(BCA) Hons. - 1st Year

Group – 3

Programming with FoxPro

1.1 What is a FoxPro program ?

A FoxPro program is an ordinary text file containing one or more FoxPro commands, each typed on a separate line. A file containing FoxPro commands is known as a **command file**, or a **program** The standard file extension for programs in FoxPro is **.PRG**.

A line within a FoxPro program normally begins with a command verb, which is a reserved FoxPro word such as USE or INDEX. The command verb is followed by a statement, which can be made up of number of values or conditions that control the effect of the operation performed by the command. Lines of a program can start at the left margin, or they can be indented for easier reading. Lines cannot be longer than a maximum of 256 characters, including spaces.

A command in a program can be written onto multiple lines by ending each line with a semicolon. Indentation can be used to clearly differentiate continuation lines from new commands. FoxPro ignores blank lines in a program, so blank lines can be freely used to separate groups of commands that make up separate functional modules. Extra spaces can be added within a command to make expressions and lists easier to read and edit.

Foxpro programs can range from a simple list of a few commands to dozen of interrelated files containing hundreds of commands. When multiple files of commands are used to perform various tasks, the collection of files is often referred to as an **application**. For example, a number of command files may be designed to process the various transactions necessary within an accounting system. These command files are referred to as an "Accounting Application" running under FoxPro.

All the commands which can be given at the command window form a part of FoxPro language. The FoxPro language supports additional commands that are valid only when used within programs. These commands will not work in the command window, i.e., they are **program-only** commands.

1.2 FoxPro's text editor

FoxPro programs can be created by using any text editor or word processor like Norton's Editor, Wordstar etc. FoxPro has its own built-in text editor that can be used to create and edit programs. This editor is always available and is integrated with the FoxPro windowing environment.

The editor can be opened by using the :

FILE,NEW command If a new command file has to be opened.
(In this case, choose File Type as PROGRAM from the "New" dialog box).

FILE,OPEN command if an already existing command file is to be opened.
(Select the file name from the "Open" dialog box).

OR

By using the Command Window, type

MODIFY COMMAND < program name >
or
MODIFY FILE < program name >

If the MODIFY COMMAND < program name > is used, FoxPro assumes that the file extension is .PRG. The MODIFY FILE < program name > command is used to edit a query, screen or menu program generated by FoxPro.

1.3 Creating our first program

To begin with let us create a very simple program. As an example of a simple programming task, it might be necessary to enter a series of commands that would open the FLIGHTS.DBF table, index it on the field FLIGHT_CODE and display the contents of the table. To do this first type the command :

MODIFY COMMAND FIRST

in the command window. This opens the FoxPro built-in editor.

The program code would be as follows :

```
USE FLIGHTS
INDEX ON FLIGHT_CODE TAG CODE
LIST
USE
```

Note that the program consists of all FoxPro commands which begins on separate lines.

1.3.1 Executing the program

After the program is written the next thing you might want to do is execute it or see its output. A FoxPro program is executed by using the command :

DO <filename>

where "filename" is a text file containing the individual FoxPro commands to be executed.

For example to execute the program FIRST.PRG enter

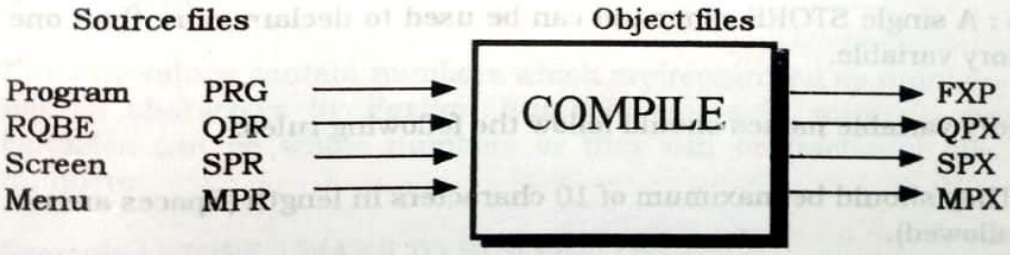
DO FIRST

On running this program, FoxPro executes each of the commands stored in the text file, one after the other. and displays the results on the screen.

1.3.2 What is compiling ?

Before FoxPro can execute the instructions in a program, it must translate each command stored in the command file from the FoxPro language to a form that is recognised by the computer's microprocessor. The first time the FoxPro program is executed, FoxPro compiles it and translates the entire program which is referred to as the source code to create an object file. This process is called as **compiling** . Hence, compiling is nothing but translation of the source code (FoxPro program) to an object file.

A compiled program cannot run independent of FoxPro. When a program is compiled, FoxPro preserves the original .PRG file and the extension .FXP is assigned to the resultant object file. The extensions for compiled screen, menu and query programs are given in the table below:



A program can also be compiled by choosing the COMPILE option from the PROGRAM menu.

1.4 Memory Variables

Memory variables are temporary quantities that are stored in the memory. They are needed in FoxPro programs to store values which can be later used in calculations or to store results of calculations. For example, if you have to write a simple program for adding two numbers and storing the sum, a total of 3 memory variables will have to be created. Two for the numbers and one for storing the result.

Memory variables can be created using the Command Window as well as in the program. If the memory variables are created from the Command Window, they are retained till the exits FoxPro and variables created in the program are retained till the program is running.

1.4.1 Declaring memory variables

Memory variables can be declared in any one of the two ways :

- Using the STORE command as follows ·

  STORE <value> TO <memory variable name>

  or

- Using an assignment statement as follows :

  <memory variable name> = <value>

Note : A single STORE command can be used to declare more than one memory variable.

Memory variable names should follow the following rules:

1. They should be maximum of 10 characters in length (Spaces are not allowed).
2. The first character must be a letter.
3. The only valid separator is an underscore ('_'). Example NUM_1.

The value stored to a memory variable can be a literal, the contents of a table field, or the contents of another memory variable.

Let us declare the 3 variables needed to add two numbers.

The declaration would be as follows :

```
STORE 15 TO NUM1
STORE 25 TO NUM2
STORE 0 TO SUM
```

OR

```
NUM1 = 15
NUM2 = 25
SUM = 0
```

The variables NUM1 and NUM2 are initialised to 15 and 25 respectively, whereas SUM is initialised to 0.

1.4.2 Types of Memory Variables

FoxPro allocates an area of memory for the storage of memory variables and they can be of four types :

- **Character**

Character variables contain strings of characters, which can be composed of a combination of letter, numbers and punctuation symbols. They are always enclosed in quotation marks.

Example : STORE "PS/2" TO SYS_TYPE, M_NAME = SPACE(10)

- **Numeric**

Numeric values contain numbers which are recognised as numbers and not as characters by FoxPro. Numbers stores in numeric memory variables can be whole numbers or they can be fractional (decimal) numbers.

Example : STORE 1344.45 TO SYSCOST

- **Date**

Date memory variables contain dates that follow the FoxPro date format. Dates are always enclosed in a pair of curly brackets.

Example : STORE {12/21/93} TO PURCHDATE

- **Logical**

This type of memory variable stores either a logical T/F or Y/N. Logical memory variables have to be surrounded by a pair of dots (.).

Example : STORE .T. ANS

The type of memory variable is ascertained by FoxPro from the manner in which the data is supplied. For example, single or double quotation marks identify data as a character string. The letters T,F,Y, and N surrounded by two periods identify the data as a logical variable. Numbers are assumed to be numeric variables unless they are enclosed in quotation marks. Dates are stored in a memory variable by surrounding them with curly braces.

After understanding memory variables and their types, let us have a look at arrays which are nothing but a set of memory variables.