

FREE EDITION

THE LITTLE BOOK OF SOURCE CODE

PREVIEW VOLUME - VARIABLES, ARRAYS AND DATATYPES



DANIEL FOREMAN

The little book of source code[©]
Preview Volume – Variables, Arrays, and Datatypes.

Written by Daniel Foreman

Copyright © Daniel Charles Foreman 2005

All rights reserved.

No part of this work may be reproduced or used in any form without the explicit written permission of the author.

While every effort has been made to insure that the content of this book is accurate, the author and the publisher will not accept liability or responsibility for any loss or damage resulting from the source code or information contained within this book.

All brand names and products used in this book are the trademarks of their respective companies and have not been Capitalized through this text.

DarkBASIC Professional is produced by The Game Creators Ltd
CodeSurge is a opens source IDE alternative to the DarkBASIC Professional default IDE.

ISBN Number: (Pending Release of Volume one, of pending The Little Book of Source Code series)

Book website and author contact e-mail:

www.dforeman.co.uk
author@dforeman.co.uk

This ebook is presented free of charge, and may be distributed freely in it's unaltered form. All credit, legal notices, and content must remain fully intact when distributing this ebook. This ebook may not be printed for commercial sale without the full written permission of the author. This ebook may be printed once per private user, but only distributed in it's original to others in it's digital PDF format.

Introduction

The little book of source code, is a series in development, that aims to produce smaller, lighter books split up into several volumes for new users. It will start from the bottom upwards, using a series of source code examples to explain how commands work within DarkBASIC Professional. To help promote the series, this e-book sample has been issued free of charge so all users can gain an idea of what to expect, and provide feedback in regards to what they would like to see appear in the upcoming series. We hope to produce a lighter, easier to access series that lets you buy what you need, when you need it. I hope you enjoy the series. If you have any questions please e-mail me at author@dcforeman.co.uk or visit our website at www.dcforeman.co.uk for more information. (Site launch Date 15 July, 2007)

I hope you enjoy reading this book, I certainly enjoyed writing it!

Content:

Variables

- i. What is a variable?
- ii. What types of variables are there?
- iii. How do I make a variable?
- iv. What is the difference between local and global?
- v. Other types of Variables.
- vi. Array's.
- vii. Datatypes.
- viii. Other Commands used in examples.
- ix. Special notes on floats.

Upcoming in volume one, The Core of DarkBASIC...

Decision making

- i. What is boolean?
- ii. How do I use an IF statement?
- iii. What are the symbols of decision making?
- iv. How do I use CASE statements?
- v. Other Commands used in examples.

Plus a reviews and examples on all DarkBASIC Core and Text commands, plus a full guide to the default IDE, and CodeSurge IDE.

Chapter One: Variables

i. What is a variable?

First of all, what is a variable? Simply put, a variable is a piece of data held within the computers memory. It can be used to hold the numbers on a shopping bill, or the result of a calculation. It can be used to hold a name, or an address, or anything else you wish to the computer to remember.

Variables are essential to any program you make in the future, and it is impossible to produce a program that doesn't use a variable of some kind.

ii. What types of variables are there?

There are three primary types of variables, an integer, a float or a string. An integer is a whole number, this can be 1, 2, 35, 65543, or anything else you like. A float is a number format that supports decimal points, for example, 1.1, 2.2, 3443.643 or any other number that is not whole. The final type is a string, a string is basically a collection of characters all stringed into a sequence, for example, James, John, or 25a Brinston Road.

How do I know what type if variable I am using? Any integer value, is a whole number, is represented with a name. This name can be anything you like providing it doesn't clash with an existing command. In DarkBASIC Professional it is easy to tell if it does. If the text turns blue, then that name is already assigned to a command. If however it is black, then it is usable. Also please note that variables are case sensitive. To identify an integer to the system you just have to give a name without any symbols. For example.

```
VariableX = 1  
Print VariableX  
Wait Key  
End
```

To mark a float number, the variable name must end with the # symbol. Informs the computer that it might make room in memory to hold decimal numbers.

```
VariableX = 1.1352  
VariableX# = 1.1352
```

```
Print VariableX
Print VariableX#
Wait Key
End
```

Did you notice that when you run the program, the display shows 1, then a 1.1352, despite having given a decimal number for both variables within the program? This as I mentioned just now, is because the second variable was marked with the # symbol at the end.

The final type of course is a string, this, as with a float, must be marked with it's own symbol. In this case it is a \$ symbol.

```
VariableX$ = "James Oliver lives at 25a Briston Road"
Print VariableX$
Wait Key
End
```

If you were to miss out the \$ symbol, DarkBASIC Professional would return the error message, Cannot perform 'integer' cast on type '\$\$1' at line 1. It is important however to remember that if you type in a float without the # symbol, then no error message will be generated, the result will just be truncated without it's decimal places.

iii. How do I make a variable?

I demonstrated in the last few examples how to hard code variable information into the program. However, variables are not called statics, this is because any variable can be modified, and changed by running instructions in the program. For example.

```
VariableX = 1
VariableX = VariableX + 3
Print VariableX
Wait Key
End
```

You can also accept variables from the user by using the INPUT command.

```
INPUT "Enter a number", VariableX
INPUT "Enter a number to multiply by", VariableY
Print VariableX * VariableY
```

Wait Key
End

However in the same way, that variables were hard coded, each variable must be marked with a symbol to show it's type. That last example will only multiply whole numbers. To accept float numbers you must mark the variables.

INPUT "Enter a number: ", VariableX#
INPUT "Enter a number to multiply by: ", VariableY#
*Print VariableX# * VariableY#*
Wait Key
End

Obviously for the last two examples, we used numbers only, but what if you want to store a string? Again, this is done in the same way but with the \$ symbol after the variables.

INPUT "Enter your name: ", Name\$
INPUT "Enter your Town: ", Town\$
INPUT "Enter a postcode: ", Postcode\$
Print Name\$
Print Town\$
Print Postcode\$
Wait Key
End

But again, just because it's a string doesn't mean the variable is fixed, it is still subject to commands within the program, for example.

INPUT "Enter your name: ", Name\$
INPUT "Enter your Town: ", Town\$
INPUT "Enter a postcode: ", Postcode\$
Name\$ = Name\$ + ", " + Town\$ + ", " + Postcode\$
Print Name\$
Wait Key
End

Did you notice that Name\$ started off with storing a single string, and then once the inputs were recorded Name was altered to hold all inputs with comma's between them?

iv. What is the difference between local and global?

A local variable is not something that is stored permanently. As programs become more and more complex, functions will be used. If you are just starting out in BASIC it is likely that you've not come across functions yet, simply put they are a way of producing your own customized commands. But lets not worry about that too much now, I will explain functions later on in the series, for now it is merely enough that you know they exist. A local variable is something the program will forget as soon as a function is completed. It's a piece of throw away data that as done is job once the function has finished its purpose. If however there is data in the program that will be used many times across many different subroutines and functions then it is called a global variable. As with all things in programming you just inform it that a certain variable must be stored and ready to use for any part of the program and any time. To do this you type in, at the beginning of any program:

Global VariableX

Now the whole # and \$ symbol business is a quick way of marking a variable, however there is another way.

VariableX as float

Will allow float information to be stored within VariableX despite not having a # marker. I recommend the # marker because it allows you to know what type of variable you are using at a glance. To save space you can also setup the global command at the same time:

Global VariableX# as float

v. Other types of variables

There are other types of variables, but mainly this refers to the amount of data that can be stored.

An integer has a plus and negative range of $-2,147,483,648$ to $2,147,483,647$, this is the default for dealing with large and small whole numbers. Other whole number ranges include:

Continued on next page...

DOUBLE INTEGER Range : -9,223,372,036,854,775,808 to
9,223,372,036,854,775,807
BOOLEAN Range : 0 to 1
BYTE Range : 0 to 255
WORD Range : 0 to 65535
DWORD Range : 0 to 4,294,967,295

Byte, word and dword will not accept negative ranges.

Decimal number ranges include:

DOUBLE Float Range : 1.7E +/- 308 (15 digits)

Both accept negative values.

When would I use any of these? Well assigning a range helps to save memory. If for instance you were storing a blue colour value, then the highest you can go is 255, so what is the point in assigning enough memory space to store 2,147,483,647? Also you might want a simple “yes or no” answer, which can be stored as a 0 or 1, so what is the point in having enough space to remember a number of 255?

As with an integer, string or float, you can setup the variables with the AS command. Because double integer's, boolean, byte words and dwords are all whole numbers, to maintain consistency it's best to treat them as if they were integer values. That is I won't add a \$ or # or any other identifying symbol.

VariableV as Double Integer

VariableW as Boolean

VariableX as Byte

VariableY as Word

VariableZ as Dword

VariableV = 9223372036854775807

VariableW = 9223372036854775807

VariableX = 9223372036854775807

VariableY = 9223372036854775807

VariableZ = 9223372036854775807

Print “Double Integer: ”,VariableV

```
Print "Boolean: ", VariableW
Print "Byte: ", VariableX
Print "Word: ", VariableY
Print "Dword: ", VariableZ
```

```
Wait Key
End
```

Interestingly in DarkBASIC 6.6b the boolean value came up with 255, when it should have presented a 1, but never mind. 9223372036854775807 is the maximum readable number, and you will notice that despite the same number being entered each time, different values were printed out in keeping with the maximum lengths I previously described. To setup a Double float it is exactly the same as demonstrated above. Remember though that the program above it just listing local variables, to make sure they are recallable throughout the whole of the program just put Global in front of each variable. For example.

Global VariableX as Double Float

Do you remember me mentioning that variables are called variables because they are changeable? Sometimes you want to create a variable that can't be changed. You do this with the #Constant command.

```
#Constant X 24
#Constant Y "Cheese"
#Constant Z 24.4
```

```
Print X
Print Y
Print Z
```

```
Wait Key
End
```

In that example, notice that there is no equals sign? This isn't a variable so it can't be changed. It is not possible or relevant to assign a constant as an integer, string or float, this is done automatically, and constants will not accept any \$ or # symbols after the name assigned. So #Constant Z# 24.4 will just generate the error "Constant name 'Z#' must only use alphanumeric characters."

If you attempt to change a Constant after it's been set, like this:

```
#Constant X 24
```

```
X=35
```

```
Print X
```

```
Wait Key
```

```
End
```

You will receive the error message “Unknown type found at line 3”. So it is impossible to accidentally change a constants value once set. Constants are useful for storing unchangeable data, such as the speed of light, keyboard scancodes, or another fixed variable. Constants can still be used in mathematical calculations though.

```
#Constant X 24
```

```
Global VariableX as Integer
```

```
VariableX = X
```

```
Print VariableX + X
```

```
Wait Key
```

```
End
```

vi. Array's

Sometimes it is necessary to setup and store large amounts of data, or have the program generate large amounts of data internally. In these cases an array is deployed. To setup an array type in the following:

```
DIM MyArray$(7)
```

The 7 represents how many cells are in the array. Each cell stores information that you can retrieve with a number. It's much like using VariableX1, VariableX2, VariableX3 etc...

continued on next page...

DIM MyArray\$(7) as String

MyArray\$(1) = "Sunday"
MyArray\$(2) = "Monday"
MyArray\$(3) = "Tuesday"
MyArray\$(4) = "Wednesday"
MyArray\$(5) = "Thursday"
MyArray\$(6) = "Friday"
MyArray\$(7) = "Saturday"

For X = 1 to 7
 Print MyArray\$(X)
Next X
Wait Key
End

If you had setup VariableX1, VariableX2 etc you would have had to have typed out Print 7 times. Using similar loops you can manipulate and store data, for instance.

DIM MyArray(399) as WORD

For X = 0 to 399
 *MyArray(X)=X*25-4*
Next X

For X = 0 to 399
 Print MyArray(X)
 Sleep 25
Next X

Wait Key
End

As you can see, you have quickly and easily stored 400 different variables that can be called back at any time for any reason. Because in computers, zero is considered a number, the first stored variable within an array will always be *MyArray(0)* rather than *MyArray(1)*.

vii. Datatypes

Data types is a way of creating a group of variable types that can be used over and over again. This saves retyping code over and over, reducing the amount of source code.

type worlddata

X as integer

Y as integer

Z as integer

normalspeed as float

rotationrate as float

runningspeed as float

endtype

global playerdata as worlddata

global enemydata as worlddata

playerdata.x = object position x(playerobj)

playerdata.y = object position y(playerobj)

playerdata.z = object position z(playerobj)

playerdata.normalspeed = 1.2

playerdata.rotationrate = 0.3

playerdata.runningspeed = 2.7

enemydata.x = object position x(enemyobj)

enemydata.y = object position y(enemyobj)

enemydata.z = object position z(enemyobj)

enemydata.normalspeed = 1.15

enemydata.rotationrate = 0.35

enemydata.runningspeed = 2.6

As you can see from this example, I have typed out a series of variable's and assigned them as an integer or float, and assigned them under the name group name "worlddata" then took that name, "worlddata" and assigned it to a couple of global variables with different names as if it were an integer or float. Because the player, and the enemy chasing the player, both have the same sort of information needing to be processed, but with different values, it becomes useful to setup a datatype rather than retyping the same information over and over. Now any variable that has been assigned to "worlddata" just has to have a .x, .y, .z, .normalspeed, .rotationalrate, or .runningspeed, to automatically

be assigned as an integer or float. I've saved myself lots of typing, and it helps to keep code more compact and easier to understand.

Furthermore this can be applied to an array, so if you had an asteroid field floating in space, each asteroid with it's own speed, vector, rotational data, position and mass, it becomes far more economical to specify that information within an array. All these settings can be produced mathematically as well, so you don't have to type in the data.

```
type worlddata  
  Speed as word  
  Vec as dword  
  X as integer  
  Y as integer  
  Z as integer  
  Mass as dword  
endtype
```

```
Global Dim Astroids(399) as worlddata
```

```
For X = 0 to 399  
  Astroids(X).Speed = RND(255)  
  Astroids(X).Vec = RND(359)  
  Astroids(X).X = RND(4500)  
  Astroids(X).Y = RND(4500)  
  Astroids(X).Z = RND(4500)  
  Astroids(X).Mass = RND(500)  
Next X
```

```
For X = 0 to 399  
  Print Astroids(X).Speed  
  Print Astroids(X).Vec  
  Print Astroids(X).X  
  Print Astroids(X).Y  
  Print Astroids(X).Z  
  Print Astroids(X).Mass  
Sleep 25  
Next X
```

```
Wait Key
```

As you can see, in 28 lines of code, we've produced over 2,400 pieces of data that could potentially be used to control 400 3D objects on the screen.

viii. Other Commands used in examples

Throughout the whole document I've used several commands in my examples to display, and manipulate variables. This section quickly explains each command, however I won't go into too much detail here as I will cover these commands specifically in the future.

The most common is Print, and as you have undoubtedly worked out by now, this is the command used to output any variable to the computer monitor. The correct usage is:

Print variable, variable, variable

You can also print a string to the screen without assigning a string name to the command. You do this by putting quotes around it.

Print "Hello World!"

You can also use Print to carry out mathematical functions with variables and numbers.

*Print 5 + 6 * 2 - 9 / 255.7*

Finally you can mix set data and maths:

*Print "The Answer to 5 + 6 * 2 - 9 / 255.7 is: ", 5 + 6 * 2 - 9 / 255.7*

There is a little more information on this command later on. The INPUT command works in much the same way as Print, however rather than Printing an existing variable, it pauses the program so the user can assign a number to a variable rather than hard coding it in.

The next most commonly used command is:

Wait Key

Simply put, this key stops the program until the user has hit any key on the keyboard, then the program resumes. The next most used command is END. This simply ends the program and exits.

The next would be the FOR and NEXT loop. This is just a quick way of repeating a step a certain number of times. As seen with the Array, it counts between two set numbers separated with the TO command. Any command between the FOR and NEXT commands will be repeated the number of times specified. At the same time, the X is a local variable (remember we talked about global and local?) so it is discarded after the loop is complete. But within the loop it can be used in any calculations make.

The Sleep command pauses the program for a specified amount of time, 1000 units = 1 second. You can also use Wait in the same way, by specifying a number instead of Key.

ix. Special notes of float variables

It is worth knowing that sometimes what is entered and what is stored in memory isn't accurate. For example, if you type in:

```
a# as float  
a# = 33.33  
print a#  
wait key  
end
```

The the result will be 33.3300018311 rather than 33.33. This is something worth looking out for if your future calculations use float's.