Structured D-chart: A diagrammatic methodology in structured programming

by C. JINSHONG HWANG
Purdue University
W. Lafayette, IN

ABSTRACT

The rules and principles of structured programming resemble the rules and principles of good musicianship. Good programmer performance depends on both a competent programmer and the proper logic design methodologies. This paper presents a new diagrammatic methodology for such programming that accurately depicts the restricted control structures and their close correlation with natural thought process. A good programming style and coding indentation are the direct results of the use of structured D-charts.
1. INTRODUCTION

During the 1970s the structured programming revolution produced several significant results:

a. The principle of good programming was universally accepted.\(^5\)

b. The use of restricted control structures and top-down programming were widely accepted methods.\(^6\)

c. The flow chart was developed as a schematic depiction of restricted control structure specification of program logic.\(^7\) The flow chart depictions are shown in Figure 1.

```
Restricted Control Structure

Sequential Structure

IF - THEN - ELSE

Statement-Group-n

Statement-Group-m

Statement-n

Statement-n + 1

Selective Control Structure

DO - WHILE

Repetitive Control Structure

Figure 1—Flow chart as a schematic depiction of restricted control structure specification of program logic
```

The IF-THEN selective structure is a particular case of the IF-THEN-ELSE selective structure. The CASE structure is a generalized case of the selective structure. DO-UNTIL is an alternative repetitive structure.

The rules and principles of structured programming resemble the rules and principles of good musicianship. A good musical performance depends on both competent musicians and proper instruments. The currently most popular instrument for depicting program structure and logic is the conventional flow chart described above. The controversies surrounding structured programming and the GOTO statement\(^8\) pertain mostly to the use of such an instrument.

The use of flow charts to depict program structure and logic can make it easier for a programmer to violate the single-entry, single-exit rule for programs and program modules. The composition rules for flow charts make it possible to draw a chart, using lines that cross one another and move off in all directions. Flow charts are not very suitable for showing how structured algorithms closely reflect natural thinking and problem-solving processes.

Other instruments introduced to represent structured programming include pseudocode and the Nassi-Shneiderman chart,\(^9\) or something similar like the Chapin Chart.\(^9\) Pseudocode, however, is not a diagrammatic visual aid for designing program logic. The Nassi-Schneiderman chart does not indicate the logic flow or progression in a clear, concise flow manner. All these instruments, of course, have their advantages and disadvantages, but this article presents a new diagrammatic methodology for structured programming that accurately depicts restricted control structures and their close correlation with natural thought processes. Parallel with the structured D-charts presented herein, pseudocode will be used to express the meaning of structured D-chart in a disciplined, restricted narrative. The original idea for the D-chart appeared in Bruno and Steiglitz.\(^1\) (The "D" in D-chart is in honor of Edsger W. Dijkstra, who was one of the earliest proponents of structured programming.)\(^3\) Certain revisions for the idea of D-chart appeared in Denning and Denning.\(^2\) This article presents a new revision of the D-chart, called the structured D-chart, which can be used by all levels of programming students and professionals. The structured D-chart was developed by the author in fall 1978, and teaching experiments using structured D-charts have continued for four years. At the end of this article, some results of these experiments will be presented. The next section will present the composition of the structured D-chart with respect to control structures and the meaning of the symbols used in the structured D-chart. Section 3 will discuss the implementation of the structured D-chart in non-structured FORTRAN, COBOL, BASIC-PLUS, and PASCAL. In that section we shall see that the use of the structured D-chart is universally applicable to every kind of programming language. Section 4 will describe the relationship between the structured D-chart and programming style.\(^10\) Section 5 will present some simple rules for using the structured D-chart. Section 6 will summarize the advantages of the structured D-chart and provide a direct comparison between the structured D-chart and the flow chart. Finally, the results of the teaching experiments will be described.
2. STRUCTURED D-CHARTS AND RESTRICTED CONTROL STRUCTURES

2.1 Structured D-chart Symbols

This section gives a detailed description of structured D-charts by illustrating and explaining the symbols used to construct them. Structured D-chart representation of control structures will be emphasized.

Structured D-charts are made up of a limited set of special geometric symbols, corresponding to specific parts of a program unit. These symbols and their meanings are as follows:

An oval (○) indicates the starting and ending point for the program unit and a return to a main program from a subroutine.

A parallelogram (□) indicates general input-output operations, the input, reading, and printing of data.

A rectangle (□) indicates assignment and arithmetic operations, where the assignment of values and computation of arithmetic operations occur.

A large dot shows the upper boundary or lower boundary of a selective control structure and designates the point at which this control structure begins or ends. All statements within this control structure will be executed, depending upon the status of a certain condition. Dots must appear in pairs to indicate one entrance into and one exit from the selective control structure.

Two or more arrows emanating from a large dot and diverging downward indicate the multiple alternate paths of a selective control structure. Each diverging arrow eventually converges into another dot which marks the end of the selective control structure.

A small circle indicates the top boundary or delimiter of repetitive control structures and indicates at what point the repetitive control structure begins. All statements contained within a repetitive control structure are executed according to the status of specific condition. The circle contains an alphabetic character to uniquely identify each repetitive control structure.

A small triangle indicates the lower boundary or delimiter of repetitive control structures, showing at what point the repetitive control structure ends. The triangle contains an alphabetic character matching the character in the top delimiter of the same repetitive control structure.

An arrow is used to indicate the flow of the D-chart. In a repetitive structure, the flow should always be to the right and down. In a DO-WHILE repetitive control structure, the conditions that determine the control structure flow will be written directly above the horizontal arrow.

This figure indicates an interrupt exit from a repetitive control structure to the first executable statement immediately following the repetitive control structure. The alphabetic character in the circle identifies the repetitive control structure from which the exit is to be made.

A rectangle of broken lines indicates that the control structure causes an automatic increment. The auto-increment is part of the DO-FROM-TO control structure. (DO-FROM-TO is an alternate form of repetitive control structure; see later in this section.)

This block figure indicates that control is passed to a subroutine, procedure, or a block of program statements, located in a separate structured D-chart. It indicates a CALL to a subroutine. Subroutines end with an oval, indicating a RETURN to the main program.

This figure indicates an implied repetitive control structure for input or output from a collection of related data items in an array.

A connector symbol indicates the continuation of the structured D-chart on another page. It should not be used for the branching of execution control. It is only used for the connection of pages. One symbol at the end of the first page and another symbol at the beginning of the second page. Numerals inside the connection indicate the location of connection.
2.2 Restricted Control Structures

Sequential control is the simplest type of control structure: control goes from statement to statement in a straight uninterrupted line. The structured D-chart and pseudocode in Figure 2 show the flow of control in a sequential control structure.

Sequential Control Structured D-Chart

<table>
<thead>
<tr>
<th>Statement-m</th>
<th>Statement-n</th>
</tr>
</thead>
</table>

In Figure 2b, after the execution of statement-\( m \), the status of condition is evaluated. A status of TRUE causes the execution of statement-group-1, followed by the execution of statement-\( n \). If the status is FALSE, statement-group-2 is executed, followed by statement-\( n \). In either situation, statement-\( m \) and statement-\( n \) are executed.

Case 2 IF — THEN — ELSE

In Figure 2c, after statement-\( m \) is executed, the expression is evaluated. Assume the values of expression are positive integers between 1 and \( i \). The line numbers are the statements or statement groups to which control is to be passed according to the integer value. In this example, if the value is 1, statement-group-1 is executed, followed by statement-\( n \). A value of 2 transfers control to statement-group-2. The value determines the flow of the program. Statement-\( m \) and statement-\( n \) are executed regardless of the value.

Case 3 CASE Construct

In Figure 2c, after statement-\( m \) is executed, the expression is evaluated. Assume the values of expression are positive integers between 1 and \( i \). The line numbers are the statements or statement groups to which control is to be passed according to the integer value. In this example, if the value is 1, statement-group-1 is executed, followed by statement-\( n \). A value of 2 transfers control to statement-group-2. The value determines the flow of the program. Statement-\( m \) and statement-\( n \) are executed regardless of the value.
Figure 4—Structured D-charts and pseudocode for repetitive structures
2.2.3 Repetitive Control Structured D-Chart

The repetitive structure is the third type of control structure. The distinctive characteristic of a repetitive structure is that it causes a statement or group of statements to be executed repeatedly, according to the value of a specified condition. There are four kinds of repetitive structures: the DO WHILE structure, where statements are executed repeatedly, while the logical value of the specified condition is TRUE; the DO UNTIL structure, where a group of statements is executed repeatedly, until the logical value of the specified condition becomes TRUE; the REPEAT UNTIL structure, where the condition is checked at the end of the repetitive structure while the similar DO UNTIL structure checks the condition at the beginning of the structure; and the DO FROM-TO structure, where a group of program statements is executed for a specified number of times. The structured D-charts and pseudocode for these repetitive structures are as shown in Figure 4, Cases 1-4.

After statement-m is executed, the value of condition is evaluated. While the logical value of condition is TRUE, the statement-group is executed. As in the selective control structure, the statement-group in a repetitive control structure can be one or more statements. When the logical value of condition becomes FALSE, control is transferred to statement-n.

The execution of statement-m is followed by the evaluation of the specified condition. Until the logical value of the condition becomes TRUE (i.e., while it is FALSE), the statement-group is executed. When the logical value of the condition becomes TRUE, control is transferred to statement-n.

The REPEAT UNTIL structure is a special case of the repetitive structure which is used to ensure that a given statement or statement block within a repetitive structure will be executed at least once. Some programming languages do not provide a formal statement or set of statements to accomplish this task efficiently. REPEAT-UNTIL structure provides this special case of the repetitive structure. Despite its name, the REPEAT-UNTIL structure does not contain the UNTIL statement in most languages; it is made up of an IF-THEN-ELSE selective structure. The structured D-chart and pseudocode for the REPEAT-UNTIL structure are as follows.

After entering the loop and the execution of statement-group, the conditional expression is evaluated. If the logical value is TRUE, the loop exits and statement-n is executed. A logical value of FALSE causes statement-group to be executed again, thus creating a repetitive structure. This iterative process is repeated, until the logical value becomes TRUE, at which point the loop exits. The conditional expression is not evaluated, until statement-group has been executed for one time. This guarantees that statement-group will be executed at least once. In the DO-UNTIL control structure, on the other hand, the logical value is determined prior to the execution of the statement group contained in the repetitive structure. The REPEAT-UNTIL structure eliminates repetitive code to guarantee one execution of a statement block. This special case of repetitive control structure can resolve the controversies over the proper use of the GOTO statement. The GOTO statement may be used if for the purpose of implementing REPEAT-UNTIL structure in a language without REPEAT or for the error handling.

After the execution of statement-m, the value of expr-1 is assigned to variable and statement-group is executed. The value of variable is then incremented by the value of expr-3 and variable is evaluated to determine if it exceeds the value of expr-2. Statement-group is repetitively executed, until the value of variable becomes greater than the value of expr-2. At that point, control is transferred to statement-n.

3. IMPLEMENTATION OF STRUCTURED D-CHARTS

This section demonstrates the implementation of structured D-charts in programming languages such as nonstructured FORTRAN, COBOL, BASIC-PLUS, and PASCAL. The example algorithm is the bubble sorting algorithm, which reads a set of numbers until an end of file marker is encountered, sorts the numbers in ascending order and prints the sorted result. The logic involves a combination of all three restricted control structures. The same structured D-chart (Figure 5) for the bubble sorting algorithm will be used to implement the algorithm in four different programming languages. Further-
Figure 5b---Implementation in PASCAL

```pascal
PROGRAM SORT (INPUT, OUTPUT);
VAR K, I, J, N, TEMP : INTEGER;
    NUM : ARRAY[1..10] OF INTEGER;
BEGIN
    N := 10;
    FOR J := 1 TO N DO READ (NUM[J])
    K := N - 1;
    WHILE K >= 1 DO
        BEGIN
            FOR I := 1 TO K DO
                IF NUM[I] > NUM[I+1] THEN
                    BEGIN
                        TEMP := NUM[I];
                        NUM[I] := NUM[I+1];
                        NUM[I+1] := TEMP;
                    END;
            K := K - 1;
        END;
    FOR I := 1 TO N DO
        WRITELN (NUM[I]);
END.
```

Figure 5c—Implementation in BASIC-PLUS

```basic
100 DIM NUM(10)
110 N = 10
120 FOR J = 1 TO N
130 READ NUM(J)
140 NEXT J
160 FOR IX = 1 TO N
170 KX = IX
180 WHILE KX >= 1
190 FOR IX = 1 TO KX
200 IF NUM(IX) <= NUM(IX+1) THEN
210 NEXT IX
220 IF NUM(IX) <= NUM(IX+1) THEN
230 TEMP = NUM(IX)
240 NUM(IX) = NUM(IX+1)
250 NUM(IX+1) = TEMP
260 NEXT IX
270 NEXT KX
280 NEXT IX
290 FOR IX = 1 TO N
300 PRINT NUM(IX)
310 NEXT IX
320 DATA 823, 791, 768, 587, 456, 345, 268, 212, 123, 100
370 END
```

Figure 5d—Implementation in nonstructured FORTRAN

```fortran
INTEGER I, J, N, K, NUM(10), TEMP
OPEN (UNIT=1, NAME='TEST.DAT', TYPE='OLD', READONLY)
N = 10
READ(1,*) (NUM(J), J = 1, N)
K = N - 1
IF (K .LT. 1) GOTO 35
DO 30 I = 1, K
    IF (NUM(I).LE. NUM(I+1)) GOTO 30
    TEMP = NUM(I)
    NUM(I) = NUM(I+1)
    NUM(I+1) = TEMP
30 CONTINUE
K = K - 1
GOTO 25
DO 40 I = 1, N
    WRITE(7,*) NUM(I)
40 CONTINUE
CLOSE (UNIT=1, DISPOSE='SAVE')
STOP
END
```

poses. The structured D-chart not only clearly reflects natural thinking via restricted control structures, but also accommodates the indentation requirements of program style in a very direct way. The use of indentation to indicate control structures in a program is one of the elements of good programming style. The degree of indentation depends upon the programmer, but the level of indentation should correspond to the level of the control structures within the program. The structured D-chart (Figure 6) illustrates what is meant by the level of a control structure.

In structured programming, a level refers to a series of sequential statements. Whenever a series is broken by a control structure (DO WHILE, DO UNTIL, DO FROM-TO, IF-THEN or IF-THEN-ELSE) all subsequent statements belonging to that control structure are considered to be on another level, and should be indented accordingly. All statements of the same level should be indented the same number of spaces. For example, the pseudocode in Figure 7 corresponds to the structured D-chart (Figure 6), and illustrates what the indentation should look like.

Note that all statements of a given level are indented the same number of spaces. All Level 1 statements are not indented, all Level 2 statements are indented 5 spaces, and all Level 3 statements are indented 10 spaces. If there were a Level 4, it would be indented 15 spaces, and so on for any other levels. The number of spaces of indentation for each level is up to the programmer, as long as it makes the structure clear. Compare the pseudocode with indentation (Figure 7) to pseudocode in Figure 8. Note how much easier the indented pseudocode is to read, how each control structure is more clearly defined, and how the structured D-chart is used to reflect the indentation levels of the programming style. The logic of this pseudocode is much harder to follow, because of the lack of indentation. The use of indentation is a very powerful tool in the writing of clear and easy-to-read programs. The example in Figure 8 on control structure levels clearly points out another advantage of using the structured D-chart. Indentation for good programming style comes very naturally from the logic design of the structured D-chart. The control levels of structured D-charts directly tells the programmer...
Structured D-Chart

where to use indentation in programming languages (and in pseudocode).

5. RULES OF STRUCTURED D-CHART COMPOSITION

There is great potential for program flexibility when restricted control structures are used to create a structured program. Structured D-charts are essential to the writing of effective structured programs, and should be understood and used as a basis for good programming practices. The following is a list of rules helpful in creating structured D-charts.

1. A structured D-chart must consist of one or a combination of any of the three types of control structures.
2. There must be only one entrance to and one exit from each selective control structure.
3. Each selective structure must be identified by a pair of large dots to symbolize the entrance into and the exit from selective control structures.
4. There must be only one entrance and one exit for each repetitive structure with the exception of (Exit @), in which case the exit must be the first executable statement following the repetitive structure @, which may be a nested outer repetitive structure.
5. Each repetitive structure must be uniquely identified by a different alphabetic character within the boundary symbols (○) and (△) for the repetitive structure.
6. The logic of the structured D-chart must proceed from top to bottom with any repetitive structures depicted to the right.
7. A control structure can completely contain another control structure. This is called nested structure. A control structure may not contain only a portion of another control structure. There may be no flow lines drawn to connect one control structure to another external structure or outer (nested) structure, the only exception being the beginning or ending boundary.
8. The GOTO statement may be used to implement a control structure. This is the only time the GOTO statement may be used for the sole purpose of implementing control structure. This is the only time the GOTO statement may be used to implement a control structure. This is the only time the GOTO statement may be used to implement a control structure. This is the only time the GOTO statement may be used to implement a control structure.
9. The structured D-charts in Figure 9 describe the illegal (left column) and the corrected D-charts (right column.)

6. STRUCTURED D-CHART VS. FLOW CHART

The structured D-chart is superior to the flow chart because it agrees completely with the restricted control structures in structured programming techniques. The logic of a structured program and a structured D-chart involves only three restricted control structures: sequential, selective, and repetitive. Implementing the logic of a structured D-chart as a structured program is a direct one-to-one translation. Structured D-charts are easier to read than flow charts because execution flow always proceeds through a structured program in a downward direction; there are no crossing or upward-pointing

Figure 5e—Implementation in COBOL
Figure 6—Level of a control structure

Figure 7—Illustration of indentation in pseudocode

Figure 8—Pseudocode without indentation

lines. All structured D-charts are drawn with single-entry, single-exit control structures, and all conditions are explicitly stated in words similar to those found in the actual program code.

Figure 10 illustrates a conventional flow chart, showing the logic necessary to read a list of number, sort the numbers into ascending numerical order, and print the results. Figure 10 is

---

LEVEL NO.   INDENTATION
1   Statement Group 1
1   DO WHILE condition TRUE
2   DO UNTIL condition TRUE
3   Statement-Group-9
2   END-DO
2   IF condition TRUE
3   THEN Statement-Group-6
3   ELSE Statement-Group-7
2   Statement-Group-8
1   END-DO
1   Statement-Group-2
1   IF condition TRUE
2   THEN Statement-Group-3
2   ELSE Statement-Group-4
1   Statement-Group-5

Statement Group 1
DO WHILE condition TRUE
DO UNTIL condition TRUE
Statement Group 9
END-DO
IF condition TRUE
THEN Statement Group 6
ELSE Statement Group 7
Statement Group 8
END-DO
Statement Group 2
IF condition TRUE
THEN Statement Group 3
ELSE Statement Group 4
Statement Group 5
a well-written flow chart, doing the best possible job of depicting program logic, given the inherent weaknesses of flow charts. Note that it includes crossed lines, lines moving right, and an upward-pointing flow of execution control. The logic is hard to follow and difficult to translate into a structured program.

Figure 5 is a structured D-chart representing the same logic as Figure 10. No lines are crossed; all control structures are single-entry, single-exit; control is never transferred upward; and repetitive structures and their conditions are clearly shown. The control structures in Figure 5 are shown in such a way that code blocks and level breaks (used for the indentation of program lines) are explicitly indicated. The structured D-chart makes it easier to implement structured technique and good programming style.

Real application programs are larger and far more complex than any found here. As a result, the logic required to produce the algorithms for such programs is larger and more complex, sometimes consisting of many pages. Because structured programming is a superior method for creating large programs that are effective and efficient, structured D-charts should be used to represent program logic. A programmer can move from a structured D-chart to a structured program quite easily, for the structured D-chart is based on the same concepts and uses the same control structures in structured programming.
7. EXPERIMENTS IN USING STRUCTURED D-CHARTS

In fall 1978 the programming curriculum for the Department of Computer Technology at Purdue University was established. The department undertook to offer an application programming education based upon an understanding of conceptual foundations. The ability to use structured D-charts to express program logic with restricted control structures in the program designing phase is an essential for the conceptual foundations. During the past four years of teaching structured D-charts to freshmen and all new-entry students, we experienced great success with the structured programming method. All students used the structured D-chart method in all programming courses and were informed concerning flow charts during the second year of the curriculum in order that they would be able to communicate with other computer professionals. But most students have learned about the use of the flow chart before entering our program. They would have to unlearn the nonstructured programming technique of using flow charts.

The following statistics are based upon a survey made in December 1980 of 148 randomly selected students who had had one or more semesters' experience using structured D-charts in writing structured programs.

1. 16 freshman-level students did not know what a flow chart was. They only understood the usage of structured D-charts.
2. 132 freshman-, sophomore-, and junior-level students had a knowledge of flow charts in addition to structured D-charts. Among them,
   a. 86 students learned the use of flow charts before entering our program.
   b. 46 students were exposed to flow charts after learning about structured D-charts.
3. Students were asked to compare structured D-charts with flow charts, if they knew both methodologies.

Question: If you know about structured D-chart as well as flow charts, give a letter grade to the usage of structured D-charts and flow charts in terms of overall performance in logic design, debugging, program understanding, program-
A second survey was made in November 1981 of 138 randomly chosen students who had had one or two semesters' experience using structured D-charts. Its results were very close to the results of the survey made in 1980. Its corresponding figures are as follows:

1. 16 freshmen students did not know what a flow chart was. They only understood the use of structured D-charts.
2. 122 freshman-level students knew the methodologies of flow charts as well as structured D-charts.
   a. 84 students learned about flow charts before entering the program and then learned the use of structured D-charts.
   b. 38 students learned structured D-charts from the very beginning and learned about flow charts later in the year.
3. The distribution curve of 11 and 12 above can be figured as shown in Figures 13 and 14. The figures are very similar to those in the first survey except that the gap between the ratings for the structured D-chart and the flow chart for the 0.5 years in Figure 14 is much closer (0.51). The possible interpretation is that there were only 8 students in that category and that the accuracy of that category might have been affected by one or two students or by simple, inadvertent error entered into the survey.
8. ACKNOWLEDGMENTS

The author is indebted to the following individuals for their contributions and suggestions to this article: Phil A. Beetley, Darryl E. Gibson, Jeff Whitten, and all other professors in the Computer Technology Department for their assistance in the survey.

REFERENCES