

Inventory Costing: Developing Business Logic In Accounting Systems

Robert W. Ingram, (E-mail: ringram@cba.ua.edu), University of Alabama

Abstract

Business logic is the link in an information system between a user interface and a database. Typically it contains formal rules, translated into computer programs, that control how data will be processed to achieve the objectives of the information system. This paper describes a classroom exercise to help students understand this segment of an information system. In addition, it is useful for helping students understand the relation between traditional financial accounting and accounting systems. They are required to describe formally and precisely how FIFO and LIFO inventory values can be derived in an automated system.

Introduction

This paper describes a classroom exercise to help students understand the role and importance of business logic in a computerized accounting system. It examines the tasks of processing inventory data to determine cost of goods sold using FIFO and LIFO methods and of updating inventory records. In addition to being designed to help students understand the function of the business logic component of an information system, it is designed to help students understand differences between how data are processed in traditional financial accounting courses and how they are processed in computerized accounting systems.

The next section of the paper explains the purpose of the exercise. The second section contains the exercise. The third section provides a solution and notes for the exercise. The fourth section provides a brief summary, and the final section identifies suggestions for future research.

Readers with comments or questions are encouraged to contact the author via e-mail.

Purpose

A system can be conceived as containing three segments: a user interface, a database that stores and retrieves data, and a business logic segment that processes data and links the user interface to the database. The business logic segment often is left out of the conceptual model in accounting courses. Financial accounting sidesteps the issue by assuming a human processor who calculates the required numbers, records transactions, and computes balances. Business logic is a mental process in which the user decides on how rules should be applied. Often systems courses do not focus on this segment either, emphasizing instead the creation of user interfaces and, moreover, the structure and design of relational databases.

The use of commercial software in systems courses can further exacerbate the problem. These programs reveal the user interface but conceal the business logic and, in many cases, the database functions. Some of these programs

are simply automated general ledger systems that take away the necessity for human data processing required of manual systems. Students play the role of data entry clerks as they learn to process transactions and create reports. More advanced products may provide exposure to a fuller range of activities (sales, purchasing, etc.), but they are not particularly useful for helping students understand how computerized systems process information. The same is true of database programs such as Microsoft Access that are commonly used in systems courses. Typical exercises in textbooks that provide instruction in using database programs emphasize the creation of tables and user interfaces. Students can learn something about processing by constructing queries, but query by example techniques do not provide a basis for creating realistic business rules, unless one is prepared to add programmed components (modules in Access). And, there is no particular advantage in using this software for program development unless one is interested in developing a stand-alone Access system for a small business environment.¹

The exercise described concentrates on the business logic segment. Though it could be connected to a user interface and formal database, it is currently designed to function independently of these segments. The issue of concern is not how data are entered, stored, or retrieved, but how the system takes the input and manipulates it to produce needed information and to update records. It forces students to think formally and precisely about how information is processed in a computerized system.

Another purpose of the exercise is to help students transition their thinking from financial accounting to systems courses. The conceptual model underlying financial accounting is the general ledger model, largely still thought of in terms of a manual system. Students learn financial accounting by recording journal transactions, posting ledger accounts, and transferring these data to financial statements. A database perspective provides the conceptual model for much of

accounting information systems.² Most transactions originate within subsystems that capture business activities as elements of relational databases. Financial statements are simply one view or type of report produced by these systems. The general ledger is a part of the database that captures data in a specific form for financial reporting purposes.

Database systems store and retrieve data using processes that are quite different from those assumed in financial accounting courses. What is similar between financial accounting and systems is the business logic that links data collection and data storage. Though financial accounting is only a part of a management information system, the business logic that applies to that part of the system is similar conceptually between traditional and computerized systems. The difference in this logic segment is in how processing occurs. The rules necessary to process data in a computer system are much more formal and precise than those in a traditional system because of the constraints and limitations inherent in a computer environment. This exercise can be used to help students understand similarities between manual and computer systems and to understand why it is important to be able to describe processes with which they are familiar in financial accounting in precise and detailed terms.

The exercise provides students a simple scenario and requires them to describe the processing necessary to accomplish a fundamental accounting task, computing FIFO and LIFO inventory values. The description occurs at different levels, all of which may not be appropriate in a particular course setting.

At a basic level, students are asked to explain the steps necessary for a program to access inventory records containing different cost layers and products, to calculate inventory values using these records, and to update the records for those items that have been sold. The purpose of this requirement is to force students to examine the mental process they go through in calculating in-

ventory values and to translate this process into individual steps that a program must follow to perform the same tasks.

At a second level, they are asked to prepare a program flowchart describing the logic associated with these tasks. The purpose of this requirement is to help students see the need for a formal description of a logical process as a means of describing the steps necessary to accomplish a task. In particular, the flowchart should reveal the recursive loops in the process of calculating inventory values and get students to think about how the looping must change between FIFO and LIFO methods.

At the third level, they are asked to prepare a step by step description of a computer program that completes the inventory tasks. This step is a useful preface for designing a computer program. It requires students to formalize the recursive processes they described in the second requirement. For courses that have no computer programming prerequisite or component, this requirement may be the last that is feasible for the exercise.

At the last level, students are required to write a computer program that completes the tasks and to provide output to demonstrate the tasks have been completed. The program is not especially complicated and provides an opportunity for students to apply their programming skills in an accounting context, for those programs and courses that require these skills. The choice of programming language is left to the instructor or student. The instructor does not even have to be particularly competent in the programming language because the student must demonstrate that the program produces accurate output.

An alternative to having students produce a program is to have them evaluate and explain the example program in the suggested solution to the exercise provided later in the paper. This can be an exercise in exposing students to computer

programming and learning to read and examine the logic of a program. Students can also experiment with the computer program provided in the solutions by running the program and changing input values to adjust the product, number of units sold, and inventory method used. All software used in the exercise are available on the web without cost.

The Exercise

As an employee of a large merchandise company, you have been given the task of developing a module for the company's information system that determines inventory costs for sales transactions and updates inventory records. The company uses FIFO for some products and LIFO for others. Therefore, the system should be able to determine either value for a particular transaction.

Transaction data that will serve as input to the module are the product number, the quantity of the product sold, and whether FIFO or LIFO value is desired. Inventory records that will be processed and updated are stored in a file containing three fields of concern for this module: the product number, the unit cost of the product, and the quantity of the product on hand. The company sells several products and maintains inventory records for each product and for different cost layers for each product. Records appear sequentially in the file by product and from the earliest to latest purchase date for each product. The following records should be used for testing your module:

<u>Product Number</u>	<u>Unit Cost</u>	<u>Quantity on Hand</u>
1	10	10
1	11	10
1	12	10
2	3	10
2	4	10
2	5	10

Your supervisor has asked that you provide a report that contains:

1. A narrative description of the steps necessary to process the inventory data and produce the required information,
2. A program flowchart that describes the logical steps necessary for the module to accomplish its purpose,
3. A listing of steps a computer program would need to complete to process the data, and
4. Program code that accomplishes the tasks described for the system.

Updated records will report the quantity on hand after the sales transaction for those layers that have not been fully depleted. You may use any standard programming language for the fourth requirement, unless otherwise specified by your instructor. You can assume that sales have already been filtered to eliminate transactions requiring more units of a particular product than are available in inventory.

The program should be demonstrated by providing cost and updated inventory records for each of the following transactions:

<u>Inventory Method</u>	<u>Product Number</u>	<u>Units Sold</u>
FIFO	1	2
FIFO	1	12
FIFO	2	4
FIFO	2	22
LIFO	1	2
LIFO	1	12
LIFO	2	4
LIFO	2	22

Solution and Notes

This section provides examples of responses to each case requirement. The examples are suggestive of appropriate solutions, but alternative approaches are possible. Rather than providing students with a specific solution, instructors should use the case as a basis for discussion of

how information systems accomplish the tasks assumed in more traditional courses. The examples provide guidance for that discussion.

Narrative Description

A key issue in writing the narrative is for students to be precise about each task a computer program must perform and the sequence in which the tasks must be completed. Unlike human processing, machine processing does not tolerate ambiguity. If the program developer is not able to define specifically what a program is to accomplish and describe precisely how the program will accomplish its purpose, it is likely that the program will not accomplish this purpose. The remainder of this section provides an example narrative.

The system must complete a series of tasks. The inventory method, product number, and quantity sold must be obtained by the business logic component of the system from user input. The file that contains inventory records must be opened, and records must be read from the file and stored in computer memory. The program must compare the product number from the transaction with the product numbers of the inventory records. When a match is found, the program must identify the appropriate inventory layers. The first layer used for FIFO is the oldest layer (first in sequence) for a particular product. The first layer used for LIFO is the newest layer (last in sequence). If a record does not match the product number or is not the appropriate layer, the program must process the next record.

Once the appropriate product and layer are identified, the program must determine whether the number of units in the layer is sufficient to cover the requirements of the transaction. If the number is sufficient, the cost of goods sold for the transaction will be the cost of units removed from other layers (if any) plus the number of units removed from the current layer times the unit cost of the current layer obtained from the inventory record. In addition, the number of

units in the layer must be adjusted by subtracting the number of units removed from the number available prior to the sale.

If the number of units in the current layer is not sufficient, the program must get the cost of the current layer and store that amount to be added to the cost of other layers. The cost of the current layer is the quantity of units in the layer times the unit cost of the layer. In addition, the number of units in the layer becomes zero, and the record for the layer should be removed from the inventory file. The next layer then must be processed and the cost of units in that layer must be added to the cost of the prior layer until sufficient units are identified to cover the requirements of the sale. As each layer is processed, the units in the layer must be reduced by the number of units drawn from the layer.

Finally, the program must rewrite the inventory records in the original sequence, omitting those for which the quantity of units has become zero. The program must also report the cost of goods sold to the user.

Program Flowchart

Figure 1 provides an example of a flowchart. The key tasks described in the flowchart are the same as those in the narrative. A primary advantage of the flowchart is its ability to demonstrate the recursive processing that must occur in the system, matching on product number and selecting the appropriate inventory record (each record represents a combination of product number and cost layer).

Though the logic underlying the tasks is relatively straightforward, describing this logic will not necessarily be an easy task for students who are not used to thinking explicitly through the steps necessary to perform an accounting task. As was true for the narrative, alternative solutions are possible. However, the number of acceptable variations of the flowchart is much more limited than for the narrative. Through

class discussion, the instructor should help students come to a strong consensus about what the flowchart should contain. Again, a purpose of the exercise is to help students learn to think precisely and to learn why such thinking is important.

Another avenue of discussion about the flowchart is how the process differs between FIFO and LIFO methods. The logical steps in the program are identical between the two methods. The difference is in where the program begins and ends its processing. Students are likely to understand that FIFO begins with the oldest inventory items and moves towards the newest and that LIFO begins with the newest and moves towards the oldest. They probably have not thought, however, about how those tasks are accomplished when processing inventory records. If the records are sequenced by product number and cost layer, the task is simply a matter of whether looping begins with the first record in the file or the last.³ The program works exactly the same way for FIFO or LIFO. It simply starts at a different location in the file and moves in a different direction.

Program Description

Writing a computer program without a specific idea of the steps the program will follow is an exercise in futility. Breaking a task into steps and sequencing those steps in a logical order is important for learning to think precisely. Thus, even if code is not to be written, it is useful for students to identify the steps that a program would follow to process inventory data.

Table 1 contains a listing of those steps. The format used is arbitrary and variations are appropriate, but the sequence of steps is relatively fixed. For example, the inventory file must be opened and data in the file must be read by the program before records can be matched or updated. Indenting is used in the table to identify related processes. Other ways of grouping the processes could be used.

Figure 1
Program flowchart for calculating cost of goods sold and updating inventory

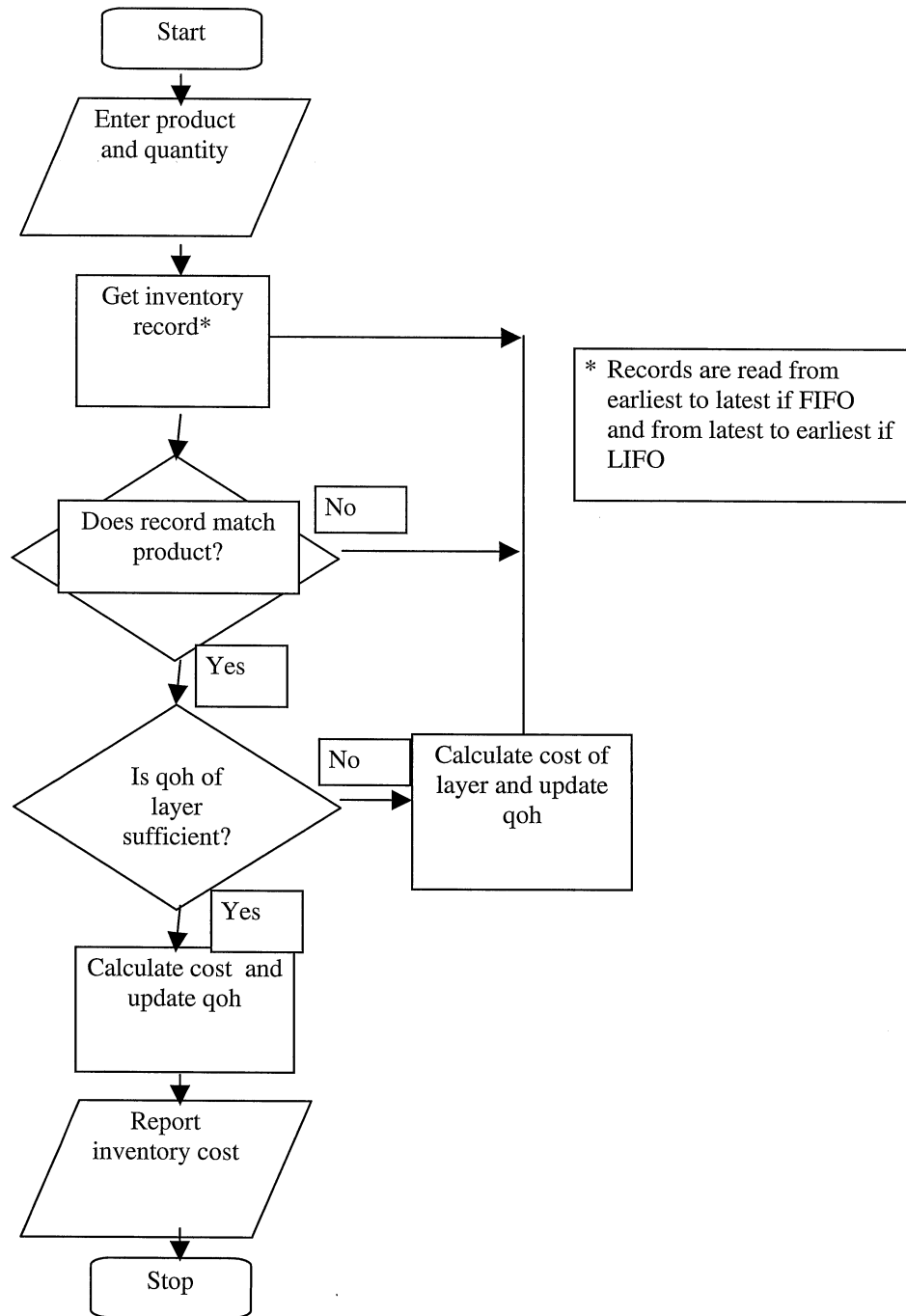


Table 1
Program Steps for Computing FIFO or LIFO Inventory Costs

1	read transaction data (product number, quantity sold) and inventory method
2	open inventory file
3	create variables to store product number, unit cost, and quantity on hand data
4	if inventory method is fifo
5	read next inventory record from beginning of file in increasing sequence
6	if product number for record matches product number for the transaction
7	if quantity on hand (qoh) is \geq quantity needed to complete sale
8	compute cost of sale [layer cost (line 13) + (quantity * unit cost)]
9	reduce quantity on hand by quantity removed (qoh = qoh - quantity)
10	report cost of sale
11	exit program
12	if quantity on hand is $<$ quantity needed to complete sale
13	compute cost for units in current layer (layer cost = qoh * unit cost)
14	reduce quantity needed to complete sale (quantity = quantity - qoh)
15	reduce quantity on hand to 0 (qoh = 0)
16	return to line 5
17	if product number for the record does not match product number for the transaction
18	return to line 5
19	if inventory method is lifo
20	read next inventory record from end of file in decreasing sequence
21	if product number for record matches product number for the transaction
22	if quantity on hand (qoh) is \geq quantity needed to complete sale
23	compute cost of sale [layer cost (line 28) + (quantity * unit cost)]
24	reduce quantity on hand by quantity removed (qoh = qoh - quantity)
25	report cost of sale
26	exit program
27	if quantity on hand is $<$ quantity needed to complete sale
28	compute cost for units in current layer (layer cost = qoh * unit cost)
29	reduce quantity needed to complete sale (quantity = quantity - qoh)
30	reduce quantity on hand to 0 (qoh = 0)
31	return to line 20
32	if product number for the record does not match product number for the transaction
33	return to line 20

Class discussion of the program description should focus on identifying all the necessary steps and putting them in the required order:

1. Line 1 requires that the appropriate variables be read from input. Three variables are necessary—the product number, the number of units of the product sold, and whether the inventory method used is FIFO or LIFO.
2. Line 2 indicates that the inventory data file that lists product and cost data must be opened. Depending on the program language used, opening the file may involve a specific statement or a reference to the data file.
3. Line 3 creates variables (address locations) to store data from the inventory file. This is an important step in computer processing that does not exist in manual processing.
4. Line 4 requires the program to determine which inventory method is being used and invokes lines 5 through 18 if the method is FIFO. Lines 19 through 33 repeat this process when the method is LIFO.

5. Line 5 indicates that processing data involves reading a record from the inventory file and manipulating that data if the product number is the same as that of the transaction that was input in line 1. The key difference between the FIFO and LIFO methods in this program is whether the processing begins with the first record in a sequential file or with the last record. Compare lines 5 and 20.
6. Line 17 provides for the option that the record read from the inventory file does not match the transaction. When a match is not found, the program loops back to read the next record from the inventory file. The program assumes that all transactions are valid references to existing products. Class discussion could consider what happens if an invalid product number is entered. Another option could be added that results in an error message, for example.
7. If the product number for the transaction matches a record in the inventory file, two possibilities arise. One of these is that the number of units in the record (number of units available in the cost layer) is sufficient to complete the transaction. Lines 7 through 11 describe the processing under this condition. In line 8, the cost of goods sold is computed as the cost brought over from a previously processed record (layer), if any, plus the number of units from the current record (layer) needed to complete the sale times the unit cost of the current layer. The number of units removed from the current record is subtracted from the quantity on hand for the record in line 9. The remaining tasks are to report the cost of goods sold to the user (line 10) and end the program (line 11).
8. A second possibility when a match occurs is that the number of units in the inventory record is less than the number of units needed to complete the sale. Lines 12 through 16 provide for this option. The cost for those units in the current record is the number of units in the record times the unit cost (line 13). The number of units still needed to complete the sale is the number of units needed to complete the sale minus the number of units that were in the current record (line 14). The number of units in the record is zero after the units sold are removed (line 15). Processing continues by reading the next record (line 16).
9. An identical process is followed for LIFO, lines 19 through 33, except that processing begins with the newest record for each product.

Additional discussion could focus on how the program could fail to produce valid information and what controls could be included to prevent or detect problems. The primary purpose of this exercise, however, is to illustrate the business logic component of information systems. Many of the primary controls (access rights to data and validation of input for correct format and data types, for example) are the responsibility of the user interface or the database. The activities that affect information validity in this program are accessing the correct records and processing them in the correct order and, most importantly, the program logic that results in the correct manipulation of the data.

Program Code

The final requirement of the exercise is to prepare a computer program that calculates cost of goods sold and updates inventory. This program will vary depending on the programming language and approach used by students. Table 2 provides an example program written in perl (practical extraction and reporting language). It is presented primarily for those who want to have students examine and perhaps experiment with a program, rather than write their own. For this purpose perl has its advantages.

Table 2
Program to Calculate Cost of Goods Sold and Update Inventory

```

1  #!/usr/local/bin/perl
2  # inv_method.pl: reads data from an inventory file, computes fifo
3  # or lifo inventory cost, and updates inventory file
4
5  # open files
6  open(INV, "inventory.data"); #open inventory data file
7  open(TEMP, "> inventory.temp"); #open temporary file in write mode
8
9  # set initial conditions
10 $i=0; #count variable
11 $prod_quant = 22; #product quantity
12 $prod_num = 2; #product number
13 $method = "lifo"; #inventory method
14 $stop = 0; #indicator to stop processing
15
16 # read data from inventory file
17 while($line = <INV>) #read file as long as records exist
18 {
19     $i++; #increase count variable
20     @inventory = split(":", $line); #read records; fields delimited by :
21     $pnum[$i] = $inventory[0]; #product number from file
22     $scost[$i] = $inventory[1]; #unit cost from file
23     $qoh[$i] = $inventory[2]; #quantity on hand from file
24 }
25 close INV; #close inventory file
26
27 $nn = $i-1; #get total number of records in file
28 $quant = $prod_quant; #get copy of product quantity
29
30 # determine subroutine based on inventory method
31 if($method eq "fifo"){&fifo;}
32 if($method eq "lifo"){&lifo;}
33
34 # Print revised inventory records
35 for($i=1;$i <= $nn;$i++) #loop through records
36 {
37     if($qoh[$i] > 0) #omit layers with no inventory
38     {
39         print "$pnum[$i]:$scost[$i]:$qoh[$i]:\n"; #print records to screen
40         print TEMP "$pnum[$i]:$scost[$i]:$qoh[$i]:\n"; #print records to file

```

```
41     }
42   }
43   close TEMP; #close temporary file
44   rename "inventory.temp", "inventory.data"; #update inventory file
45
46   # fifo subroutine
47   sub fifo
48   {
49     for($i=1;$i<=$nn;$i++) #set loop parameters to start at beginning of file
50     {
51       while($pnum[$i]==$prod_num) #product number matches current record
52       {
53         #inventory layer has sufficient quantity and quantity has not
54         #already been met from another layer
55         if(($quant <= $qoh[$i]) and ($stop==0))
56         {
57           #get total cost from current layer
58           $tcost=$tcost+($quant*$cost[$i]);
59           $qoh[$i]=$qoh[$i]-$quant; #recompute quantity on hand
60           $stop = 1; #set to stop processing
61
62           #print total cost of goods sold
63           print "The FIFO cost for $prod_quant units is \$$tcost.\n";
64
65           last; #end loop
66         }
67
68         #inventory layer does not have sufficient quantity and quantity
69         #has not been met from another layer
70         elsif(($quant > $qoh[$i]) and ($stop==0))
71         {
72           #get as much of total cost as possible from current layer
73           $tcost = $tcost+($qoh[$i]*$cost[$i]);
74           $quant = $quant - $qoh[$i]; #reset quant to amount still needed
75           $qoh[$i] = 0; #reset layer quantity to 0
76           $i++; #increase count for next layer
77         }
78
79         #continue if inventory layer is not needed
80         else {$i++;} #increase count for next layer
81       }
82     }
```

```
83     }
84
85     # lifo subroutine
86     sub lifo
87     {
88         for($i=$nn;$i >= 0;$i--) #set loop parameters to start at end of file
89         {
90             while($pnum[$i] == $prod_num) #product number matches current record
91             {
92                 #inventory layer has sufficient quantity and quantity has not
93                 #already been met from another layer
94                 if(($quant <= $qoh[$i]) and ($stop == 0))
95                 {
96                     #get total cost from current layer
97                     $tcost = $tcost + ($quant*$cost[$i]);
98                     $qoh[$i] = $qoh[$i] - $quant; #recompute quantity on hand
99                     $stop = 1; #set to stop processing
100
101                     #print total cost of goods sold
102                     print "The LIFO cost for $prod_quant units is \\\$tcost.\n";
103
104                     last; #end loop
105                 }
106
107                 #inventory layer does not have sufficient quantity and quantity
108                 #has not been met from another layer
109                 elsif(($quant > $qoh[$i]) and ($stop == 0))
110                 {
111                     #get as much of total cost as possible from current layer
112                     $tcost = $tcost + ($qoh[$i]*$cost[$i]);
113                     $quant = $quant - $qoh[$i]; #reset quant to amount still needed
114                     $qoh[$i] = 0; #reset quantity for layer to 0
115                     $i--; #decrease count for next layer
116                 }
117
118                 #continue if inventory layer is not needed
119                 else {$i--;} #decrease count for next layer
120             }
121         }
122     }
```

Perl is derived from C and uses a syntax similar to other C-based languages such as C++ and Java. It is not strongly typed, however, and is easier to use than many other languages. Also, it has strong text handling capabilities. The product number used in the exercise is a simple number (1 or 2). The program will handle longer text identifiers as well. Thus, the product number could be something like "A12b3-45c", and the program would work equally well. Perl is an interpreted, rather than compiled, language. Therefore, the program is run directly from source code without the need to compile object code. Of primary importance is that perl is free and easily installed on most computers. Language files can be downloaded from www.perl.org. Versions are available for Windows and Unix operating systems. On a Windows machine, the code is in the form of an executable file. Simply execute the install program and specify where the files should be saved. Typically, this is to `c:\perl`. Programs are run and files are accessed from `c:\perl\bin`. To run the program in Table 2 from the command prompt, go to the `c:\perl\bin` directory where the inventory program and data file have been stored and enter `perl inventory.pl`. The `inventory.pl` program in Table 2 and the data file accessed by the program can be downloaded from (<http://bama.ua.edu/~ringram/inventory/>).⁴

The code in Table 2 follows fairly closely the steps in Table 1. The perl program begins with header information in lines 1 through 3. Perl uses the hash symbol (`#`) to identify comments. Comments are provided throughout the program to describe the purpose of each line. The first line is standard for perl programs and identifies the location of the perl interpreter in most Unix systems. The line is ignored and does not need to be modified on Windows systems.

The actual program begins in line 6. Two files are opened. One is the inventory file and the other is a temporary file to which changes in the inventory file are written. The inventory file is opened in read mode and the temporary file is

opened in write mode, indicated by the `>` symbol before the file name.

Lines 10-14 create the input data and initial conditions for the program. Input normally would be read from a user interface program that passes data to the perl file for processing. Because the user interface is not part of this exercise, the data are input directly as variables in the program. The values of `$prod_quant`, `$prod_num`, and `$method` can be changed in a text editor.⁵

Line 17 begins reading records from the inventory file. Records will continue to be read until end of file is reached. Each record is separated into the product number, unit cost, and quantity fields in lines 20-23. Fields are separated in each inventory record by a colon. Line 20 uses this identifier to create the fields that will be stored in memory. These data are stored in arrays numbered from 1 for the first record through the last record number. Storing the fields in memory, permits the processing to begin with the first record (1) and work forwards for FIFO or to begin with the last record (`$nn` determined in line 27) and work backwards for LIFO.

After the data are read from the inventory file, the file is closed in line 25.

Line 28 creates a copy of the number of units sold. One copy will be modified later in the program to keep track of how many units are still needed to complete the sale. The other copy is retained to confirm the number of units sold with the user (line 63).

Before data manipulation begins, the program needs to know whether to perform the manipulation with FIFO or LIFO. Lines 31 and 32 are used for this purpose. These lines call the appropriate subroutine. The subroutines begin with line 47 for FIFO and 86 for LIFO. A third subroutine could be added to return an error

message if the method entered by the user were something other than FIFO or LIFO.

After the FIFO or LIFO routine has been run, the program returns to the operations beginning in line 35. Lines 35 through 42 print the revised inventory records. Two versions of the records are printed. One is printed for the user to see, and the other is printed to the temporary file. Records are printed in the same order as the original inventory file so the sequence of layers is unchanged. This is an important step so the inventory program will work each time it is run. Layers that were depleted by the current transaction are removed from the file.

After the records have been transferred to the temporary file, the file is closed (line 43) and the original inventory file is replaced by the temporary file. This replacement completes the updating process. If this program is being used to illustrate the effect of various transactions on cost of goods sold and the inventory records using the original data file, line 44 should be commented out by placing a # at the beginning of the line. Otherwise, the updates will be written to the inventory file and the original file will be deleted.

The FIFO and LIFO subroutines are substantially identical. An important difference is in lines 49 and 88. The FIFO routine begins with record number 1 ($i=1$) and continues forward ($i++$) until all records have been read ($i \leq n$). The LIFO routine begins with the last record ($i=n$) and continues backwards ($i--$) until the first record has been read ($i \geq 0$). In the FIFO routine, records are incremented after processing ($i++$ in lines 76 and 80). In the LIFO routine, records are decremented ($i--$ in lines 115 and 119).

Line 51 provides a branch for processing records when the product number in the inventory file matches the product number for the current transaction. If a match occurs, three possibilities are handled. One (lines 55-66) is that the quan-

tity in the current layer is sufficient to complete the sale. Two (lines 70-77) is that the quantity in the current layer is not sufficient to complete the sale. Three (line 80) is the current layer is not needed because required units have already been removed from other layers.

When the first possibility arises, the cost of the sale ($cost$) increases by the number of units removed from the current layer (line 58). The quantity of units in the current layer is reduced by the number removed (line 59). Also, the indicator $stop$ is set to 1 to prevent processing of additional layers. Because the first condition signals that all costs have been added to the total cost number, this amount can be returned to the user (line 63) and the loop can be ended (line 65).

When the second possibility arises, the cost of the sale ($cost$) is increased by the cost of units in the current layer (line 73). The number of units needed to complete the sale ($quant$) is reduced by the number of units in the current layer (line 74), and the number of units in the current layer is reduced to 0 (line 75). Processing then returns to the printing commands in line 35, and the program ends.

Table 3 provides the cost of goods sold returned by the program for each transaction included in the exercise.

Table 3
Cost of Goods Sold Computed by Inventory Program

Product Number	Units Sold	FIFO Cost	LIFO Cost
1	2	\$20	\$ 24
1	12	122	142
2	4	12	20
22	22	80	96

Summary


This paper describes a classroom exercise for examining the business logic segment of an information system. The exercise uses a simple

inventory costing situation to illustrate the importance of business rules and data processing in an information system. Students work through a series of steps to describe the process of calculating FIFO and LIFO costs and updating inventory amounts, to produce a program flowchart to clarify the logic in the process, to identify the program steps necessary to complete the task in an automated environment, and to create a computer program that will accomplish the necessary tasks. The last step could involve reviewing and using a program rather than creating one.

Important learning objectives of the exercise are for students to understand the role of the business logic segment, to compare and contrast the implementation of business rules in automated and manual environments, and to illustrate the importance of precision and organization in developing a computerized information system. These objectives can help students grasp the relation between financial accounting and accounting systems.

Suggestions for Future Research

The exercise described in this paper is a prototype for any number of similar exercises involving various accounting processes. This exercise could be expanded to include other inventory valuation methods (weighted average, for example) and to incorporate other accounting rules (such as lower of cost or market). From a systems perspective, the exercise could be expanded to include a more comprehensive system development project that included a user interface and a database management system. This system could be implemented using Access and Visual Basic. Alternatively, the perl approach used in the example solution could be implemented using a web browser interface with a cgi

script and could use the perl DBI (database interface) module to connect to a standard database management system. 

Endnotes

1. A few commercial accounting systems in fact are offered as stand-alone Access programs. For example, Advanced Software provides a version of its UA Corporate Accounting product as an Access program. The product relies on the Access interface and database engine, but all of the processing is in the form of Visual Basic modules.
2. Some textbook authors attempt to guide students through the transition by beginning with a manual, general ledger approach and moving to a database approach (see Romney and Steinbart, 2000, for example).
3. The task of sequencing records could be handled in creating the inventory file used in this exercise by the SQL commands GROUP BY for the product number and ORDER BY for the purchase date if raw data are not sequenced.
4. Save the zip file to disk. Unzip the files, and copy them to c:\perl\bin.
5. If the program is run on a Windows machine, it can be opened in a standard word processor such as Microsoft Word or WordPerfect. These programs add end of line markers to program lines that prevent the program from running on most Unix machines, however.

References

1. Romney, M. and P. Steinbart, *Accounting Information Systems*, 8th ed., Prentice Hall, Upper Saddle River: NJ, 2000.