Introducing the BASIC Programming Language in a General Course on Computers in Pharmacy: An Approach and Examples

William E. Fassett
Dale B. Christensen

INTRODUCTION

Should programming be taught to pharmacy students as part of an introductory course in computers in pharmacy? Robert Barger suggests that computer literacy includes both computer awareness (an understanding of computer function, structure, terminology, and the impact of computers on one's life and profession) and the ability to do computing, which requires a fundamental understanding of programming concepts as well as minimal skill in at least BASIC programming (1). Speedie published the earliest description of a computer applications course in a pharmacy school in 1980; he did not include programming proficiency in the course objectives (2). The most recent summary of pharmacy computer courses, however, revealed that the majority of courses (16 out of 27, or 59.3%) included programming (3).

We believe that some familiarity with programming concepts is essential to computer literacy and have included a programming experience as part of our survey course, Computer Applications in Pharmacy. Our goal is not to produce programming competency but

William E. Fassett, M.B.A., is Assistant Professor and Dale B. Christensen, Ph.D., is Associate Professor in the Department of Pharmacy Practice SC-69, School of Pharmacy, University of Washington, Seattle, WA 98195.

to show students that programming a personal computer is within their grasp. Students gain experience in analyzing a problem and developing a step-by-step solution. They also learn to describe the solution precisely, in a formal grammar. We believe these objectives can be met not only by exposure to programming concepts using a general purpose language, such as BASIC, but also by introduction to specialized application languages, such as spreadsheets and data base managers. Herein we present three instructional examples for use by others in a similar educational setting.

THE PROGRAMMING COMPONENT

Minimal Lecturing Combined with Self-Study

Our approach involves a minimum set of lectures on general approaches to programming, using BASIC as the example language. In a 30-lecture course, 3 class sessions are devoted to programming, and students are asked to work in groups to produce a small program related in some way to pharmacy practice. To aid the completion of the assignment, three suggested programs are outlined, and students may choose one of them or create one of their own.

Approach to Teaching Programming Concepts

We use the input-processing-output (IPO) model as the suggested structure for introductory level student programs (4). For each of the suggested programs, input and output requirements are given, and a short discussion of a processing approach is included. Finally, an abbreviated list of BASIC commands that are essential to completion of the specific example program is provided to the student. Thus the student can concentrate on learning only the syntax essential to the program he or she has chosen. Admittedly, the students do not learn sufficient grammar to become proficient BASIC programmers, but they have the experience of developing a successful BASIC program.

As an alternative to programming in BASIC, interested students can choose to complete their programming projects using spreadsheets (e.g., Lotus 1-2-3 or Excel), data base managers (e.g., dBASE III or R:Base), or other high-level programming languages (including Turbo Pascal, Turbo C, or Turbo Prolog). Typically, in a class of 40 students, 1 or 2 groups use spreadsheets, 1 or 2 students work individually using another high-level language, and the majority work with BASIC.

Three Sample Programs

The three programs we have used were developed as examples of program fragments that might be found in larger pharmacy computer systems. They also provide a range of contexts suitable to the varied academic preparation of our students (our course is open to students in all three years of our program). Each example is briefly discussed below, and the Appendix contains the complete handout provided to our students.

PRICE.BAS is a program that calculates the retail price of a prescription, given the acquisition cost and the quantity dispensed. Students must implement a prescribed pricing schedule, which includes a minimum retail price, a sliding professional fee, and a fixed percentage markup for acquisition costs greater than $20. More than anything else, this program teaches students how to implement conditional tests. It also provides experience in formatting numeric output, since the output requirements include expressing results in standard dollars-and-cents notation.

The second example program, DEATEST.BAS, attempts to validate a given DEA registration number using the check digit method published by the Drug Enforcement Administration. The student is asked to accept the registrant's last name and DEA number as input, then determine whether the number is internally consistent. This program stresses conversion between text and numeric data types and requires string manipulation. Students also learn to test for ASCII code values and to convert between lowercase and uppercase text. Of the three examples, this one requires the least amount of background knowledge of pharmacy practice but may require the most advanced programming techniques.

The final example program simulates a single infusion dose of a drug in a one-compartment kinetic model. Students are asked to create a program that allows input of the desired dose, the length of infusion, the desired sampling interval, the length of the simulation, and the parameters for clearance and volume of distribution. The output required is a list of plasma concentrations at each of the sampling intervals. This simulation requires the student to determine whether a given sampling time is pre- or post-infusion and to apply the appropriate formula. It provides an introduction to the mathematical commands in BASIC, as well as numeric formatting. It also demonstrates the use of loops and/or arrays.

Implementation in the Course

The programming exercise is integrated into the course during the last half of the term. Two chapters in the textbook are assigned; the first covers programming concepts, and the second reviews programming languages (5). Three lectures are provided. The first lecture discusses the programming cycle and the IPO model of program structure. The second and third lectures cover the rudiments of BASIC programming using as an example the creation of a program to convert pounds to kilograms. Students are supplied with a handout entitled "Entering a BASIC Program" that takes them through a simple example whereby they enter a program, save it to a diskette, then reload the program, modify it further, and run it. Copies of the BASIC manual for the computer system they will be using are also made available.

In our environment, students may use either Macintosh or MS-DOS compatible computers running Microsoft BASIC. Microsoft BASIC (Microsoft Corporation, Redmond, WA) was chosen as the primary programming language for our course because it is almost universally available on MS-DOS-based systems as well as on the Macintosh and because it is upwardly compatible with other microcomputer BASICs, including Quick BASIC and Turbo BASIC.

Faculty members in our department favor the use of group work, particularly in problem-solving situations, to encourage collaborative decision making. Such an approach also allows sharing of resources, such as computers. For both of these reasons, students work in groups of three or four to develop their programs and are given about three weeks to finish the project. During this time, the instructor is available by appointment to answer questions and to provide help. Additional assistance is available in a microcomputer laboratory situated near our regional health sciences library.

The programming project is one of five outside assignments that collectively account for half of the course grade; the project is, therefore, worth about 10% of the final grade. The finished program is submitted to the instructor on diskette and is evaluated on four criteria: the program runs without crashing, the input requested and output produced meets or exceeds the requirements specified, the processing algorithm operates correctly, and attention is given to the user interface (i.e., displays are attractive, adequate prompts are used, prompts and output are correctly spelled, the program recovers from obvious user input errors). Students submit confidential evaluations of the amount of effort each group member contributed to the project, and each student's grade for the project is weighted according to these peer evaluations.

It is common with survey courses to introduce a variety of concepts that will serve the student as a framework for further learning. It is expected that only some students will choose to pursue any specific course inclusion in greater depth. So it is with the inclusion of programming in a computer literacy course. We have noted elsewhere that "just as real fluency in reading and writing our native language is the mark of a literate person, fluency with computers may include, but is not limited to, programming skills" (5). In doing so, we noted with approbation the comments of Alan Kay:

> Computer literacy is not even learning to program. That can always be learned, in ways no more uplifting than learning grammar instead of writing. . . . Computer literacy is a contact with the activity of computing deep enough to make the computational equivalent of reading and writing fluent and enjoyable. (6)

We consider the limited exposure provided by these examples as only one part of a combination of activities that starts students on their way toward mastery over the computer.[1]

[1]Copies of working solutions to these examples are available to faculty members from the authors on request.

REFERENCES

1. Barger RN. Computer literacy: toward a clearer definition. T.H.E. J 1983;11(2):108-12.

2. Speedie S. A computer literacy course for pharmacy students. Am J Pharm Educ 1980;44(1):158-60.

3. Jacobs E, Christensen DB, Gibson T et al. Survey of courses relative to computers in pharmacy practice. Presented to the Section of Teachers of Pharmacy Administration, American Association of Colleges of Pharmacy Annual Meeting, San Francisco, CA, 7 July 1985.

4. Shelly G, Cashman T. Introduction to computers and data processing. Brea, CA: Anaheim Publishing Co., 1980.

5. Fassett WE, Christensen DB. Computer applications in pharmacy. Philadelphia: Lea & Febiger, 1986.

6. Kay A. Computer software. Sci Am 1984;251(Sept):53-59.

APPENDIX

PRICE.BAS: A program to calculate the retail price of a prescription

Inputs: The user should be prompted to enter the wholesale cost of the stock bottle, the size of the bottle in dosage units (e.g., caps, tabs, ml), and the number of dosage units dispensed.

Outputs: The program should implement the following pricing schedule:

Cost of quantity actually dispensed	Dispensing fee
$0.00 - 1.99	$2.00
$2.00 - 4.99	$3.00
$5.00 - 9.99	$4.00
$10.00 - 14.99	$5.00
$15.00 - 19.99	$6.00
$20.00 and over	25% of cost

Further, the minimum retail price, regardless of cost, should be $3.95.

Example:	Cost/Bottle	Pkg. Size	Qty. Disp.	Retail Price	GM%
	$ 2.00	100	30	$ 3.95	84.8
	6.00	100	30	3.95	54.4
	7.00	100	30	5.10	58.8
	15.00	100	50	11.50	34.8
	20.00	100	80	22.00	27.3
	25.00	100	100	31.25	20.0

The program should display on the screen the cost for the quantity dispensed, the amount of the dispensing fee, the gross margin percentage based on selling price, and the total retail price. It should then ask the user whether to calculate another price or quit.

Formulas: *The following formulas may be useful in developing the program:*

$$\text{Cost per quantity dispensed} = \frac{\text{Cost per bottle}}{\text{Package Size}} \times \text{Qty. Dispensed}$$

Retail price = Cost per quantity dispensed + dispensing fee

Dispensing fee = Retail price – Cost per quantity dispensed

$$\text{Gross Margin \%} = \frac{\text{Dispensing fee}}{\text{Retail price}} \times 100$$

Useful BASIC Commands, Functions, and Operators: The program can be developed using only the commands, functions, and operators listed below (although it may be made more fancy using additional commands not listed):

```
*
+
-
/
<
<=
=
CLS
END
GOTO
IF ... GOTO
IF ... THEN
INPUT
KEY OFF
PRINT
PRINT USING
```

DEATEST.BAS: A program to verify a DEA number

Background: The DEA number consists of a 9-character code in the form AAnnnnnnn, where AA = uppercase characters and nnnnnnn = numeric digits.

For physicians and pharmacists, the first letter is usually A or B, the second letter is the same as the first initial of the registrant's last name (or is "9" in some cases).

The first 6 numbers are the registrant's actual serial number, and the 7th digit is a "check digit" that can be used to internally verify that the DEA number is not invalid.

The check digit is calculated as shown by the following example:

DEA Number AB4136126

Add digits 1, 3, and 5 . . .	4 + 3 + 1 =	8
Add digits 2, 4, and 6 and multiply the sum by 2 . . .	(1 + 6 + 2) x 2 =	18

Add the two intermediate results to produce a "checksum" . . .	26
The check digit should equal the right-most digit in the checksum . . .	↑ 6

Inputs: The program should prompt the user to enter the DEA Number and the registrant's last name.

Outputs: The program should advise the user whether or not the DEA Number is internally valid and then determine whether to process another number or end.

Processing: The program should check to see that the second letter of the DEA Number matches the first letter of the last name. Note that if you compare an uppercase letter with a lowercase letter, the computer will determine that they are different.

The program should then calculate the checksum and compare the check digit with the rightmost digit of the checksum.

BASIC needs to know whether entries are character strings (either characters or numbers) or numeric values (numbers only). You will need to accept the DEA Number from the user as a character string so that letters and numbers can be mixed, but to calculate the checksum, you will have to parse ("peel off") the individual digits and convert them into values.

Useful BASIC Commands, Functions, and Operators: This program can be written using the BASIC commands, functions, and operators listed below. It may be made fancier using additional commands that are not listed.

```
*
+
<>
=
ASC( )
CHR$( )
CLS
END
FOR ... STEP ... NEXT
GOTO
IF ... THEN
INPUT
KEY OFF
LEFT$( )
LEN( )
MID$( )
PRINT
RIGHT$( )
STR$( )
VAL( )
```

ONECOMP.BAS: Simulation of a single infusion in a one-compartment model

<u>Background</u>: Among the earliest programs to estimate plasma concentrations of drugs were those used to simulate the levels of a single dose of a drug whose kinetics can be described by a one-compartment model (see text, p. 120-121).

When a BOLUS dose is given, the distribution phase is almost instantaneous, so the blood levels at any point in time following the dose can be described by the following equation that describes the elimination process:

$$Cp = (dose/Vd) \times e^{-((Cl/Vd)xT)}$$

where Cp = plasma concentration
dose = bolus dose
Vd = apparent volume of distribution
e = base of the natural logarithms (2.718282)
Cl = clearance
T = time since bolus dose

The half-life of the drug can be calculated as

$$T-1/2 = 0.693/(Cl/Vd)$$

When an INFUSION is given, the calculations are a bit more complex because the concentration at any time T must account for the infusion rate and whether T is during or post-infusion.

Instead of dose, the concentrations are calculated using the infusion rate (Ko):

Ko = Dose/Ti where Ti = length of infusion

If T is <u>during</u> infusion, the following formula may be used:

$$Cp = Ko/Cl \times \left[1 - e^{-((Cl/Vd)xT)}\right]$$

If T is <u>after</u> infusion, the following formula may be used:

$$Cp = Ko/Cl \times \left[1 - e^{-((Cl/Vd)xTi)}\right] \times \left[e^{-((Cl/Vd)x(T-Ti))}\right]$$

<u>Inputs</u>: The program should prompt the user for the following:

Desired dose (in mg)
Length of infusion (in hours)
Desired sampling interval (in hours) -- how often does the user want concentrations to be displayed?
Desired length of the simulation (in hours) -- for how many hours of simulated time should the program continue?
Desired clearance (in L/hr)
Desired volume of distribution (in L/Kg)

<u>Outputs</u>: The program should print a list (on the screen and/or printer) of the parameters as entered and the calculated half-life. It should then print two columns showing the time since the beginning of infusion and the corresponding plasma concentration in mcg/ml. At the end, the program should determine whether to perform another simulation or quit.

The flowcharts (Fig. 12-1, 12-2) on pp. 276-277 of the text are for an example program to simulate a bolus dose. They may be helpful in designing this program.

Note: There is a typographical error on p. 274 in the REM for line 115. The line should read:

```
115 REM CP = plasma conc, VD = vol of distrib, DOSE = bolus dose
given, CL = clearance, EXP function raises e to the power,
-((CL/VD)*SAMPLE_TIME)
```

Useful BASIC Commands, Functions, and Operators: This program may be created using the commands, functions, and operators listed below. Fancier versions may utilize other commands not listed.

```
*
+
-
/
=
>
<=
CLS
END
EXP( )
GOTO
IF ... GOTO
IF ... THEN
INPUT
INT( )
KEY OFF
LPRINT
PRINT
```