A DarkBasic DataBase

By: Phaelax(Phaelax@hotmail.com)

So you want to create a database program in DarkBasic. It's easier than you may think. Nothing more to say in this introduction, so let's get started.

By the end of this tutorial, your database will be able to create, read, and save entries. It will teach you how to use arrays and write to external files, thus creating your own file format.

So what is an array?

An array is an easy to use way of storing related data together. Here's an easy way to show you what that means. Suppose you have the names of 10 people stored into 10 different string variables. To output those names to the screen would look something like this:

```
Print Name1$
Print Name2$
Print Name3$
Print Name4$
Print Name5$
Print Name6$
Print Name7$
Print Name8$
Print Name9$
Print Name10$
```

Now what if you had to display 100 names? That's a lot of very repetitive code to write. Let's turn the above statements into something more simple by using an array.

```
For i = 1 to 10
    Print name$(i)
Next i
```

Those 3 lines of code will do the same as the 10 lines of code above. Now again imagine writing the output for 100 names using the first method. That's 100 lines of code. If you use an array like above, all you have to do to output those 100 names is change that number 10 to a 100. Couldn't be any easier. Now to explain how the above code works. To use an array, you must define it first by using the dim command.

```
Dim variableName(amount)
```

This will make an array with the specified name and desired amount. The amount is how many elements the array can hold. So to store 10 names, the array would have to have a size of at least 10. Think of these boxes as our array with a size of 10. Each box is an element or in other words, a variable. Each box can hold a name.

Name1	Name2	Name3	Name4	Name5	Name6	Name7	Name8	Name9	Name10

To access each element, you have the array variable's name followed by an index number in parenthesis.

```
Name$(4)
```

The index points to which box you want to access. So the above code statement would return whatever Name4 is.

So for what reason do we need to know about arrays to create a database? I'll get to that farther down. An array can hold an type of data that you want. Integers, floats, strings, etc… But the entire array can only hold 1 type of data, meaning you could not make an array that stored integers in some elements and strings in another. For this reason, we'll need to create our own data type and define the array as that type of data.

What is a Type?

In Dark Basic Pro, you can create your own data structures.

```
Type person
    Name as string
    Age as integer
Endtype
```

The above code makes a new data type called **person**. Think of this **person** as like saying **integer** or **string**. The **name** and **age** variables you see defined inside the TYPE structure are like sub variables to the **person**. So to define a variable of type person would look like the following.

```
Me as person
```

Now this is how you would write data to the new variable. Because **Me** now has 2 sub variables as part of its data type, you can't simply say **Me = "Phaelax**".

```
Me.name = "Phaelax"
Me.age = 21
```

Seems simple enough, right? We can define an array as a **person** type as well, and use it in the same way, while following the syntactic rules of an array of course.

```
Dim people(10) as person

People(1).name = "Phaelax"
People(1).age = 21
People(2).name = "Duosoft"
People(2).age = 18
```

Once you have all the data written into the new array, this is how you would recall the data.

```
For i = 1 to 10
    Print people(i).name
    Print people(i).age
Next i
```

Wow, this is easy isn't it. Now for the database, you need to think of what type of data you're going to be using and saving. For this example, let's create a database of friends' addresses. First, you create your new data type that will have a variable to define all parts of the data you need to store such a record.

```
Type friend
        Name as string
        Phone as string
        Address as string
        City as string
        State as string
        Zip as string
Endtype
```

These can all be strings if you want. Now you create your array of friends. Create the array with a big enough size that you're sure you won't reach. I have well under 100 friends I need to keep track of, so a size of 100 is more than plenty.

```
Dim addressBook(100) as friend
```

How to create and access external files?

Now to fill our address book with data. Oh wait, we need a way to easily access these variables from within a program. Let's create something very simple to allow users to enter data into our address book. This should program will prompt the user to either add a new entry into the address book, or quit.

```
Type friend
        Name as string
        Phone as string
        Address as string
        City as string
        State as string
         Zip as string
Endtype

Dim addressBook(100) as friend

rem number of current entrys
index = 0

do
        cls

        input "Would you like to add a new entry? y/n", a$

        if a$ = "y"
                inc index, 1
                input "Enter the name: ", addressBook(index).name
                input "Enter the phone number: ", addressBook(index).phone
                input "Enter the address: ", addressBook(index).address
```

```
                input "Enter the city: ", addressBook(index).city
                input "Enter the state: ", addressBook(index).state
                input "Enter the zip code: ", addressBook(index).zip
        endif

        if a$ = "n"
                rem quit program
                end
        endif
loop
```

There's only one small problem with this code. Every time you run the program, you'll have to re-enter the data. Kind of defeats the whole purpose of an address book. So now we need a way to save the data to be recalled later on. The following code will write all inputted address entries into a text file.

```
filename$ = "testing.book"

if file exist(filename$)=1 then delete file filename$

open to write 1, filename$

write word index

for i = 1 to index
        write string 1, addressBook(i).name
        write string 1, addressBook(i).phone
        write string 1, addressBook(i).address
        write string 1, addressBook(i).city
        write string 1, addressBook(i).state
        write string 1, addressBook(i).zip
next i

close file 1
```

First, we check to see if the file of the specified already exists or not. If it does, then we delete the file, otherwise the **open to write** command will fail. Next, we create the text file used to store our data. To make things easier, the first bit of data we store is the number of entries that were created. This makes it easier when we go to retrieve that data later. Now we go into a simple **FOR** loop and write each address out to the file in order. After we're done writing our data out to the file, we close the file. The next time the user goes to run the database, we need to load up the old data back into the array.

```
if file exist(filename$)=1

        open to read 1, filename$

        read word 1, index

        for i = 1 to index
```

```
                read string 1, addressBook(i).name
                read string 1, addressBook(i).phone
                read string 1, addressBook(i).address
                read string 1, addressBook(i).city
                read string 1, addressBook(i).state
                read string 1, addressBook(i).zip
        next i

        close file 1

    endif
```

First, we check to see if the file exists, otherwise there would be nothing to open, causing an I/O Exception error. Then we read in the number of entries into our appropriate variable. Then comes the **FOR** loop to load the rest of the data into the correct fields.

The next steps in creating a database are up to you. You could create a simple menu, giving the user simple options to choose from.

```
Print "---Main Menu---"
Print "1. Load address book"
Print "2. Save address book"
Print "3. Add new entry"
Print "4. View entry"
Print "5. Quit"
input "Select an option by number", option$
```

Then perform the appropriate action based on the user's response. Look for future installments of this tutorial. Future updates will include searching entries, sorting entries, deleting entries, and the use of array lists instead of a normal array.