INTRODUCTION TO PROGRAMMING USING DBASE

Sasa M. Dekleva
DePaul University
College of Commerce
25 East Jackson Boulevard
Chicago, IL 60604-2287

ABSTRACT: An introduction to information technology and information systems is normally a component in business school curricula. Students are often required to write a few simple programs, mostly using programming language BASIC. However, dBASE is a good alternative to BASIC in fulfilling this function; indeed, in many respects it is a superior option. The choice of the language is based on many factors. Reasons such as dBASE's excellent debugging facility, students' previous (and possibly frustrating) experience with programming in BASIC, and a perception that dBASE will more likely be used after graduation than BASIC may support the substitution of BASIC with dBASE. The problem with using dBASE at this elementary level is that it offers much too extensive a set of commands and functions, which could easily overwhelm beginning students. However, asmall subset of dBASE programming language, provides sufficient functionality to reach the above goals.This minimal subset should support variables, data input and output, database file creation, loading, listing, updating and sequential processing, program editing, saving, execution and debugging, and structured program constructs sequence, iteration (or repetition) and decision (or selection). This paper presents a functional subset of dBASE commands and the order of coverage which have been successfully used on many occasions. The paper can also be useful to prepare handout materials for students.

KEYWORDS: Information Systems Education, Programming Techniques, Introduction to Programming, dBASE.

WHY NOT BASIC?

Undergraduate curricula in business schools normally include at least one introductory course on information technology (IT) and management information systems (MIS) [1]. This introductory course is typically required for all business students and not just for MIS majors. Computer programming is often one of the concepts presented in such a course, and students normally use BASIC to write, test, and execute a few simple programs. BASIC is, after all, a programming language for beginners.

The usual arguments in support of an exposure to programming include:

- improves student's understanding of how the computer works
- demonstrates what programming is and improves appreciation for information system development and maintenance
- enforces rigorous algorithmic thinking and improves intellectual capability in general
- encourages systematic and structured writing, which should improve any writing and not just writing computer programs

Further discussion of the appropriateness of requiring all business students to practice programming (they very likely did it in high school already), and to practice programming using a third generation language, exceeds the scope of this paper.

The choice of programming language depends on a variety of factors, such as ease of learning and use, popularity, availability, level of the support of structured programming, and the sophistication (or the generation) of a language. In addition to these rational criteria, some less obvious factors may also influence the choice. An argument can be made that programming processes (and struggles) can be better experienced by using a procedural language of a third generation than by using a more advanced fourth generation language. Some pragmatic issues may also be considered. The majority of students bring some (often frustrating) programming experience from high schools [2], mostly in programming in BASIC. They refuse to go through the same affair again. To make the perception about programming in BASIC even worse, the laboratory assistants--most of whom pursue computer science programs--openly ridicule BASIC as inferior to C or Pascal, not to mention Smalltalk. Although BASIC appears to be appropriate for an introductory exposure to programming, some of the above issues may support the selection of an alternative language.

dBASE is a good candidate for a programming language of choice [3, 4, 5, 6]. It has some advantages compared with BASIC. It is readily available, at least the educational version of the product. Its future looks good according to the reviews of the much improved version 1.1 of dBASE IV. It is versatile and provides a typical third generation language with an excellent debugger as well as powerful nonprocedural capabilities. The debugger facility is very informative because it shows the progression through the program step by step. Although not necessarily any better than BASIC in terms of its support of structured and modular programming concepts, it has the advantage of having been designed to manipulate relational databases. Also, very few students have used it before coming to college; dBASE can be a "clean slate." Maybe most importantly, students recognize it as a leading microcomputer database management system and believe that they will, in fact, be using it after graduation to organize and manipulate their data. In other words, students do not show open aversion to dBASE for their programming assignments. These items should demonstrate sufficient grounds for consideration of dBASE as an alternative to BASIC in introductory courses.

students recognize (dBASE) as a leading microcomputer database management system and believe that they will, in fact, be using it after graduation to organize and manipulate their data. In other words, students do not show open aversion to dBASE for their programming assignments.

The following discussion may help those who would like to consider or who plan to use dBASE third generation (called "dot prompt") programming capabilities as a substitute for BASIC in introductory IT or MIS courses.

MINIMAL SET OF DBASE COMMANDS AND SEQUENCE OF THEIR INTRODUCTION

As with any other complete programming language, dBASE has a comprehensive set of commands and functions. To enable students to concentrate on the programming process, program design and logic, it is important to identify a minimal set of commands, knowledge of which enables the writing of simple programs. These commands must also be introduced in an appropriate sequence. The rest of this paper presents a subset of commands and the structure of their coverage, both of which have been successfully used. The whole programming segment of the introductory computer course consumed 12 hours of class time in a laboratory environment with a microcomputer available to each student.

1. Memory variables

The concept of memory variables must be introduced first. Although five types of variables are available (character, numeric, date, logical and memo), the first two types should suffice. The other three types are normally used for database attributes, called "fields." Memory variable names must start with a letter, may contain upper and lower case letters, numbers and the underline character, and may be up to ten characters long. The type of a variable is determined automatically when a value is assigned to it.

The command that creates a memory variable is the STORE command:

STORE {expression} TO {memory variable}

Examples:

STORE "Peter" TO FirstName

STORE 1 TO PageNo

Memory variables can be created even more easily by simply using assignment statements:

{variable} = {expression}

Examples:

LastName = "Smith"

RowNo = 4

Memory variables can be reviewed with the DISPLAY MEMORY command.

2. Presenting data as output

Data can be positioned to any location on a screen with the SAY command and its optional PICTURE part:

@ x,y SAY {expression} [PICTURE {format}]

where x represents row number and y represents column number. Both x and y can be either constants or expressions. The range of x is from 0 to 24; the range of y is from 0 to 79.

Examples:

@ 5,10 SAY "This will be positioned to the 6th row and 11th column."

@ RowNo,10 SAY "RowNo is a numeric memory variable."

@ 0,70 SAY PageNo

Some of the valid format characters are the following:

$	**Displays dollar signs in place of leading zeroes**
*****	**Displays asterisks in place of leading zeroes**
.	**Indicates position of decimal point**
,	**Indicates position of comma**
9	**Is substituted by digits (and signs for numeric data**
A	**Is substituted by letters**
N	**Is substituted by letters or digits**
X	**Is substituted by any characters**

Examples:

Salary = 12458.15

@ 5,10 SAY Salary PICTURE "$99,999.99"

*@ 6,10 SAY Salary PICTURE "*999,999.99"*

The first of these two output commands displays $12,458.15; the second displays **12,458.15.

3. Interactive data entry

Input of data is requested with the GET and the accompanying READ command:

@ x,y GET {variable} [PICTURE {format}]

READ

The function is essentially inverse to data output. The GET command captures characters typed on a keyboard and displays them on the screen; READ then actually loads captured characters to the memory variable(s).

Example:

@ 5,10 GET LastName

@ 8,10 GET SocSecNum PICTURE "999-99-9999"

READ

Input and output operations can be conveniently combined with SAY, GET and a consequent READ:

@ x,y SAY {expression} GET {variable} [PICTURE {format}]

READ

Example:

@ 5,10 SAY "Last Name: " GET LastName

@ 7,10 SAY "Home Telephone: " GET HomePhone PICTURE "(999)999-9999"

READ

4. Program creation, modification, and debugging

Up to now, all commands were typed in at the dot prompt and immediately executed. A sequence of commands can also be entered in a form of a command file (or program) and edited before its execution. This can be achieved with MODIFY COMMAND which has the following format:

MODIFY COMMAND {program name}

The program name must conform to DOS file name constraints and is extended with PRG if no extension is manually provided.

The DO command triggers the program's execution. Its format is:

DO {program name}

One of the many features of dBASE is the so-called TALK. It can be toggled ON or OFF by the SET TALK command. Its format is:

SET TALK [ON/OFF]

The usual default setting is ON, in which case dBASE responds to each command by displaying the evaluated expression. Such a display may be very useful during program testing but should later be turned off because it may interfere with the regular output. This means that after the program is fully tested, the SET TALK OFF command should be inserted at the top of the program and the SET TALK ON command at the end.

The program can be listed and optionally printed with the TYPE command:

TYPE {program name.PRG} [TO PRINT]

When the dBASE IV full screen editor is used, the edited program can be printed by simply selecting the Print option from the menu.

It is a good practice to insert comments into programs to explain the meaning of variables and the function of individual program segments. To place comments in the command file use:

*****	**as the first nonspace character in a line**
&&	**to place a comment to the right of a command**

dBASE IV has a powerful debugger which can be invoked with the DEBUG command. It enables the user to run a program and see the commands as they are executing, edit the program, set breakpoints to halt program execution and display the results of expressions and values of variables while the program is running. This command's format is:

DEBUG (program name}

5. Database files

The CREATE command originates a new database file. Its format is:

CREATE {file name}

After the file structure is defined, the data can be loaded. The file structure can be reviewed by:

DISPLAY STRUCTURE

It can be modified by:

MODIFY STRUCTURE

An old file can be activated by:

USE {file name}

Data can be added to the active file using:

APPEND

or edited by using:

BROWSE

Work can be saved with the key combination:

<Ctrl><End>

or by activating another database file with USE {file name}.

Data from the active database file can be displayed with the LIST command, as follows:

```
LIST [{field 1 name, field 2 name, ...}]
     [TO PRINT]
```

An optional part TO PRINT submits output to the printer also. If field names are omitted, all data are listed.

6. Controlling the program flow

Iteration, repetition, or the loop is implemented with the DO WHILE:

```
DO WHILE {condition}
{action to be repeated}
ENDDO
```

There are two decision constructs, IF and CASE. The format of the IF statement is:

```
IF {condition}
     {action 1}
[ELSE
     {action 2}]
ENDIF
```

The format of the CASE construct is:

```
DO CASE
CASE {condition 1}
{action 1}
CASE {condition 2}
{action 2}
.
.
[OTHERWISE
{action n}]
ENDCASE
```

The CASE construct is preferable when the number of actions is greater than two, to avoid imbedded IF statements.

7. Sequential database file processing

The following implementation of the DO WHILE loop is appropriate when all records from a database file need to be processed sequentially:

```
USE {file name}         && Activate a
     file and retrieve the first record
DO WHILE .NOT. EOF() &&  Repeat
     until the end of file is reached
:    && Current record is processed
     here
     SKIP      && Retrieve the next
     record
ENDDO
```

8. Other output considerations

Two functions are useful for relative positioning of output on the screen; similarly, two other functions are available for printing:

```
ROW()[expression], COL()[expression]
     && Screen functions
PROW()[expression],
     PCOL()[expression]      & &
     Printer functions
```

The function ROW(), for instance, returns the current row position of the cursor. In the following example, the last name is displayed in column 10 of the next row:

@ ROW()+1,10 SAY LastName

Similarly, the function COL() returns current column position on the screen and is used to distribute data horizontally along a certain row.

The default output device is the screen. SET DEVICE TO command is used to redirect output to the printer:

SET DEVICE TO PRINTER

Another SET DEVICE TO command is issued to redirect the output back to the screen:

SET DEVICE TO SCREEN

The SET DEVICE command can be executed from the dot prompt, and can also be imbedded in the program.

Before the output is redirected from the screen to the printer, all occurrences of the relative positioning functions ROW() and COL() must be substituted by PROW() and PCOL(), respectively. Functions ROW() and COL() only work with the screen and their equivalents PROW() and PCOL() only work with the printer.

SAMPLE PROGRAM ASSIGNMENT

The following is an example of a program assignment which can be nicely handled with the set of commands presented above. It demonstrates the use of a repetition construct for sequential processing of a database file and the use of both selection constructs.

CONCLUSION

There is, then, and alternative to BASIC for introductory computer classes in a business school environment. A rather small set of about 28 different dBASE (III+ or IV) commands and functions enables writing of simple programs to reach the goals of an introductory computer course. The careful use of dBASE presents significant advantages to instructor and students alike, and manages to avoid some of the difficulties associated with BASIC. In particular, students gain by becoming familiar with a language they are more likely to use outside of class.

Since there is no book in the form of a short course in dBASE programming presenting only the minimal set of commands and functions, a combination of handout material and a general dBASE text must be used. The author successfully combined a five-page handout with cross references to the appropriate sections of the Ross text [7]. The same set of commands can also be used with dBASE IV, enjoying additional functionality of its debugger. The use of dBASE can be extended with the coverage of its application-generation capabilities, which illustrate different levels of system development sophistication as well as other concepts, such as end user development, prototyping, and database querying.

REFERENCES

[1] Frand, L. J. and Britt, J. A. "Sixth Annual UCLA Survey of Business School Computer Usage," Communications of the ACM, Volume 33, Number 5, May 1990, pp. 544-562.

[2] Myers, J. P., Jr. "The New Generation of Computer Literacy," SIGCSE Bulletin, Volume 21, Number 1, February 1989, pp. 177-181.

[3] Otto, J. C. "dBASE IV: Major Enhancements Make This Version Powerful and Easy," Business Forum, 15, no. 4 (1991): 48-49.

[4] Weixel, S. "dBASE IV: Feature-Rich, Slower than Others," Computerworld 25, no. 13 (1991): 45.

[5] McMullen, J. "End Users Create PC Applications," Datamation 35, no. 23 (1989): 41,43.

[6] Buchanan, R. L. Structured Programming in dBASE IV, Belmont: Wadsworth Publishing Co., 1991.

[7] Ross, S. C. Understanding and Using dBASE III PLUS, St Paul: West Publishing Company, 1987.

SAMPLE PROGRAM:

```
**************************************************************************************************
* Write a program to present some statistics on the students in a class. One record holds the following data for each
* student: student's age, sex code (1=female, 2=male), and student's standing code (1=freshmen, 2=sophomore,
* 3=junior, and 4=senior). The output should contain the average age, the percent of males and females, and the
* percent of freshmen, sophomores, juniors, and seniors. Assume that data are already available in a database
* file called STUDENTS, which contains fields AGE, SEX, and STANDING.
**************************************************************************************************
* Programmer: Terry Byte, SS#: 123-45-6789
**************************************************************************************************
CLEAR
CLEAR ALL
***** Defining memory variables to accumulate total and counts
STORE 0 TO TOTAL_AGE
STORE 0 TO COUNTER
STORE 0 TO NO_MALES
STORE 0 TO NO_FEMALES
STORE 0 TO NO_FRESHMN
STORE 0 TO NO_SOPHMRS
STORE 0 TO NO_JUNIORS
STORE 0 TO NO_SENIORS
***** Setting up the data base file
USE STUDENTS
***** Starting the processing of records in the file
DO WHILE .NOT. EOF()
   TOTAL_AGE = TOTAL_AGE + AGE
   COUNTER = COUNTER + 1
   IF SEX = 1
      NO_FEMALES = NO_FEMALES + 1
   ELSE
      NO_MALES = NO_MALES + 1
   ENDIF
```

SAMPLE PROGRAM, continued:

```
      DO CASE
          CASE STANDING = 1
              NO_FRESHMN = NO_FRESHMN + 1
          CASE STANDING = 2
              NO_SOPHMRS = NO_SOPHMRS + 1
          CASE STANDING = 3
              NO_JUNIORS = NO_JUNIORS + 1
          CASE STANDING = 4
              NO_SENIORS = NO_SENIORS + 1
      ENDCASE***** Advancing to the next record
      SKIP
  ENDDO
  ***** Presenting the results
  @ 1,10 SAY "DESCRIPTIVE STATISTICS OF CLASS POPULATION"
  @ 3,10 SAY "Average age ="
  @ 3,24 SAY TOTAL_AGE/COUNTER PICTURE "99.9"
  @ 5,10 SAY "Percent of males ="
  @ 5,34 SAY NO_MALES*100/COUNTER PICTURE "999.9%"
  @ 6,10 SAY "Percent of females ="
  @ 6,34 SAY NO_FEMALES*100/COUNTER PICTURE "999.9%"
  @ 8,10 SAY "Percent of freshmen ="
  @ 8,34 SAY NO_FRESHMN*100/COUNTER PICTURE "999%"
  @ 9,10 SAY "Percent of sophomores ="
  @ 9,34 SAY NO_SOPHMRS*100/COUNTER PICTURE "999%"
  @ 10,10 SAY "Percent of juniors ="
  @ 10,34 SAY NO_JUNIORS*100/COUNTER PICTURE "999%"
  @ 11,10 SAY "Percent of seniors ="
  @ 11,34 SAY NO_SENIORS*100/COUNTER PICTURE "999%"
  SET TALK ON
```

AUTHOR'S BIOGRAPHY

Sasa M. Dekleva is an Assistant Professor of Information Systems at the DePaul University. He received his bachelor's degree from University of Ljubljana and Ph.D. from University of Belgrade. His areas of expertise include information systems development methodologies, software engineering productivity, data modeling and database design, software maintenance and natural language querying. Dr. Dekleva has been in software engineering more than twenty years which includes nine years of industrial experience in systems engineering, systems analysis and management.

ISSN 1055-3096