDToNum routines.

CLS



PRINT "This is a demo of BCDtoNum - it converts a Binary Coded Decimal"
PRINT "into a number. "
PRINT
LINE INPUT "Enter up to four characters (your name): "; X$
X$ = LTRIM$(RTRIM$(X$))
IF LEN(X$) > 4 THEN X$ = LEFT$(X$, 4)

PRINT X$; " coverted into decimal is ";
PRINT USING "#,###,###,###"; BCDtoNum(X$)

PRINT
```

```
 PRINT
 INPUT "Enter a number less than 4,294,967,296 : ", X#
 PRINT X#;
 PRINT " converted to a BCD charatcer string is : "; NumToBCD(X#)

 SLEEP
 DO: LOOP UNTIL INKEY$ = ""  'cheap way of blowing out the keyboard buffer

 '----------------------------------------------------------
 'this demo shows you the use of the low level ToBin and
 ' ToNum routines.

 CLS
 PRINT "This is a demo of ToNum and ToNum"
 PRINT
 INPUT "Enter a number : ", Num&        'Num& is number to convert

 in! = TIMER                            'I'm gonna show off a little...

 X$ = ToBin$((Num&))                   'X$ is the string '00000101010101'
 n& = ToNum&(X$)                       'N& = Num& means it all works

 Outt! = TIMER

 PRINT
 PRINT "ToBin ---> ";
  PRINT USING "##,###"; Num&;
  PRINT " = "; X$

 PRINT "  and..."

 PRINT "ToNum ------------> "; X$; " = ";
  PRINT USING "##,###"; n&
  PRINT

 IF Num& = n& THEN
  PRINT "Hey! It works, how about that! ";
  PRINT "And in ";
  PRINT USING ".####"; Outt! - in!;
  PRINT " seconds - now that's fast!"
 ELSE
  PRINT "Oops. Better enter a number less than 65535!"
 END IF

 SLEEP
 DO: LOOP UNTIL INKEY$ = ""  'cheap way of blowing out the keyboard buffer

'----------Read file info--------------------------------------------
'this code gets a files associated information from it's DOS file entry
'It sets up a disk transfer area (DTA) using DOS INT21 func 1AH to point
'to an ASCIIZ string. The string gets the file information put into it
'by DOS using INT21 function 4EH. Enter any file name, extension or
'anything else. You can read sub-dirs, files etc..

PRINT
PRINT "Following is a demo of GetFileInfo."
```

```
PRINT
PRINT "Enter any valid file. You may use path and wildcards [*?.:\]"
PRINT "For example *.EXE find the first file with .EXE, in the current"
PRINT "directory."
PRINT

LINE INPUT "File name : "; FileName$           'ask for a file name

FileName$ = LTRIM$(RTRIM$(FileName$))
IF FileName$ = "" THEN
  FileName$ = "C:\IBMBIO.COM"                   'I put this here cause I
END IF                                        '  got tired entering
                                             '  filenames.

 GetFileInfo FileName$                           'do it

 PRINT "  Base name      : "; FileI.Nam              'print results
 PRINT "  Extension      : "; FileI.Ext
 PRINT "  Time stamp     : "; FileI.TimeStamp
 PRINT "  Date stamp     : "; FileI.DateStamp
 PRINT "  File size      :"; : PRINT USING "###,###"; FileI.size
 PRINT "  Read only flag :"; FileI.readonly
 PRINT "  Hidden flag    :"; FileI.Hidden
 PRINT "  System flag    :"; FileI.Sys
 PRINT "  Archive        :"; FileI.archive

 SLEEP

 '------------------------SetFileAttr--------------------------------
 'the following code can be used to change the attributes of any file.
 ' use it as shown below. It used the same FileI. type array as above
 ' only this time - YOU set the FileI.xxxx value to a 1 if you want that
 ' attribute to be ON or a 0 to turn that attribute OFF.
 '
 ' FileName$ = "box.exe"       'file to change - with or without path
 ' FileI.Hidden = 1            'hide filename$
 ' FileI.Sys = 1               'make it a 'system file'
 ' FileI.readonly = 0          'make it read or write
 ' FileI.archive = 0           'turn archive off
 '
 ' SetFileAttr FileName$       'call the sub & do it to it

FUNCTION BCDtoNum# (BCD$)

 'converts up to a 4 character Binary Coded Decimal (BCD)
 ' string into a number. The maximum number is 4,294,967,295
 '
 'example A#=BCDtoNum(LEFT$(A$,#))
 '

 'parse out the positions & assign the bytes
 IF LEN(BCD$) THEN Byte1# = ASC(MID$(BCD$, 1, 1))
 IF LEN(BCD$) > 1 THEN byte2# = ASC(MID$(BCD$, 2, 1))
 IF LEN(BCD$) > 2 THEN byte3# = ASC(MID$(BCD$, 3, 1))
 IF LEN(BCD$) > 3 THEN byte4# = ASC(MID$(BCD$, 4, 1))
 'add 'em all up
 Num# = (byte4# * 16777216) + (byte3# * 65536) + (byte2# * 256) + Byte1#

 'assign the function
 BCDtoNum = Num#

END FUNCTION

FUNCTION GetBit (Byte&, Bit)

  'returns the value of 'Bit' in 'Byte&'. Byte& is a long integer,
  ' meaning that Bit can be from 0 to 15.

  GetBit = (Byte& \ (2 ^ Bit)) AND 1

END FUNCTION

FUNCTION GetDir$ (Drive$)

 'This function returns the currently active path from DOS, like
 ' C:\SOFTIPS\NEW

 'Drive must be a number where 0=default, 1=A etc.,
 IF Drive$ = "" THEN
   Drive = 0   'default
 ELSE
   Drive = ASC(UCASE$(LEFT$(Drive$, 1))) - 64
 END IF

 'fix up incase of some invalid drive passed to routine
 IF Drive < 0 OR Drive > 26 THEN Drive = 0

 'make a sratch buffer for DOS to load with the drive & path
 Scratch$ = SPACE$(64)

 Regs.AX = &H4700                      'get current directory
 Regs.DX = Drive                      'use Drive number
 Regs.DS = VARSEG(Scratch$)            'point to scratch
 Regs.SI = SADD(Scratch$)              ' "    "    "
 Interrupt &H21, Regs, OutRegs         'call DOS

 'parse out the drive path
Path$=LEFT$(Drive$,1)+":\"+MID$(Scratch$,1,INSTR(Scratch$," ")-2)

 'clean off any leading/trailing blanks & set the function
 GetDir$ = LTRIM$(RTRIM$(Path$))

END FUNCTION

FUNCTION GetDrive$

'Returns the default or current drive
Regs.AX = &H1900                              'get drive
Interrupt &H21, Regs, Regs                     'do it to it

'fix it up the way we like to make it a D: or C: or A: or whatever
```

```
GetDrive$ = CHR$((Regs.AX AND &HFF) + 65) + ":"


END FUNCTION

SUB GetFileInfo (FileName$)

'-Setup new DTA for this file read----------------------------------

 'Need to build a place for DOS to put the information it will return
 ' to us. This is refered as the Disk Transfer Area, DTA.
 ' DOS fills in DTA$ when we call the &H4E function below.

 DTA$ = SPACE$(64) + CHR$(0)          'DOS work area - ASCIIZ

 Regs.AX = &H1A00                     'set DTA
 Regs.DX = SADD(DTA$)                  'set DX pointer to DTA$ ASCIIZ

 Interrupt &H21, Regs, Regs           'call it...

 IF Regs.Flags AND 1 THEN EXIT SUB     'error...

 'Disk transfer area is now setup pointing to DTA$

'-Now read the file name----------------------------------------------

 F$ = FileName$ + CHR$(0)

 Regs.AX = &H4E00                     'find first match
 Regs.CX = &HFF                      'FF = attribute of anything
 Regs.DX = SADD(F$)                   'pointer to filename
 Interrupt &H21, Regs, Regs           'call it...

 IF Regs.Flags AND 1 THEN EXIT SUB    'error occured

 'Now DTA$ holds the data from DOS about F$

 '-Now parse out FCB -------------------------------------------------
 ' FCB - File Control Block holds file information in DOS's 'mind'

  'reset everything to zero, then change'em if needed
  FileI.readonly = 0
  FileI.Hidden = 0
  FileI.Sys = 0
  FileI.archive = 0
  FileI.label = 0
  FileI.subdir = 0

 'figure file base name that DOS uses - 8 bytes located at offset 2 in FCB
  FileI.Nam = MID$(DTA$, 2, 8)

 'figure file extension that DOS uses - 4 bytes located at offset 10 in FCB
  FileI.Ext = MID$(DTA$, 10, 4)

  'figure full name from DOS - 13 bytes located at offset 31 in FCB
  FileI.FullName$ = MID$(DTA$, 31, 13)
```

```
'figure file attribute(s) - 1 byte located at offset 22 in FCB
Atr = VAL(HEX$(BCDtoNum(MID$(DTA$, 22, 1))))
FileI.attr = Byte1&

'get the file attributes
IF Atr >= 20 THEN
 FileI.archive = 1
 Atr = Atr - 20
END IF
IF Atr >= 10 THEN
 FileI.subdir = 1
 Atr = Atr - 10
END IF
IF Atr >= 8 THEN
 FileI.label = 1
 Atr = Atr - 8
END IF
IF Atr >= 4 THEN
 FileI.Sys = 1
 Atr = Atr - 4
END IF
IF Atr >= 2 THEN
 FileI.Hidden = 1
 Atr = Atr - 2
END IF
IF Atr = 1 THEN
 FileI.readonly = 1
END IF

'figure time stamp - this is cumbersome BUT DOS stores the date & time
' as bit coded binary coded decimal - here we use the ToBin and ToNum
' functions to do bit manipulation. I wanted to work in these routines
' this month - so here they are!

TimeStamp& = BCDtoNum(MID$(DTA$, 23, 2))
TimeStamp$ = ToBin(TimeStamp&)
Hour$ = LTRIM$(STR$(ToNum(LEFT$(TimeStamp$, 5))))
 IF VAL(Hour$) < 10 THEN Hour$ = "0" + LTRIM$(Hour$)
Min$ = LTRIM$(STR$(ToNum(MID$(TimeStamp$, 6, 6))))
 IF VAL(Min$) < 10 THEN Min$ = "0" + LTRIM$(Min$)
Sec$ = LTRIM$(STR$(ToNum(RIGHT$(TimeStamp$, 5))))
 IF VAL(Sec$) < 10 THEN Sec$ = "0" + LTRIM$(Sec$)
FileI.TimeStamp = Hour$ + ":" + Min$ + ":" + Sec$

'figure date stamp - this is cumbersome BUT DOS stores the date & time
' as bit coded binary coded decimal - here we use the ToBin and ToNum
' functions to do bit manipulation.

DateStamp& = BCDtoNum(MID$(DTA$, 25, 2))
DateStamp$ = ToBin(DateStamp&)
Year$ = STR$(ToNum(LEFT$(DateStamp$, 7)))
Year = (VAL(Year$) + 1980) - 1900
Year$ = LTRIM$(STR$(Year))
Month$ = LTRIM$(STR$(ToNum(MID$(DateStamp$, 8, 4))))
 IF VAL(Month$) < 10 THEN Month$ = "0" + LTRIM$(Month$)
```

```
  Day$ = LTRIM$(STR$(ToNum(RIGHT$(DateStamp$, 5))))
   IF VAL(Day$) < 10 THEN Day$ = "0" + LTRIM$(Day$)
  FileI.DateStamp = Month$ + "-" + Day$ + "-" + Year$

  'figure file size - 4 bytes at offset 27
  FileI.size = BCDtoNum(MID$(DTA$, 27, 4))

END SUB

FUNCTION INV (Num)
 IF Num = 0 THEN INV = 1 ELSE INV = 0

END FUNCTION

FUNCTION NumToBCD$ (Num#)

  'Given four characters, the highest number possible is
  '4,294,967,295 - more than 4 BILLION! If you need more, then
  ' add a Byte5# etc.

  IF Num# > 4294967295# * 256# THEN
    NumToBCD = "overflow"
    EXIT FUNCTION
  END IF

  BCDNum# = Num#

  byte4# = INT(BCDNum# / 16777216)
    BCDNum# = BCDNum# - byte4# * 16777216

  byte3# = INT(BCDNum# / 65536)
    BCDNum# = BCDNum# - byte3# * 65536

  byte2# = INT(BCDNum# / 256)
    BCDNum# = BCDNum# - byte2# * 256

  Byte1# = BCDNum#

  'set size of field
  IF byte4# > 0 THEN size = 1
  IF byte3# > 0 THEN size = size + 1
  IF byte2# > 0 THEN size = size + 1
  IF Byte1# > 0 THEN size = size + 1

  'setup a buffer to use
  type$ = SPACE$(size)

  'convert to ascii
  IF Byte1# > 0 THEN MID$(type$, 1, 1) = CHR$(Byte1#)
  IF byte2# > 0 THEN MID$(type$, 2, 1) = CHR$(byte2#)
  IF byte3# > 0 THEN MID$(type$, 3, 1) = CHR$(byte3#)
  IF byte4# > 0 THEN MID$(type$, 4, 1) = CHR$(byte4#)

  'assign function
  NumToBCD = type$
```

```
END FUNCTION

SUB SetBit (Byte&, Bit, Value)

  'changes bit 'Bit' in 'Byte&' to the value of 'Value'

  IF Value = 1 THEN
     'turn on a bit
     Byte& = Byte& + (2 ^ Bit)
  ELSE
    'turn off a bit
    Byte& = Byte& - (2 ^ Bit)
  END IF

END SUB

SUB SetFileAttr (FileName$)

'----------------------------------------------------------------------
'This sub uses the FileI.xxx type to turn the file FILENAME$'s attributes
' on or off. This sub uses the GetDrive and GetDir routines to fix up
' a path-less file name before calling DOS. (DOS is wierd that way.)
'----------------------------------------------------------------------

F$ = LTRIM$(RTRIM$(FileName$))         'fix up file name
IF INSTR(FileName$, "\") = 0 THEN      'If no \ then get the drive &
  d$ = GetDir$(GetDrive$)              ' dir
  F$ = UCASE$(d$ + "\" + F$)           ' fix it up nice
END IF

Atr$ = STRING$(16, "0")                'make a blank atr string

IF FileI.readonly = 1 THEN MID$(Atr$, 16, 1) = "1"   'set values
IF FileI.Hidden = 1 THEN MID$(Atr$, 15, 1) = "1"
IF FileI.Sys = 1 THEN MID$(Atr$, 14, 1) = "1"
IF FileI.archive = 1 THEN MID$(Atr$, 11, 1) = "1"

AtrByte& = ToNum(Atr$)                        'convert atr string to
                                       'a number

Regs.AX = &H4301                              'set attributes
Regs.CX = AtrByte&                            'atr value
Regs.DX = SADD(F$)                            'file pointer
Interrupt &H21, Regs, Regs                    'do it to it...

END SUB

FUNCTION ToBin$ (NumToChange&)

 'This function changes a number less than 65535 into a 16 digit binary
 ' number. For example X$ = ToBin$(8) -> '0000000000001000'
 Num& = NumToChange&   'So we don't muck-up the value passed to us. This
                       ' is just polite programming (for your own POM!)

 size = 16
```

```
'Maybe later I will make it work on 32 bit numbers for DOS 4.X ...
                      ' but for now 16 bits if fine DOS 3.3 & down

n$ = STRING$(size, "0") 'Set our prospective number to all '0s'

' ----------------------------------------------------------

FOR X = size TO 1 STEP -1

  'This code below was substituted for a MUCH simpler
  ' N& = 2 ^ X to save time. It bought about .5 second
  ' over using the BASIC ^ arithmetic command!

  SELECT CASE X
    CASE 16
      n& = 65536
    CASE 15
      n& = 32768
    CASE 14
      n& = 16384
    CASE 13
      n& = 8192
    CASE 12
      n& = 4096
    CASE 11
      n& = 2048
    CASE 10
      n& = 1024
    CASE 9
      n& = 512
    CASE 8
      n& = 256
    CASE 7
      n& = 128
    CASE 6
      n& = 64
    CASE 5
      n& = 32
    CASE 4
      n& = 16
    CASE 3
      n& = 8
    CASE 2
      n& = 4
    CASE 1
      n& = 2
  END SELECT

  N2& = (n& \ 2)

  IF Num& <= n& AND Num& >= N2& THEN
    MID$(n$, 17 - X, 1) = "1"        '17 - X 'cause X is from 16 to 1
                                     ' and we want to start at 16...1
                                     ' so 17-16=1; 17-15=2; etc.,
    Num& = Num& - N2&
  END IF

 NEXT

 ToBin$ = n$

END FUNCTION

FUNCTION ToNum& (B$)

 'This function takes a string in the form of '001010' or any other
 ' binary number and converts it into a long integer. The length of
 ' B$ determines where the translation begins. This is good for doing
 ' MID$ or LEFT$ or RIGHT$ extractions from strings. For example the
 ' FAT of a floppy drive uses a 12 bit number - so you read two bytes
 ' and the take the left or right - most 12 bits & convert it into a
 ' number.
 '
 ' For example:
 '
 '    FloppyFAT = ToNum&(ToBin$(MID$(FAT$,FATOffSet,2))
 '

 size = LEN(B$)                'establish the length of the string & hence
                          ' the start count for the loop below

 FOR X = size TO 1 STEP -1     'counting backwards from Size...

   IF MID$(B$, X, 1) = "1" THEN Num& = Num& + (2 ^ (size - X))

   'if the value of the bit is 1 then our number = 2 to the power
   ' of the position of this bit - which is bit Size-X.

 NEXT

 ToNum& = Num&                 'assign it & boogy...

END FUNCTION
```

END ADVANCED BASIC ADVANCED BASIC

THE BOOK OF THE MONTH THE BOOK OF THE MONTH

THE BOOK OF THE MONTH THE BOOK OF THE MONTH THE BOOK OF THE MONTH THE BOOK OF

In this segment we review a book that has to do with programing. This month the book is "QuickBASIC Programmers Toolkit", a book-disk set featuring BASIC functions, routines and some full programs.

Book	: QuickBASIC Programmers Toolkit
Author	: Tom Rugg & Phil Feldman
Publisher	: QUE
Dated	: 1988
Cost	: $39.95
Available	: this copy bought at Software Etc.,

QB Tool kit is a large book which comes with a diskette of programs which are developed in the book. The best thing about the book is it's concept - providing ready to run programs.

The author goes through each program in a rigid style explaining the applications, variables et al. Then lays out the program. The diskette contains many useful programs, most of which are demonstrated. The books sections cover keyboard & screen I/O, printer control, file management, sorting & searching, text manipulation, math and system utilities.

The printer section contains a couple of nice routines for managing LaserJet printers, the keyboard and screen sections let you get or set most options and in general are very well written. You can determine monitor, screen, adapter, memory, key states (control, alt etc) and more.
Fully one half of the book is dedicated to theoretical/scientific concepts. Such topics as differential equations, matrix math and statistics are not every-day, but if you need then you really need them, and here they are.

I personally, really rather prefer a book deal with day-to-day needs of the typical programmer using BASIC - and that in here too! Julian and date manipulation routines and others. Where was this book when I was writing all my routines from scratch?! Oh, well you don't have to! The price is steep, but what you are getting is really a well done book and not one bu several software libraries which are ready to run. Well, almost.

Which brings me to what I didn't like about this book. While the disk concept is great, unfortunately it's implementation here is flawed. The file names on the disk are not standard - they don't carry .BAS or .MAK file names - they are instread .QPT, .SU and .PGM! Nice huh? try loading up those names, and what about make files? Why not use regular old .BAS & .MAK file name? I guess you just can't have your cake and it too. Beyond that, the source code isn't commented and the coding doesn't use any structure or indentation to make reading it easier.

This book does contain many good, useful programs and routines which the novice or advanced BASIC programmer can use immediately (after figuring out the strange file conventions) after loading. Some use DOS calls others don't. Some are small 2 line functions others are hundred line programs. It's a real mixed bag. I feel it is geared more toward the professional programmer or developer as it's business, scientific and sorting sections are not for the everyday user. Still, there is enough here though to satisfy anyone who shells out the $39.95.

If you are a programmer and can develop these routines yourself - save the money. If you don't want to spend the time or are a new user and just want to 'plug-and-play' then by all means this is the book for you. Just remember - it assumes that you already know a lot about QB, such as functions & sub routines and how to load programs et al.

-HM

END THE BOOK OF THE MONTH THE BOOK OF THE MONTH
SOFTWARE OF THE MONTH SOFTWARE OF THE MONTH
SOFTWARE OF THE MONTH SOFTWARE OF THE MONTH SOFTWARE OF THE MONTH SOFTWARE

In this segment, we review a software program, utility or add-on for BASIC.
This month the QuickPak Professional library from Crescent
Publisher   : Crescent Software
Version tested : 3.1
Dated   : 1990
Cost   : $149.00
Available   : Contact Crescent at: 203-438-5300

QuickPAK is a collection of programs and routines to supplement both QuickBASIC and DOS. QuickPAK comes with over 100 programs and utilities to do everything from getting the DOS version to a complete text editing word processor with word wrap, block commands and much, much, more.

Many routines are BASIC replacements - either replacing directly QuickBASIC commands or offering enhancements over them. For example many routines are designed to circumvent the need to use ON ERROR (see this months FORUM). These replacements let you perform a function - like killing a file and then simply test for the success of the operation. Often a -1 means success and 0 means failure. This lets your programs be as small as they can be.

QuickPAK is written mostly in assembly language, and as such it is small and the routines are really fast. I find the screen save and restore functions of the most benefit, but then again the whole package makes your life easier.

I specially like the fact that Crescent also provides you with the source code for ALL routines - BASIC or assembler. The BASIC code is well written and amply commented. In fact I've made many changes to the core routines with no problems.

QuickPak is divided into sections:

A nice introduction to FUNCTIONS, SUB ROUTINES and programming.
Comprehensive array manipulation far string, number or fixed length arrays.
A DOS section with many substitutes for BASIC as well as must have additions to BASIC.
A section packed with many useful functions - everything from Celsius to Fahrenheit conversion to a complete spread sheet math package!
Menu & Input - all types of menu systems, LOTUS style, various pull-down pop-up and scrolling menu types. Several complete menu programs.
Keyboard & Mouse - complete mouse and keyboard control, including routines to let you get or put characters into the keyboard buffer.
Miscellaneous - A string manager to put strings into 'far' memory, an EMS memory manager module and a whole lot more.
String functions and programs - complete string parsing, trimming managing and many handy routines.
Video functions - saving, painting, restoring and displaying screens.

There is a lot more - the book is about 3 inches tall and it comes on 6 floppies! There are several complete working BASIC programs that you simply load and use. Getting up and running is really fast. This is goodness if I ever saw it!

There are several versions of the package, QuickPAK and QuickPAC Professional. QuickPAK lacks all of the routines found in the Professional version, and for the small difference in money, I think you're better of with QuickPAK Professional. I can't think of any programmer who would not benefit immediately from the use of this package.

By the way, this program (READER.EXE) uses the following routines from QuickPAK Professional - with no modifications!

EDITOR
SCREEN SAVING & RESTORING
DIALOG BOXES
SCREEN PAINTING
MANY FUNCTIONS MAKING A PROGRAMMERS LIFE EASIER!

-HM

END SOFTWARE OF THE MONTH SOFTWARE OF THE MONTH