Computers Educ. Vol. 20, No. 4, pp. 311–314, 1993
Printed in Great Britain. All rights reserved

0360-1315/93 $6.00 + 0.00

STRUCTURED PROGRAMMING TECHNIQUES: A STUDY OF RELATIVE COMPLEXITY—SOME PROGRAMMING CONSIDERATIONS

M. Er
ICS Department, KFUPM, Box 1779, Dhahran 31261, Saudi Arabia

(Received 14 October 1992; accepted 12 April 1993)

Abstract—This paper examines a previous empirical study on the complexity of structured programming techniques and finds that all five structured solutions contain programming errors, although they are all based on Nassi–Schneiderman diagrams. The programming errors, of course, invalidate the previous empirical results on the complexity of structured programming techniques. An elegant programming solution to the multi-level control break reporting logic problem is then presented.

INTRODUCTION

In a recently published paper[1], Headrick and Fowler report an empirical study investigating the complexity of some structured programming techniques. The programming problem chosen is known as the *multi-level control break reporting logic* problem, which has something to do with multi-conditional reporting in a data processing setting. It is somehow believed that different structured programming techniques when applied to solving the same programming problem using different nesting of if-statements and loops may yield different levels of complexity that affect program comprehension and software maintainability. If their theory is correct, it implies that it is better to teach students good structured programming techniques that yield least level of complexity so that they can inherit good programming styles.

Headrick and Fowler[1] investigated 35 COBOL programming textbooks on their solutions to the multi-level control break reporting logic problem and classified them as follows:

Technique A: un-nested if-statement logic;
Technique B: nested if-statement logic;
Technique C: nested perform-loop logic.

They then added two more structured programming techniques of their own:

Technique D: perform-loop with a nested if-statement logic;
Technique E: if-statement with a nested perform-loop logic.

Technique E was found to be significantly more complex than other techniques at pre-test, and was subsequently deleted. In the actual test, a group of novice programmers were chosen as subjects, and their perceived levels of complexity of the four remaining techniques are listed below in increasing order of complexity:

(1) A
(2) B, D
(3) C.

However, what comes as a surprise is that Headrick and Fowler[1] simply refused to accept the empirical test results which clearly suggest that they should switch over to Technique A rather than continue using Technique C in teaching. In fact, they stated quite unambiguously:

"Not having totally overcome our pre-investigation biases, we are not quite prepared to merely accept the results presented in this paper and begin teaching un-nested if-test (i.e. Technique A) control break logic." [Adapted from [1], comments in parentheses added.]

More disturbingly, we find that *all* of the five Nassi–Schneiderman diagrams which employ structured programming techniques are wrong. We are obliged to point out the programming errors so that they are avoided in subsequent surveys of the same kind. This is the main purpose of this paper. The unfortunate errors, of course, nullify the empirical test results.

The second purpose of this paper is to present an elegant programming solution which is believed to be superior than any known solution to the multi-level control break reporting logic problem.

THE MULTI-LEVEL CONTROL BREAK REPORTING LOGIC PROBLEM

For the benefit of the readers who do not know what the multi-level control break reporting logic problem is, we state the problem succinctly below.

An input file consists of a set of sales records. Each sales record consists of salesman name (and/or salesman number), customer name (and/or customer number), and sales information, such as item name, unit price, volume, and total price. The object is to print a report listing each sales record in detail, a summary total of each customer, and a summary total of each salesman.

For the purpose of this exercise, we are interested in the control logic, and therefore the input file may be simplified as follows:

```
salesman 1, customer 1, sales information
salesman 1, customer 1, sales information
salesman 1, customer 2, sales information
salesman 1, customer 2, sales information
salesman 2, customer 3, sales information
salesman 2, customer 3, sales information
...
```

It is further assumed that the input file is sorted using salesman as the primary key, and customer as the secondary key. An implicit assumption is that each customer is handled by one and only one salesman.

BASIC ASSUMPTIONS AND PROGRAMMING ERRORS

For each of the solutions reported in [1], the following basic assumptions are made, although they were not stated previously.

(i) An initial read is performed in program initialization.
(ii) The previous salesman is initialized to the current salesman.
(iii) The previous customer is initialized to the current customer.
(iv) When a data record is read, the current record becomes the previous record, and the new data record becomes the current record. This means previous salesman, current salesman, previous customer, current customer are updated accordingly.
(v) When an end of file is reached, arbitrary values are assigned to current salesman and current customer such that current salesman $\neq$ previous salesman, and current customer $\neq$ previous customer.
(vi) The calculations of total sales for each customer and each salesman are irrelevant to the control logic and are ignored.

Although assumptions (ii) and (iii) are un-natural, there is nothing wrong with them.

Having said that, we now discuss each of the five programming techniques reported in terms of the Nassi–Schneiderman diagram, and point out its errors, if any.

Technique A: un-nested if-statement logic (Fig. 1 of [1])

At the end of a subsequence of a salesman's sales records, the customer total line is printed twice, which is wrong. This is because current salesman $\neq$ previous salesman $\Rightarrow$ current customer $\neq$ previous customer. Another error occurs toward the end of the diagram: "Print a customer total line" appears twice—the second one should be changed to "Print a salesman total line".

Table 1. A summary of programming errors of five structured programming techniques

Technique	Input file not empty	Input file empty
A	Wrong	Prints garbage
B	Correct but clumsy	Prints garbage
C	Infinite loop	Terminates correctly
D	Infinite loop	Terminates correctly
E	Wrong	Prints garbage

Finally, if the print file is empty to begin with, the program is wrong as there is no "customer total" nor "salesman total" to print but the program still prints them as garbage.

Technique B: nested if-statement logic (Fig. 2 of [1])

Again, if the input file is empty to begin with, this program still prints "customer total" and "salesman total" which are just garbage. Furthermore, we observe that "Print a customer total line" appears twice in the nested if-statement—there is nothing wrong with these but that they are simply signs of clumsiness.

Technique C: nested perform-loop logic (Fig. 3 of [1])

In the innermost perform-loop, the second conjunct "current salesman = previous salesman" is redundant because current customer = previous customer ⇒ current salesman = previous salesman. Even so, the program as it stands is wrong, as it is trapped in an infinite loop when the following situation occurs:

Previous record: salesman 1, customer 1, sales information
Current record: salesman 1, customer 2, sales information.

In other words, the condition of the innermost loop is always false (because current customer ≠ previous customer), but the condition of the middle loop is always true (because current salesman = previous salesman); there is no "Read a data record" statement to alter the situation, hence the infinite loop.

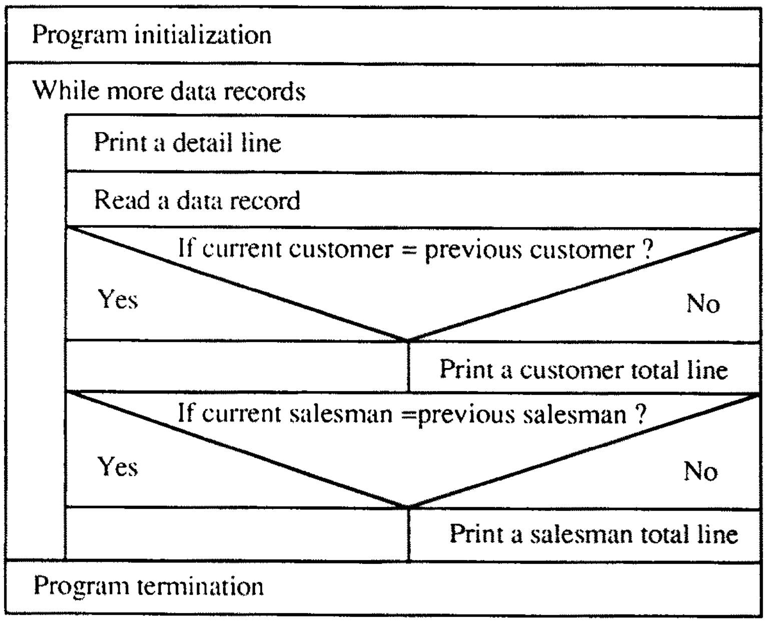
Program initialization
While more data records
Print a detail line
Read a data record
If current customer = previous customer ?
Yes
No
Print a customer total line
If current salesman =previous salesman ?
Yes
No
Print a salesman total line
Program termination

Fig. 1. An elegant programming solution presented in a Nassi–Schneiderman diagram for solving the multi-level control break reporting logic problem.

Technique D: perform-loop with a nested if-statement logic (Fig. 4 of [1])

Apart from the clumsiness of having two "Print a customer total line" statements, the program is also wrong. It is trapped in an infinite loop when the following situation occurs:

Previous record: salesman 1, customer 2, sales information
Current record: salesman 2, customer 3, sales information.

In other words, the condition of the innermost loop is always false (because current salesman $\neq$ previous salesman), but the input file is not depleted yet, implying the condition of the outermost loop is always true, hence the infinite loop. Again, there is no "Read a data record" statement to alter the situation.

Technique E: if-statement with a nested perform-loop logic (Fig. 5 of [1])

The program as it stands is wrong. The two arms of the if-statement should be interchanged. Beside the clumsiness of having two "Print a salesman total line" statements, we observe that the program is also incorrect when the input file is empty to begin with—it prints a salesman total line which contains garbage.

We summarize all five techniques and their programming errors in Table 1. To our disappointment, none of the programs using the structured programming techniques is correct.

AN ELEGANT PROGRAMMING SOLUTION

We now present an elegant programming solution to the multi-level control break reporting logic problem using the Nassi–Schneiderman diagram as shown in Fig. 1. It is not based on assumptions (ii) and (iii), but relies on assumption (v).

Its elegance may be appreciated as follows:

(a) The previous salesman and previous customer need not be initialized to some artificial values.
(b) If the input file is empty to begin with, the program terminates correctly without printing garbage.
(c) No duplication of statement nor redundancy may be found in the Nassi–Schneiderman diagram.
(d) A test is carried out before printing a total line. This applies both to a customer total line and a salesman total line.
(e) The solution is transparent and straightforward with only one loop.

CONCLUDING REMARKS

Out of five structured programming techniques which purport to solve the multi-level control break reporting logic problem, none of them has been found to be free of programming errors—either it does not terminate on a legitimate input file, or it prints garbage on an empty input file.

Fortunately, we are happy to report an elegant programming solution to the multi-level control break reporting logic problem. Its logic is simple, transparent and pleasing.

It is our contention that some "structured" programs are complex and hard to understand because their logics are wrong or entangled as Headrick and Fowler's paper[1] shows.

Acknowledgement—The use of KFUPM's facilities in undertaking this research and in preparing this paper is gratefully acknowledged.

REFERENCE

1. Headrick R. W. and Fowler G. C., Structured programming techniques: a study of relative complexity. *Computers Educ.* **12,** 539–544 (1988).