

**Program Design with
Structure Charts
&
Flowcharts**

Contents

1. STRUCTURED PROGRAM DESIGN METHOD	1
1.1 Logic Components of Structured Programming	1
1.2 Hierarchal Structures.	2
1.3 Programs Based on Problems.	3
1.4 The Development of Program Structures.	3
1.5 Pseudo Code	7
1.6 Summary	8
2. FLOWCHARTING PRINCIPLES - GENERAL CONCEPTS	9
2.1 Flowchart Symbols	11
2.2 Standardisation of Flowchart Symbols	11
2.3 Flowcharting Aids	13
2.4 The Flowcharting Template	13
2.5 The Flowcharting Forms	13
2.6 A Dry Run	13
2.7 Basic Flowcharting Rules	15
2.8 Summary	15
3. FLOWCHARTING TUTORIAL	16
3.1 Notations	16
3.2 Input/output, Loops & Calculations	17
3.2.1 Illustrative Problem 1	17
3.2.2 Looping and Last Record Check	20
3.2.3 Student Problem 1	22
3.3 Headings, Switches, Accumulation of Total & Dry Running	23
3.3.1 Accumulation of Totals	23
3.3.2 Illustrative Problem 2	23
3.3.3 Headings	25
3.3.4 Switches	25
3.3.5 Dry Running	28
3.3.6 Student Problem 2	29
3.3.7 Illustrative Problem 3	30
3.4 Control Breaks, Sub-routines, Initial Record Read & Compare Areas	32
3.4.1 Control Breaks	32
3.4.2 Control Codes	32
3.4.3 Student Problem 3	32
3.4.4 Subroutines	33
3.4.5 Initial Record Read	33
3.4.6 Illustrative Problem 4	34
3.4.7 Student Problem 4	39

1. STRUCTURED PROGRAM DESIGN METHOD

1.1 Logic Components of Structured Programming

The term "Structured Programming" encompasses a wide variety of ideas; we are concerned with one of the more fundamental concepts. This concept is that programs should consist of logic constructed from only three constructs or component types:

Sequence (Figure 1)

A is a sequence of 3 parts: B, C and D always in that order.

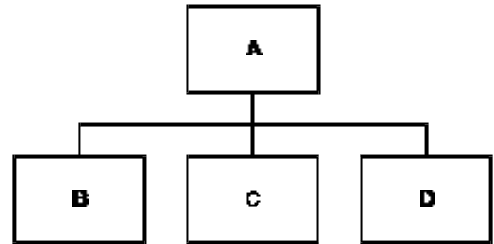


Figure 1

Selection (Figure 2)

A is a selection of either B, C or D. Component A can only be one of B, C or D at a time. The symbol 'o' in the top right hand of the box denotes a selection.

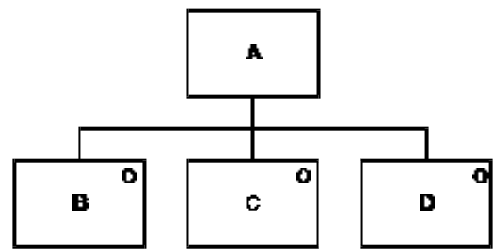


Figure 2

Iteration (Figure 3)

A consists of Zero, One or Many occurrences of the component B. A is an iteration of B. The symbol '*' denotes this.

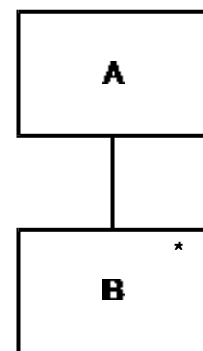


Figure 3

ELEMENTARY components are those which cannot be further divided OR a decision is taken not to divide them further. In the above sequence diagram B, C, and D are elementary components (boxes, items), A is a non-elementary component.

On a program structure the functions of the program can only be attached to elementary components.

These three constructs were introduced for building structured programs. We can use these same components for describing the data upon which the program operates and from which the program structure is derived.

1.2 Hierarchical Structures.

In order to form a data or program structure we arrange the three constructs in a hierarchical fashion, from the top down. For example:

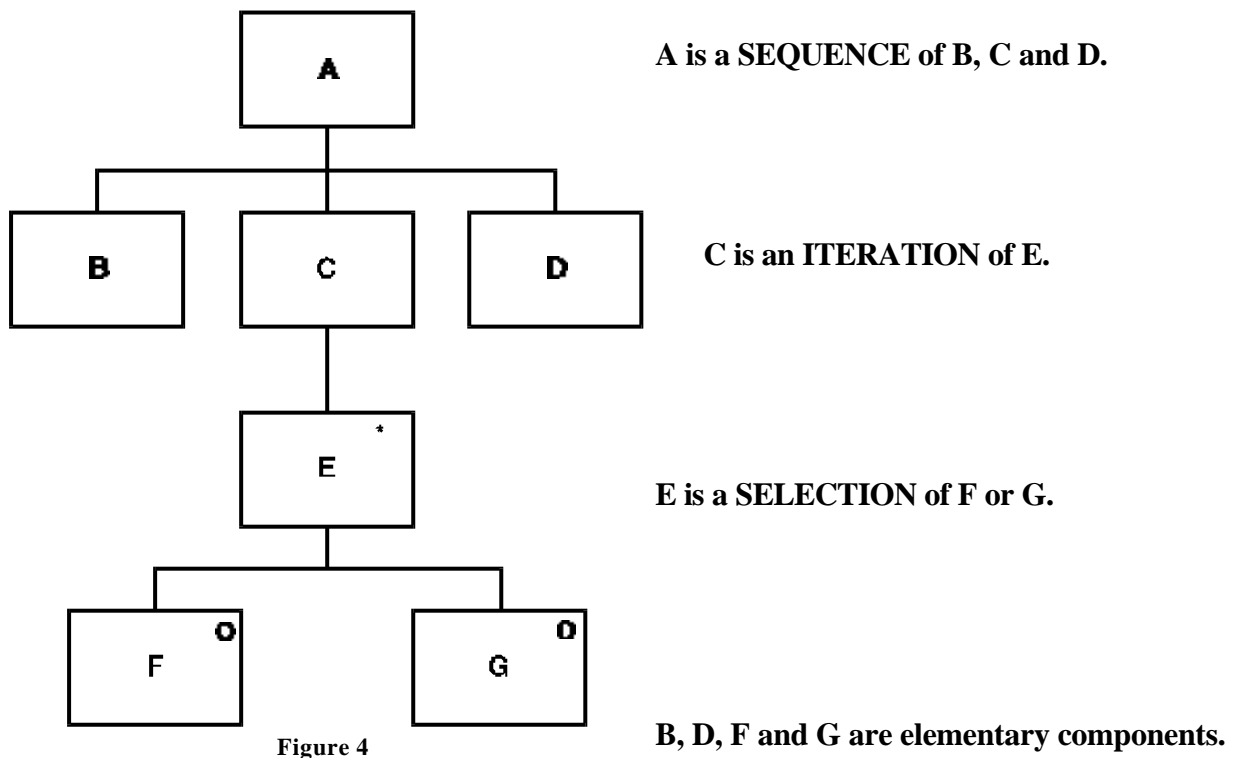


Figure 4

An important property of this type of design is that there is only one path from the highest level to any lower level component. The path through any structure designed according to these rules is well designed and immediately apparent. The flow of logic does not branch from one part of the structure to another; in programming terms we have no uncontrolled GO TO's.

1.3 Programs Based on Problems.

Every program we wish to design is concerned with a particular problem. It is essential that the program should fit the problem. We must hilly understand the problem structure before we can design the program structure.

The best way of getting at the problem structure is to consider the data structure; not just the physical structure of the data but the logical structure relevant to the problem we wish to solve.

We can use the same notation to describe both the data structure and the program structure. The key to Structured Program Design is that the program structure should bear close resemblance to the (problem orientated) data structure. In fact, the closer the resemblance, the better.

1.4 The Development of Program Structures.

The first stage of Structured Program Design is to represent the problem as a logical data structure. Sometimes this data structure serves as a program structure without the need for changes, more often a few small additions are needed. For example most programs require some sort of start and end procedures.

Consider the following example. We wish to count all the dead people on a file of records; there is one record for each person that contains an indicator set to I for "dead" and 0 for "alive".

PHYSICAL DATA STRUCTURE

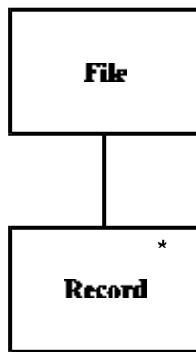


Figure 6

LOGICAL DATA STRUCTURE

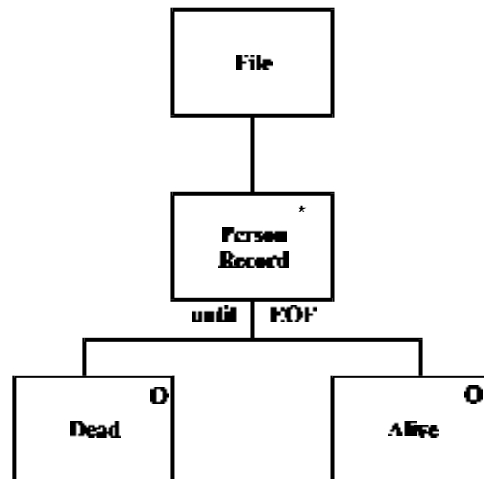


Figure 5

The Physical data structure does not help us with the problem, the logical data structure on the right represents the problem we must deal with In order to form the program structure we have to introduce INITIALISE, MAIN BODY and FINISH components above record.

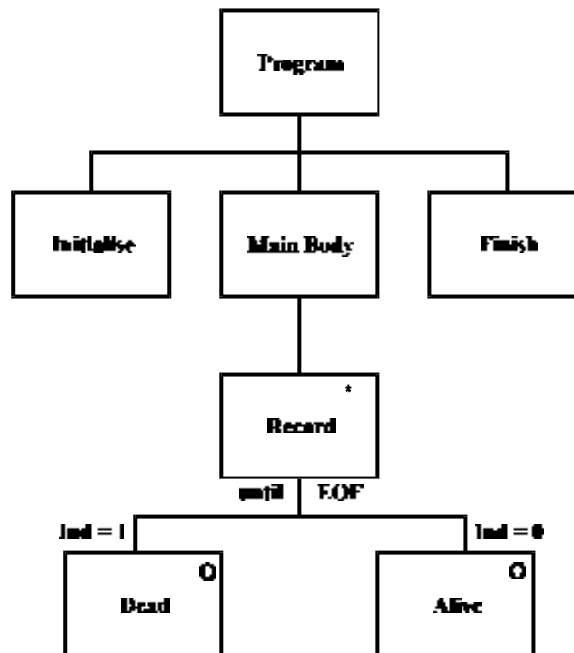


Figure 7

The next stage is to determine the functions of the program and list them. Function List

1. Open File
2. Close File
3. Read Record
4. Move 0 to Dead Count
5. Add 1 to Dead Count

We can now place each function with the appropriate Elementary component(s) of the program structure.

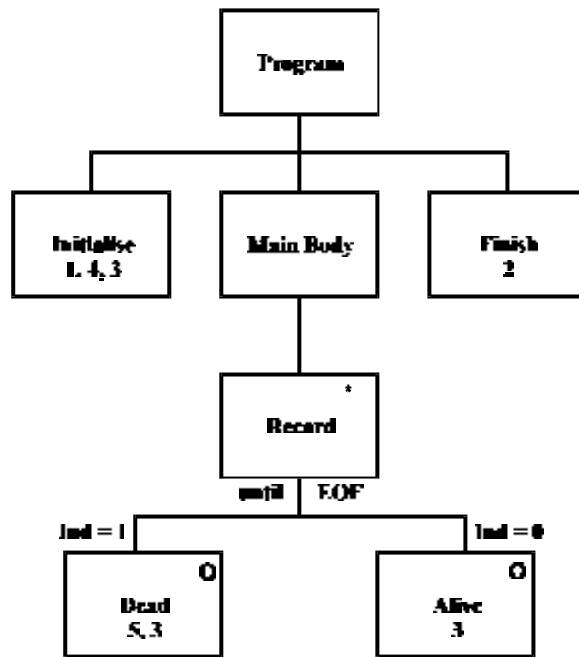


Figure 8

It is important to understand the positioning of the "read" statements, There are two rules for applying "read" statements to program structure. Firstly we must "read ahead", that is get the first record into store at the very start of our program. Secondly after we have finished the processing for a record the last function will be to fetch the next record, or "read replace".

Finally, consideration could be given to an "optimisation" of this program structure, which will reduce the store size of the program, but increase its run time.

In our example we can save the duplication of function 3 by expanding the program structure thus:

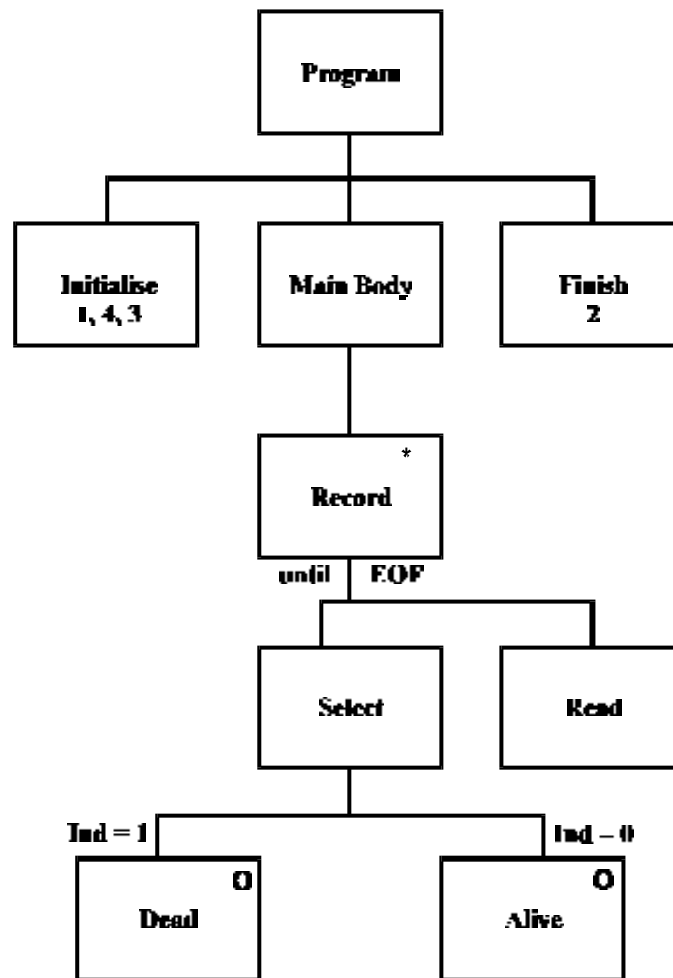


Figure 9

The inclusion of a new level in the structure may be justified on the grounds that we are creating a component for the special file handling function “read”. In the same way the highest level of Initialise, Main Body and Finish may be regarded as compulsory, to cater for the file-handling statements such as "open" and “close”

However, every time that we introduce new components into a program structure, its clarity and amenability suffers; the reason being that it becomes less related to the logical data structure of the problem. It is recommended that such "optimisations" should be kept to a minimum.

1.5 Pseudo Code

Pseudo code is the narrative equivalent of the hierarchical structure diagram. It is a high level language of very simple construction, to continue with the un-optimised example the pseudo code equivalent would be:

PROGRAM:

Begin:

```
Open file, Move zero to dead count, Read record
{ DO WHILE not EOF
    IF Indicator = 1
        Add 1 to Dead count
        Read record
    End IF
    IF Indicator 0
        Read record
    End IF
} END DO
Close file
END OF PROGRAM
```

Three points to note

1. The code is indented to reflect the levels of the program structure.
2. Elementary component titles are omitted.
3. Iterations are bracketed to show logic flow.

The word 'DO' should be applied to components that are expanded elsewhere, it does not imply that a component will necessarily become a module or subroutine.

1.6 Summary

Steps of Basic Structured Program Design

There are three basic steps in the design of a program

1. *Define the data structure(s).*

This step involves the representation of the data as a hierarchical structure using the three component types; sequence, selection and iteration.

2. *Create the program structure.*

This step involves adding to the data structure components which the problem specification shows are necessary. Such additions should be kept to a minimum.

Also involved at this stage is the inclusion of conditions required to make the selections and iterations in the structure,

3. *Allocate functions to the structure.*

The task must be expressed in terms of executable operations (functions) and these must be allocated to the elementary components of the structure.

It should be noted that while we should not consider the program structure when drawing up a function list the reverse is not true It is necessary to think about what functions are required (from the problem specification) when we construct the program structure (step 2).

To implement the program we may then produce pseudo code or flowchart, which ever is appropriate and turn this into a computer language.

2. FLOWCHARTING PRINCIPLES - GENERAL CONCEPTS

A Flowchart is a diagram that shows a series of steps needed to accomplish a task. A flowchart consists of a series of symbols Connected in lines. The symbols are used to represent steps in an overall process, while the lines, or flowlines, connect the symbols and establish a sequence or order to the steps The symbols and the flowlines are combined in a flowchart to represent the logic of a single program or program function.

Flowcharting is the next step from producing a Structured Diagram. Each component from the structured diagram can be expressed as a flowchart.

The flowchart serves two purposes. First, it is a convenient way to design logic for programs. It is easier to determine and arrange complex logic on a flowchart than it is to find logical errors in a completed program Second, it becomes an important part of the documentation. The documentation is vital if the program is to be properly maintained or if modifications must be made at a later date.

The flowchart in Figure 10 is an example of a program flowchart used to print customers names and addresses.

Flow Chart Worksheet

Programmer _____	Program No. _____	Date _____	Page _____
Module Name Print Customer Names & Addresses		Program Name A Simple Program Flowchart	

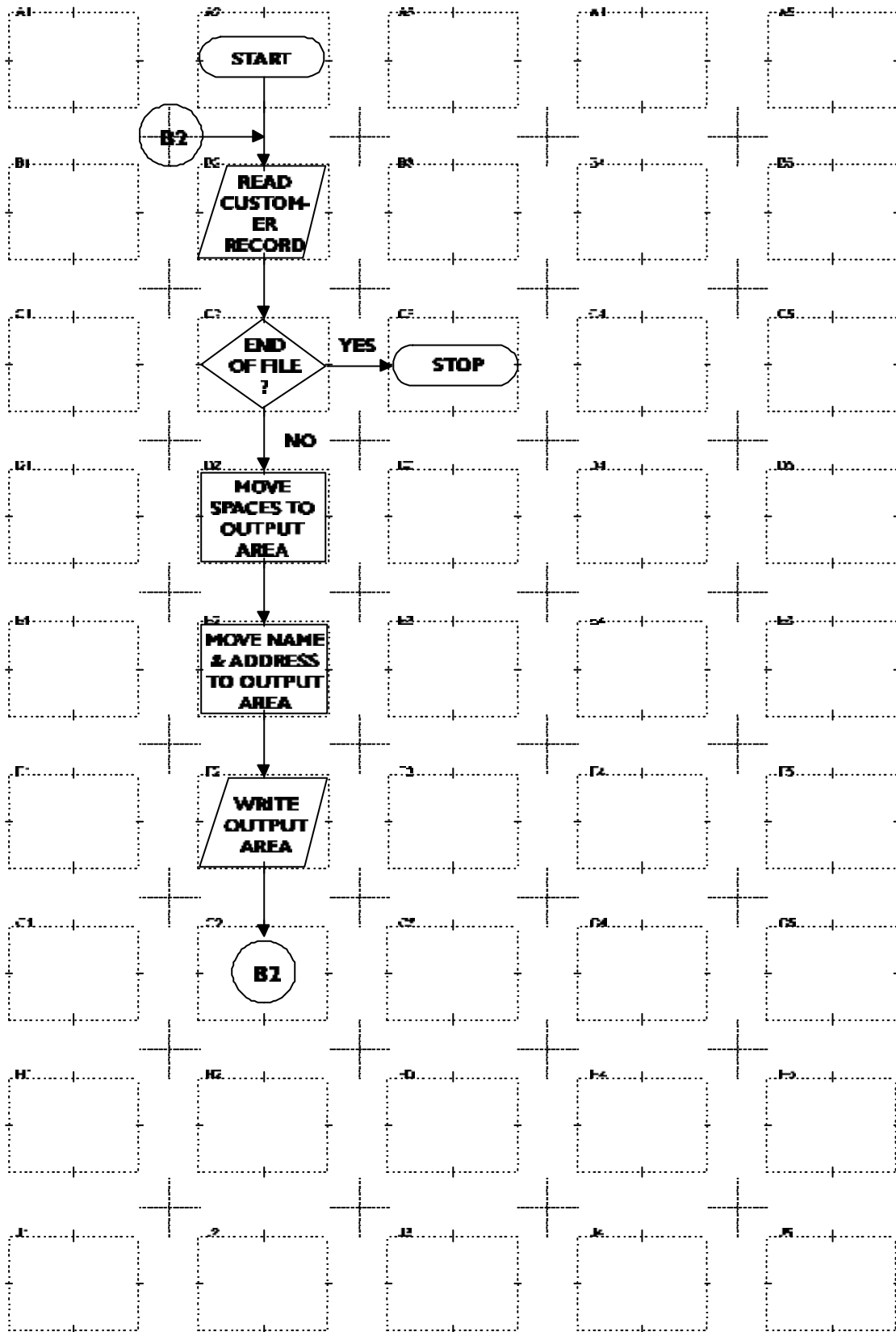


Figure 10

2.1 Flowchart Symbols

Look carefully at the different symbols used in the flowchart in Figure 10.

The lines and arrows are flowlines which show the direction for flow for processing. The two flattened, oval shaped symbols labelled START and STOP are called terminal symbols. They are used at the beginning and the end of the program, or at any point where the program is interrupted. In this chart, terminal symbols are used at the beginning and the end of the program.

Another type of symbol shown in the diagram is the parallelogram. This symbol is shaped like a rectangle leaning to the right and is called an input/output symbol. Input/output symbols are used to show an information exchange between a peripheral device and the computer. In Figure 10 the input/output symbols are labelled READ CUSTOMER RECORD and WRITE OUTPUT AREA. The symbol READ represents input. The other input/output symbol containing WRITE OUTPUT AREA represents output to any device type, in this case a printer.

The third symbol on the diagram is the rectangle labelled MOVE SPACES TO OUTPUT AREA followed by MOVE NAME & ADDRESS TO OUTPUT AREA. Notice that these do not lean to one side as does the input/output symbol. The rectangle is a processing symbol. It is used when the computer is to perform a processing function in the program.

The fourth symbol on the diagram is the diamond shaped labelled END OF FILE?. This is a decision symbol and is a point in the program where a branch is possible. A branch indicates that two directions can be taken from any one point. If the answer to the question END OF FILE? is "yes", the YES flowline is followed. If the answer to the question END OF FILE? is "no" the NO flowline is followed.

The last symbol on the diagram is the small circle labelled B2 both near the top of the flowchart and again at the bottom. These are connectors and are used to continue processing. The contents of the circle contain the identity of the symbol at which processing continues. The arrow on the flowline attached to the connector indicates which is the exit or entry point. In this case B2 in the symbol indicates that processing continues at location B2 ie column B row 2.

2.2 Standardisation of Flowchart Symbols

It is necessary to standardise flowchart symbols so that anyone can read a given flowchart. The data processing industry has developed an international set of standardised symbol definitions. These standardised symbols allow the flowcharts to be intelligible to anyone who has studied the standard. Thus, standardisation improves the value of flowcharts as a communication tool.

It should be noted that flowcharts are used for several different purposes other than software program design. Under different circumstances, e.g. system design, additional or different symbols are used. Throughout this paper "flowchart symbols" refer to a basic standard set used in program design.

Take time now to study Figure 11 which summarises the most common flowcharting symbols. By examining each symbol shape and the associated example usages, you can develop an understanding of the use of flowcharting symbols. It should be noted that although these standard symbols are universally

endorsed, there are often a few minor variations from company to company.

Basic Program Flowcharting Symbols






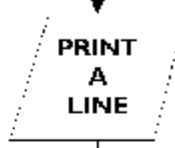


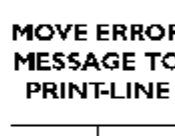

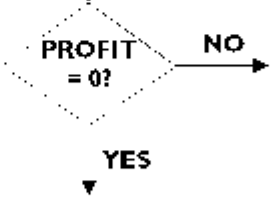
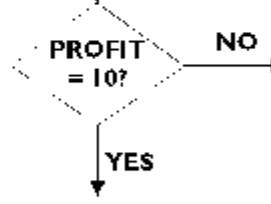

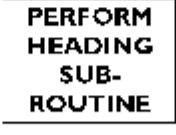




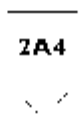
SYMBOL	USE	EXAMPLE	
	TERMINAL Represents the beginning and end of the flowchart		
	INPUT / OUTPUT Represents data either entering or leaving a program		
	PROCESSING Represents the execution of an operation		
	DECISION Note that there should only be two exits from a decision box		
	PREDEFINED PROCESS Represents a group of operations specified elsewhere		
	CONNECTOR Represents a junction point in the flow		
	OFF PAGE CONNECTOR Represents a branch in the flow to page 2, position A4		

Figure 11

2.3 Flowcharting Aids

Now-a-days, it is likely that any serious flowcharts will be done using a Flowcharting Computer Application Initially however, it is a good idea to get use to flowcharting by drawing them by hand. To ease this process Flowchart Templates and Flowchart Worksheets or Forms are used. Also in this section a technique called Dry Running is discussed, this technique applies equally to computer drawn flowcharts.

2.4 The Flowcharting Template

The Flowcharting Template is designed to ease the process of actually drawing the Flowchart. Flowcharting Templates are usually standard in symbol size so that special Flowcharting may be used to layout the Flowchart symbols

2.5 The Flowcharting Forms

Flowcharting Worksheets or Forms are available in a variety of sizes, but all are laid out in a similar fashion. The Forms provide space for documenting information about the Flowchart. A sample is shown in Figure 12.

In addition, the flowcharting form provides a key which may be used to reference symbols on the flowchart. This key or grid is used in much the same way as the grid reference upon a road map The first location in the upper left portion of the form is denoted by A1, the locations to its right are denoted by A2, A3, A4 and A5.

The locations down the first Column are denoted by A1, B1, C1 through J1. This system can prove to be very useful, especially in the case of multi-page flowcharts. In such cases, the grid reference number is prefixed by the page number.

In the sample flowchart worksheet provided the columns and rows are represented by dotted boxes, each box able to hold a single flowchart symbol. The top left box is labelled A1 and the bottom right box labelled J5 in accordance with the numbering conventions described above.

In the flow charting exercises to follow, you will be expected to adopt this convention.

2.6 A Dry Run

Once you have completed your flowchart, the next step is to check that it actually accomplishes what you intended it to The most commonly used technique to perform this check, is known as DRY RUNNING.

A Dry Run may be used to test the logic of both a flowchart and a program. Dry Running a flowchart involves creating a set of typical test data and then running this through the flowchart manually. If the flowchart generates the expected results, then we can be fairly certain that the logic is correct. If not, then its back to the drawing board! More detail on this technique is given in section 3.3.5.

Flow Chart Worksheet

Programmer _____	Program No. _____	Date _____	Page _____
Module Name _____	Program Name _____		

A1	A2	A3	A4	A5
B1	B2	B3	B4	B5
C1	C2	C3	C4	C5
D1	D2	D3	D4	D5
E1	E2	E3	E4	E5
F1	F2	F3	F4	F5
G1	G2	G3	G4	G5
H1	H2	H3	H4	H5
I1	I2	I3	I4	I5
J1	J2	J3	J4	J5

Figure 12

2.7 Basic Flowcharting Rules

Consider the following basic rules for flowcharting:

1. Use standard flowcharting symbols.
2. Precise labelling is important in flowcharting. A descriptive phrase is written inside each flowchart symbol to describe its purpose. Short, accurate phrases enable the reader to understand the flow and process much more quickly. Notice in Figure 10 that by merely reading the symbol labels from top to bottom, a good understanding of the logic is achieved,
3. It is customary in program flowcharting to proceed initially top to bottom and secondly, left to right. This tends to keep the 'main loop' of the program going top to bottom with branches from this going out to the right.
4. Although arrowheads on the flowline are not strictly necessary, they are best included. Please avoid having flowlines going bottom to top or right to left. Although on a simple flowchart this may look acceptable, when the flowcharts you are designing get more complicated, you can end up with something that looks more like a picture of a railway marshalling yard. This is where you use connector symbols.
5. Connector symbols are used to get from one position in the flowchart to another, without abusing the above rules. The usual connection is to 'direct' the logic of the flowchart to a symbol elsewhere by giving its 'address' eg 2B4 symbol B4 on Page 2. We can then show a connector symbol entering the flowchart at this point.
6. Test the logic of your flowchart by performing a dry run upon it.

Draw all flowcharts in pencil - this allows you to change your mind AND the flowchart! You may find it useful to draw in rough first and then copy this onto the flowcharting forms once you are satisfied that is correct.

If you keep these simple rules in mind and remember that the primary purpose of the flowchart is to communicate logic, flowcharting will become easy and intelligible.

2.8 Summary

The flowchart is used to design programs. It is also an important part of the documentation process

Standardised flowcharting symbols should always be used when drawing a flowchart. The basic flowcharting rules should always be followed.

3. FLOWCHARTING TUTORIAL

What follows is a step-by-step guide to the construction of flowcharts from single, straightforward problems to more complex flowcharts dealing with the processing of data to produce detailed printed reports.

There will be a number of practical problems which you will need to complete. These are labelled 'STUDENT PROBLEM' The problems labelled 'Illustrative Problem' have answers and explanations accompanying them These should be read carefully and frilly understood before attempting the Student Problem'. Each Student Problem should be attempted in turn and checked by doing a dry run on it

3.1 Notations

For input/output

- x denotes Alphanumeric characters
- 9 denotes Numeric characters
- z denotes that leading zeroes are not printed.

Whenever a report is to be produced, the items to be printed may be described explicitly on a Printer Layout Chart showing what the finished report will look like, File definition sheets also contain the same notation. Page 29 shows an example of how reports may be laid out on a Layout Chart.

3.2 Input/output, Loops & Calculations

In this Section you will become familiar with the main processes in a program, these include: INPUT Records; OUTPUT Records; How data is transferred from INPUT to OUTPUT Areas; End of File checking; Simple Calculations; The Technique of Program Loops.

The Output area is allocated by the Computer and, as such, there is no way of knowing the initial contents of this area. Consequently it is important to clear this area, i.e. by moving spaces to it, prior to the transfer of the data to be printed from this area,

3.2.1 Illustrative Problem 1

Involves the computation of an employee's gross pay. The input record contains the employee's number, the employee's name, the hours worked by the employee and the employee's hourly rate of pay. The printed output is the employee's number, the employee's name and the computed amount of gross pay.

The formula used in processing is:

$$\text{GROSS PAY} = \text{hours worked} \times \text{rate of pay}$$

In order to construct a flowchart for this problem we must first analyse the problem and determine the main steps involved in the solution,

The problem says'

1. "Involves the computation of an employee' s gross pay".
This does not give us any idea of the steps involved.
2. "The input record contains the employee's number, the employee's name, the hours worked by the employee and the employee's hourly rate of pay"
From this we can determine that we need to access the employee record to obtain the input data.
3. "The printed output is the employee's number, the employee's name and the computed amount of gross pay".
From this statement we can determine that we have at some stage to print these details.
4. "The formula used in processing is: GROSS PAY hours worked x rate of pay".
This specifies the steps involved in obtaining the gross pay, ie. take the hours worked and the hourly rate of pay from the input record and multiply them together to give the gross pay.

Now that we know the steps required to solve the problem we must put them into the correct order. For example, the print line cannot be printed until the gross pay has been calculated as it is part of the output required. The gross pay cannot be calculated before the input record has been accessed. Why?

So, having determined that the main steps these can then be put into a logical sequence:

STEP 1 to access the input record
STEP 2 to calculate the gross pay
and STEP 3 to print the data.

We can then flowchart the problem, which as you will see requires that the steps above be further broken down before the final result is achieved. (see Figure. 13).

The method of breaking a problem into simple steps and further breaking those down can be applied, not only to a very simple example as described but also to much more complex problems. For example, a complete computer system could also be seen in this light ie. break the whole system down into functions, further breakdown the individual functions into programs, the programs into modules, the modules into steps, the steps into instructions etc.

Flow Chart Worksheet

Programmer _____	Program No. _____	Date _____	Page _____
Module Name Illustrative Problem (I)		Program Name _____	

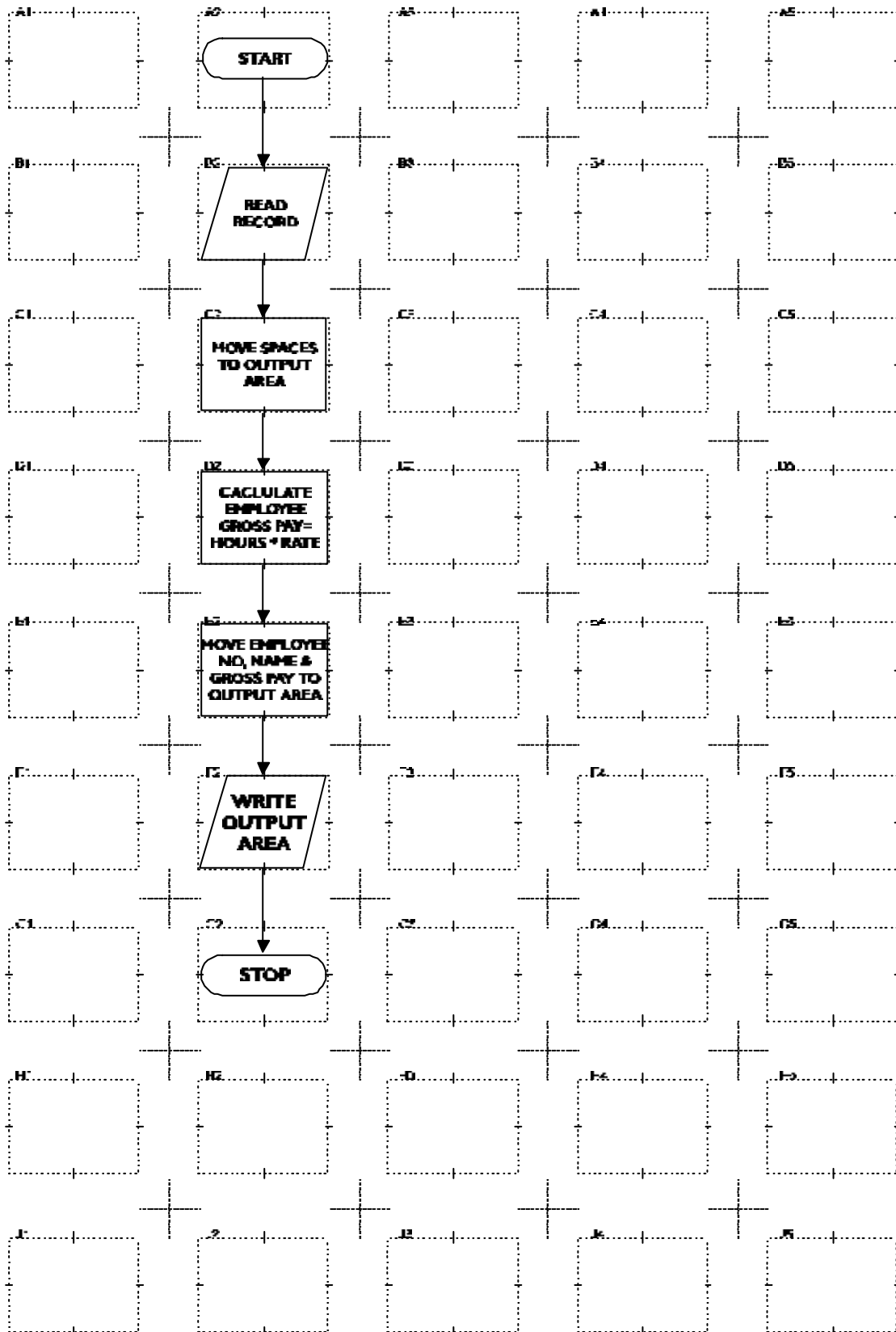


Figure 13

The START symbol (terminal symbol) serves to indicate the starting point of the flowchart. The START symbol is followed by an I/O (Input/output) symbol indicating that a record is to be read (ie input) from the file into the input area. The record contains the employee's number, name, hours worked and hourly rate of pay.

Once read the record can be processed. The processing required is specified by the 3 processing symbols. Firstly the output area is initialised and then the employee's gross pay is calculated. The asterisk used in the calculation of gross pay is the symbol used to indicate the multiplication. The universally accepted symbols and the operations they represent are:

+	addition	-	subtraction
*	multiplication	/	division

The final stage of processing is to move the employee number and name from the input area to the output area and the gross pay to the output area.

After processing, another I/O symbol is used, this time indicating that a record is to be written (ie output) from the output area to the printer. Note that output does not have to reflect input, the data output is as required on the printed report.

The terminal symbol containing STOP indicates the end of the flowchart.

3.2.2 Looping and Last Record Check

The problem looked at so far has involved the reading and output of only one set of data. In reality a program would have to be able to handle more than one record. Look at the flowchart shown in Figure 14. It is the same as the one in Figure 13 except that it contains a PROGRAM LOOP or UNCONDITIONAL BRANCH which transfers the Program logic back to the Read I/O symbol, after the first I/O symbol has been executed. The one branch has changed the program from one that deals with only one record to one that can deal with any number of records.

However, the Program loop, if used on its own, would throw up a severe problem i.e how do you stop the program once all the records have been read. The LAST RECORD CHECK concept is used. A DECISION symbol is inserted between the READ and PROCESSING symbols. By doing this you are checking if all records have been read. If there are still more records to be processed then the NO branch is followed, if the last record has been processed then the YES branch is followed. This is an example of a CONDITIONAL BRANCH.

The computer will only decide that end of file has occurred when attempting to read a record after the last one on file. There is NO indicator on the last record to say "I am the last one". Normally every read statement will be followed by a decision symbol asking "Is it E.O.F.?".

Flow Chart Worksheet

Programmer _____	Program No. _____	Date _____	Page _____
Module Name Illustrative Problem (1)		Program Name _____	
with Loop			

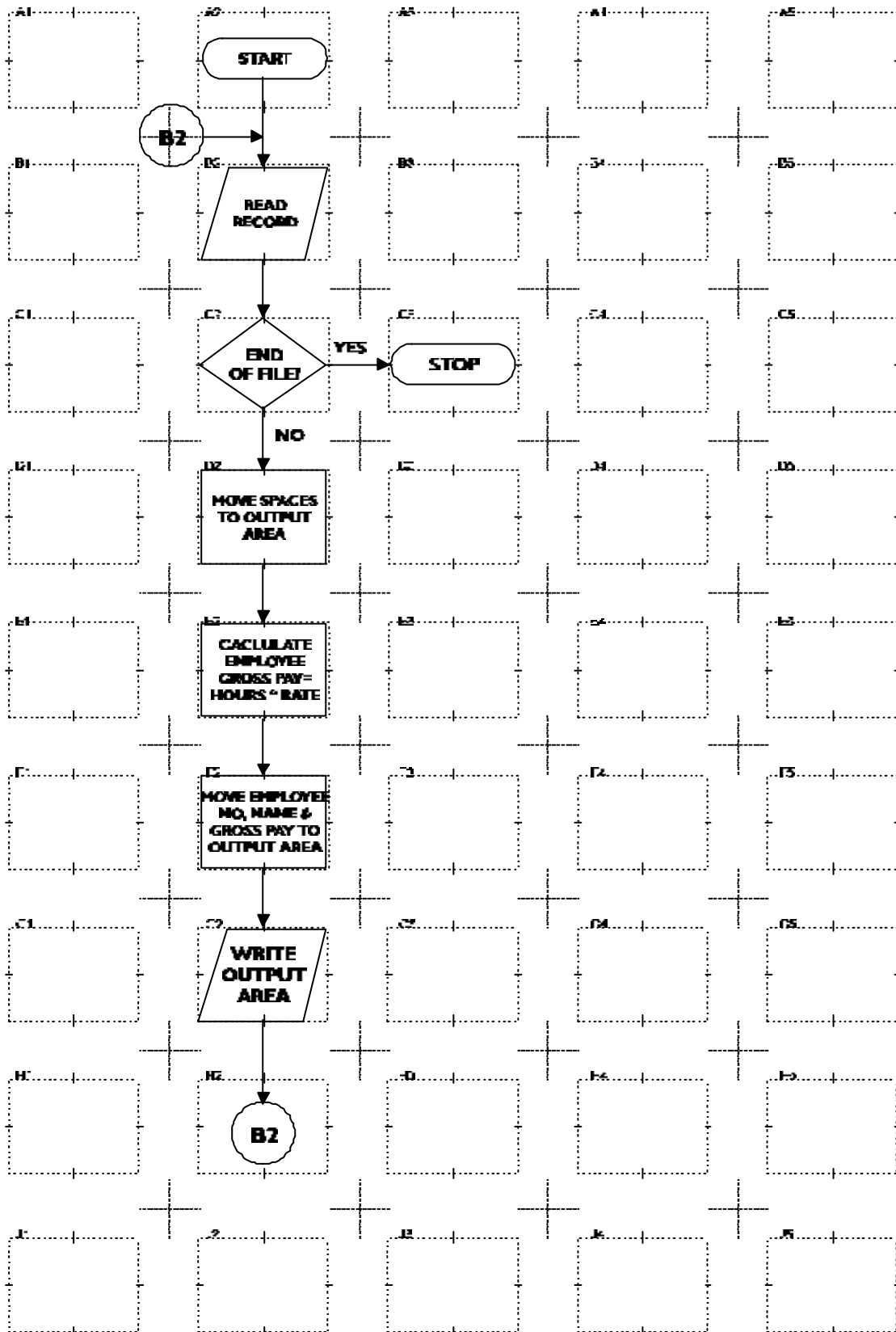


Figure 14

3.2.3 Student Problem 1

The Input data is the same as that used in Illustrative Problem (1). The processing involves the computation of the employee's Gross Pay, Tax and Net Pay.

The formulae used in processing are.

- (a) GROSS PAY = hours worked * rate of pay.
- (b) TAX PAYABLE = 24% of GROSS PAY.
- (c) NETPAY = GROSS PAY - TAX PAYABLE

The printed output is the employee's number, employee's name, GROSS PAY, TAX PAYABLE and NET PAY.

Construct a flowchart to show the above.

3.3 Headings, Switches, Accumulation of Total & Dry Running

In this section you will learn how to deal with Accumulation of totals; Headings; Use of Switches; and the principle of Dry Running

3.3.1 Accumulation of Totals

Many reports require the printing of totals at some point or another. Illustrative problem (2) demonstrates the logic used to generate a total.

3.3.2 Illustrative Problem 2

This problem involves the accumulation of a sales total for a salesman. The total is to be printed after all records for the salesman have been processed (ie. end of file). Input data consists of salesman number and name, item description and a sales amount. Output is to consist of those fields input, for each detail line, and the message "SALES TOTAL" along with the accumulated sales total on the total line.

For this problem a sales total accumulator (work field) will need to be set up in memory.

In the example flowchart (Figure 15) it can be seen that the second symbol indicates that the sales total accumulator is to be set to zero before any other processing takes place.

An accumulator (or counter) is merely a term used to refer to an area defined in memory within the program, in a similar way to that used for input and output areas. It must initially be set to zero to ensure that there is no current value in the accumulator, as this would result in the wrong total being produced.

The wrong total would be produced if the initial value of the accumulator was not zero because of the way in which accumulated totals are calculated, i.e. $\text{previously accumulated amount} + \text{this amount} = \text{new accumulated amount}$. When accumulating the first "amount" there is no previously accumulated amount.

Why then was the employee gross pay amount in Illustrative Problem 1 not required to be initially set to zero? The reason for this is that the employee gross pay was the result of multiplying hours * rate i.e. the result in no way depended upon the previous value of gross pay itself

So, whenever you use work fields for calculations consider whether they need to be initialised. Generally it is safe to say that if a field is used as an accumulator it must be initialised, if it is the result of ie. not part of a calculation, initialisation is unnecessary and in fact a waste of time.

The next two symbols in the flowchart (ie. I/O and decision symbols) read a record and test for end of file, as we are following the first record through the "NO" branch will be taken

Next the output area is initialised and then the sales amount from current record is added to the sales total. The required data is then moved to the output area and a detail line is written on the report.

Flow Chart Worksheet

Programmer _____ Program No. _____ Date _____ Page _____
 Module Name **Illustrative Problem (2)** Program Name _____

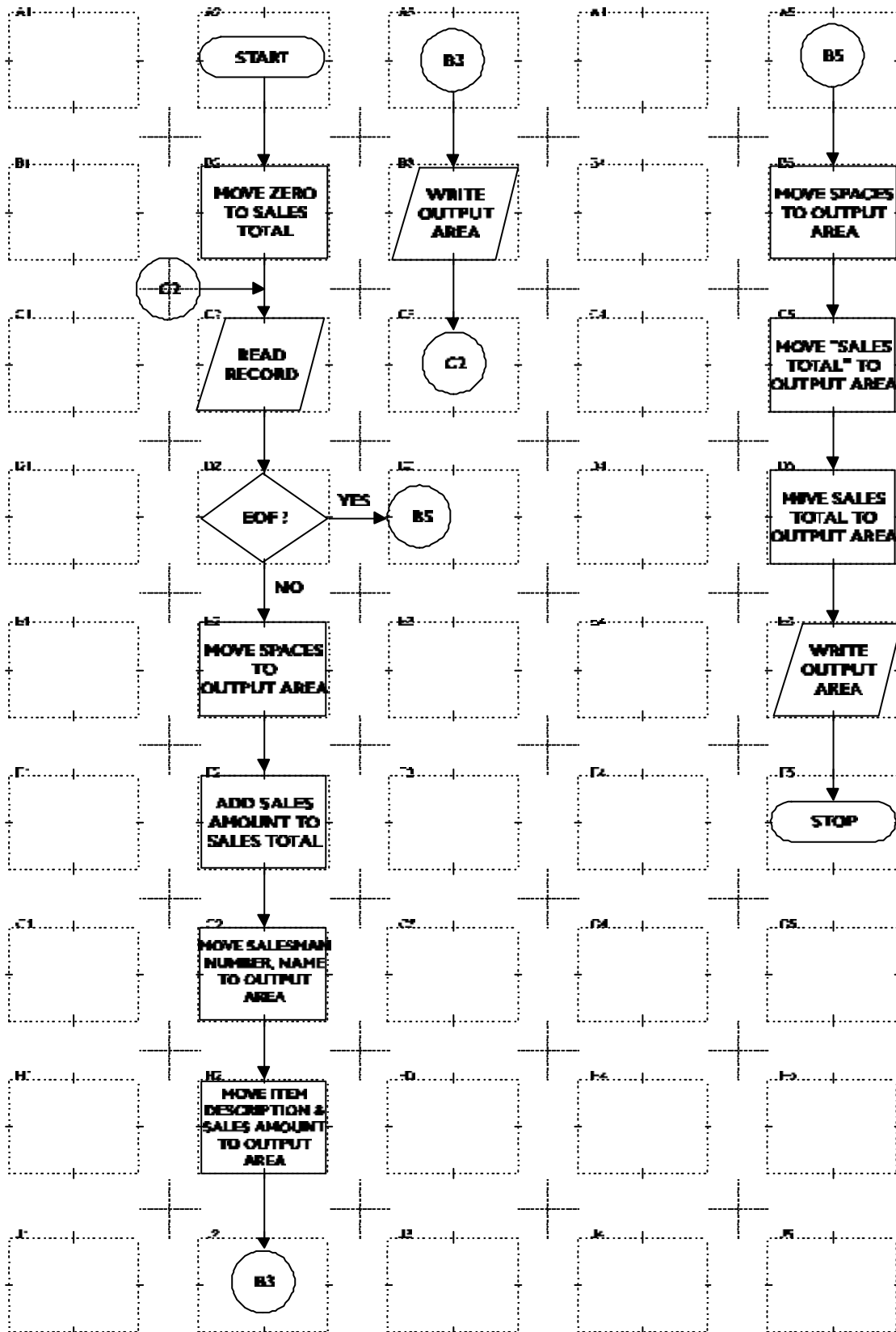


Figure 15

The unconditional branch sends the logic flow back to read the next record, This processing loop will continue until the end of file condition is satisfied at which point the "YES" branch will be taken from the decision symbol.

The next symbol indicates that the phrase "SALES TOTAL" be moved to the output area. To indicate the use of a phrase (literal) within a flowchart double quotes (") should be used around the literal.

The accumulated sales total amount is then moved to the output area, the total line is printed and processing is ended

3.3.3 Headings

Most printed reports will require headings at the top of each page, with page numbering. This must be catered for in the flowchart, with a method whereby, when a page full of data has been printed, new headings are printed at the top of the next page, before more detail data is printed.

If a page number is required as part of the heading, it will be necessary to setup a page counter in storage, which will be initialised to zero and incremented by one each time headings are printed.

It will also be necessary to keep a count of the number of lines written on a page, so that a new page can be started when necessary

It is usual to print about 50 detail lines (including spacing lines) per page, so as soon as the line count passes 50, the page count should be incremented, headings printed at the top of the next page and the line count set to zero.

3.3.4 Switches

A switch is a facility used in programming to decide whether certain operations should occur or not.

A switch, therefore, is effectively a YES/NO statement (or similarly O/1). It is used in conjunction with a decision symbol.

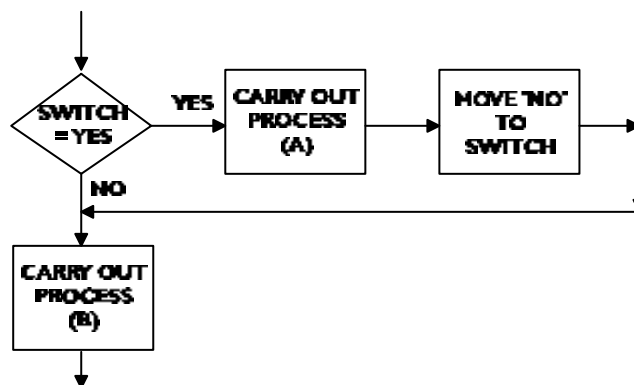


Figure 16

Switches have to be preset in order that, on processing of a certain record (eg. the first record in a file), the processing logic will follow the required path. The switch will then (usually) be changed to prevent that same progressing logic path from being followed every time a record is read. By changing the switch back again at a later stage, the flow of the program will again follow the original processing logic path.

The example (Figure 17) that follows shows switches can be used but is only part of a flowchart.

In the example the **FIRST RECORD SWITCH** is set to 'YES' in order to trigger the Headings after reading the First Record. The switch is then set to 'NO' so that the headings will not be printed after each record.

Flow Chart Worksheet

Programmer _____	Program No. _____	Date _____	Page _____
Module Name Example of SWITCHED flow		Program Name _____	

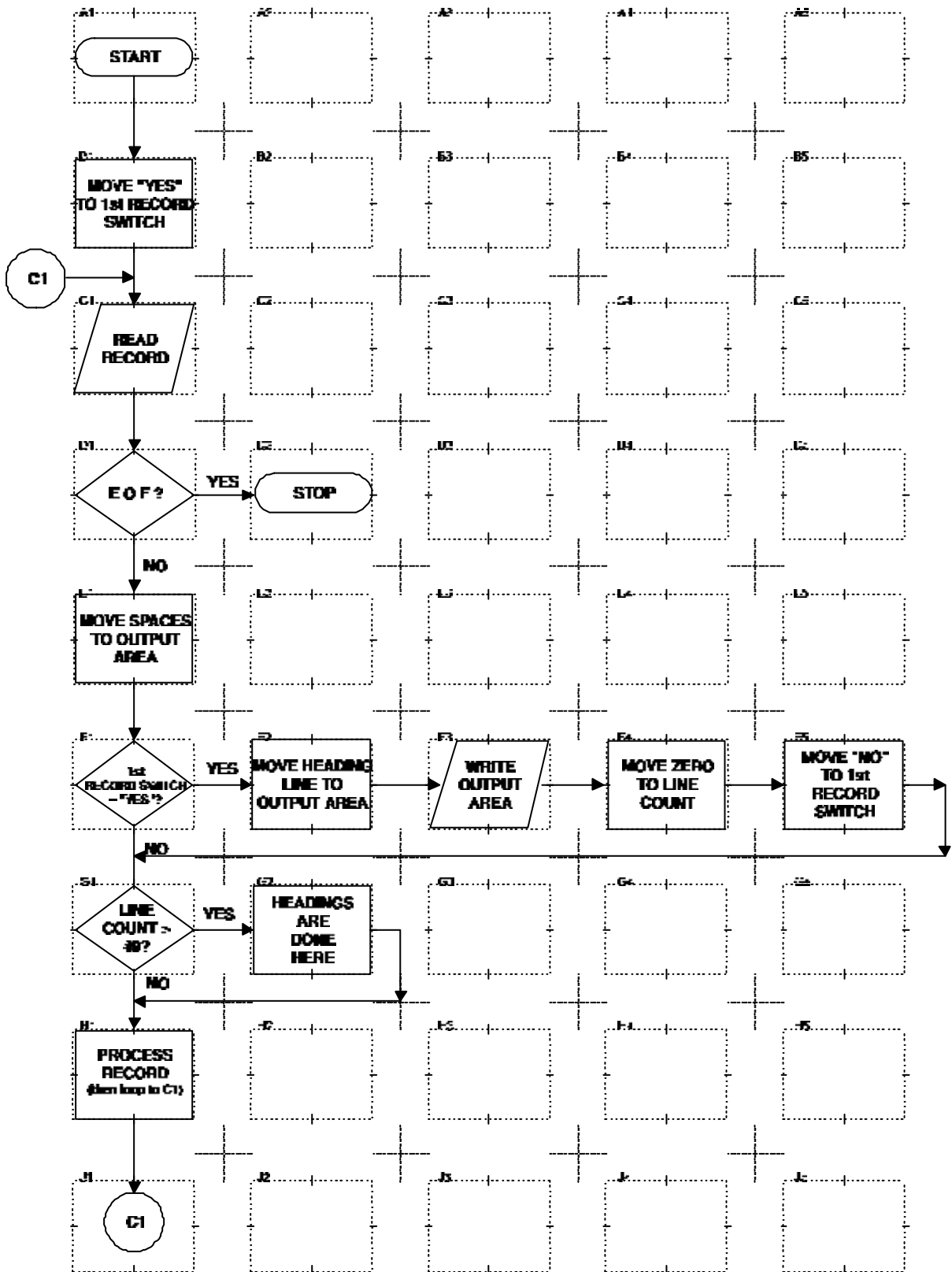


Figure 17

3.3.5 *Dry Running*

Dry running is a very important part of program design both at the flowchart stage and after coding. As previously mentioned dry running is the process of running through the flowchart, with a set of specially designed test data, to ensure that the expected results are produced ie. seeing if your solution to the problem works.

The first step involved with a dry run then, is to produce a set of test data. This test data must be designed in such a way that every possible path through the flowchart is frilly tested. So, as you can imagine, it can take many man hours to produce test data of sufficient quality, especially for a complex problem.

To design the test data you must again look at the original requirements of the problem and from this determine the input, the decisions and/or calculations carried out (if any) and the output. The test data is then created to suit these requirements eg several records that will take each branch from each decision should be included in the test data that contain all invalid fields as well as valid records and records containing individual invalid fields. Calculations should be supplied with the maximum and minimum values possible as well as the more usual values expected to occur, etc.

Having designed the test data the actual process of dry running can begin, which means playing computer ie, following each step through the flowchart exactly as instructed.

By following the test data through the flowchart and carrying out the specified instructions any logic errors within it will be found. These can then be corrected and the flowchart dry run again. This process will continue until such time as the dry run produces exactly those results expected from the test data.

When the expected results are achieved and the programmer is as sure as he can be that his logic is correct, the flowchart can be translated into the appropriate program code. Once coding is complete the dry run is repeated to ensure that the flowchart has been translated correctly i.e. the logic is still correct.

Unfortunately there is often a tendency for experienced and inexperienced programmers alike to assume that their logic will be right first time and they therefore do not bother with a dry run. This very often results in them having to spend time debugging (i.e. find the logic errors) and correcting a newly written program. It is far better to spend the time getting the problem correct at design stage than to have to "fix" the problem once the program is written.

In addition to dry running, once the problem has been coded, the code should be checked for syntax or semantic errors, this process is known as desk checking. So, at the coding stage before you compile and try to run your program, save it, desk check it and dry run it. Only if everything appears to be right should you compile your program. If compiler errors are shown, repeat the above procedure after correction.

It is not easy to dry run a problem, especially if it is one you have written yourself. There is a tendency to assume things, or to skip bits of the flowchart because you have been through that part 20 times before. However, you can bet that this time through the one decision that could have a different outcome, will do so and because you assumed it would not, you will miss the logic error that lies in that part of the flowchart.

So assume nothing, do exactly as the flowchart instructs you (just as the computer would) and you should find any logic errors that exist.

3.3.6 Student Problem 2

Input data: Salesman's name
Salesman's number
Product
Sales value

Note. The file contains the sales for 1 salesman only, although there will be any number of sales for this salesman.

Processing: Produce output containing one detail line for each record comprising Salesman's Name, Salesman's Number, Product and Sales Value. When all records have been processed a total line should be printed containing the heading "Salesman's Average" and the average value calculated as. total sales/no of products. The total line should be printed with a blank line between it and the previous detail line.

Construct a flowchart to show the above. Develop your own test data to "dry run" your solution.

There are two possible methods of indicating within a flowchart that a blank line is required to be printed. These are as follows:

1. To move spaces to the output area and then print the output area.
2. To specify the fact within the I/O symbol. i.e. "write print line after 1 blank line".

Report Layout

Salesman Average Sales

SALESMAN NAME	SALESMAN NUMBER	PRODUCT CODE	SALES VALUE
XXXXXXXXXX	99	99999	ZZZZZ9.99
XXXXXXXXXX	99	99999	ZZZZZ9.99
		AVERAGE	ZZZZZ9.99

3.3.7 Illustrative Problem 3

Report Layout

STUDENT MARKS

STU. NO.	STUDENT NAME	EXAM	MARKS
99	XXXXXXXXXXXX	XXXXX	99.9
		XXXXX	99.9
		XXXXX	99.9
		XXXXX	99.9

This output shows that the Name and Number is required to be printed only on the first record. By setting the 1st record switch initially to 'YES' the flow of the program is such that the Student Name and No is moved to output. The switch is then set to 'No.' Flow is then returned to the main logic of the program and the Exam and mark are also moved to output. Output is then printed. However, after every subsequent read, the Student Name and No will not be moved to output because the switch now reads 'No'.

By moving spaces to the Output before processing, it is ensured that the Student Name and Student No. fields in Output will be spaces unless something is moved into it.

This example shows a simple, but effective, use of switch.

Switches are an important tool available to the Programmer but they should be used sparingly. Later you will learn about other, and often more efficient ways of causing specialist operations to be performed.

Note also that page headings have been triggered in the first loop by initialising the line count to 50 rather than zero. The test in the main loop "is line count > 49" will ensure that headings are printed. This provides an excellent alternative to the first record switch which may be needed for other reasons.

Flow Chart Worksheet

Programmer _____	Program No. _____	Date _____	Page _____
Module Name Illustrative Problem (3)		Program Name _____	

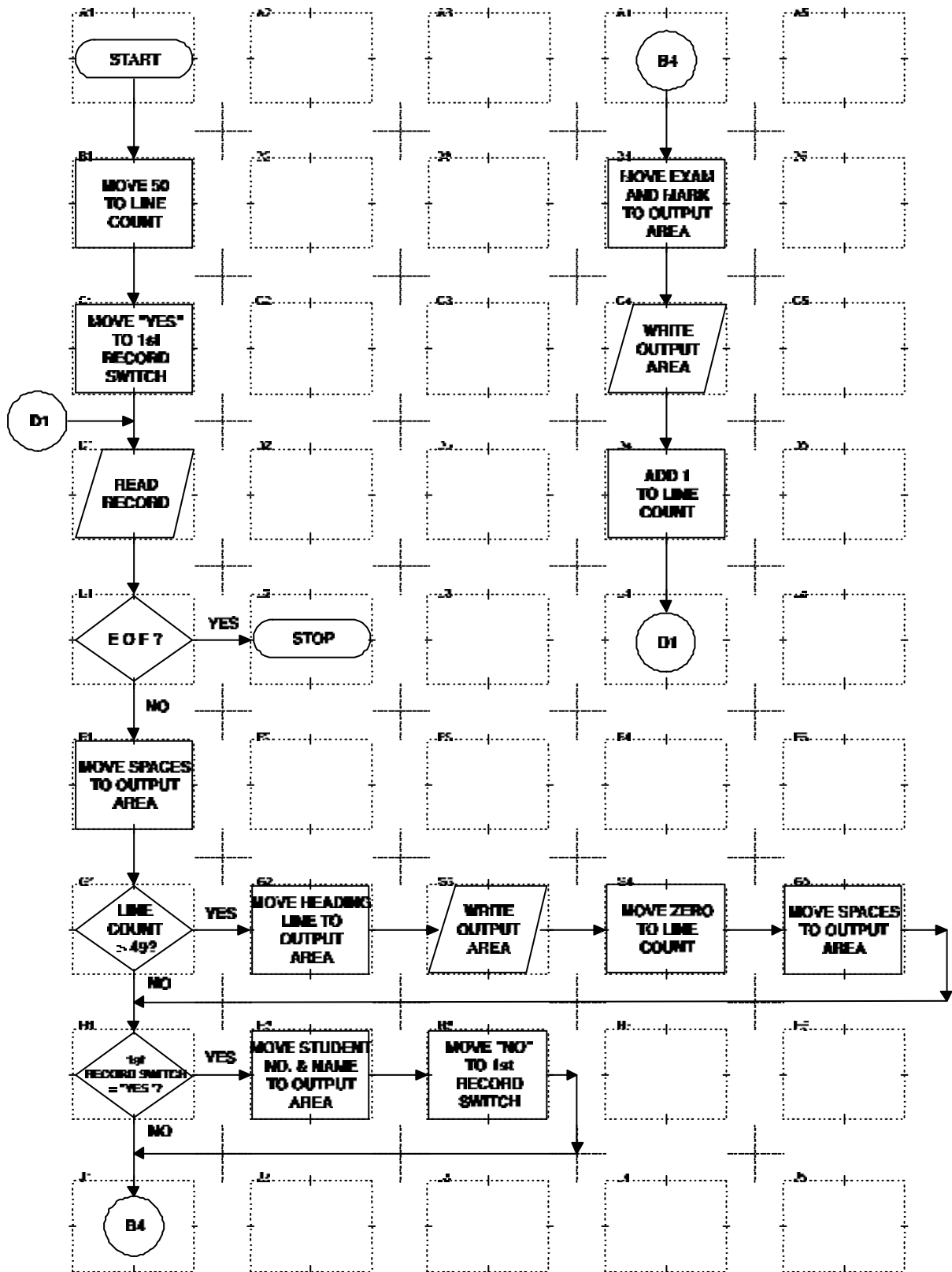


Figure 18

3.4 Control Breaks, Sub-routines, Initial Record Read & Compare Areas

This section shows other methods of controlling program flow and how comparison checks can be made. As you work through the flowcharts you will discover how to use: Control Breaks; Control Codes; Subroutines; Initial Record Read and Compare Areas.

3.4.1 Control Breaks

Processing can take alternative paths dependent upon the relative values in two fields in main storage. One of the more common types of comparisons which are made to determine the path of processing concerns comparing the same field in each input record which is read, and executing a unique set of instructions when the value of the fields changes. This is the concept of Control Breaks.

3.4.2 Control Codes

Very often an application calls for input to be identified by special control codes within the record, these control codes indicating specific types of processing to take place depending upon their value.

For example, a set of employee salary records may contain a bonus code in the twentieth position of each record. When this code = 1 the employee is to receive a £10 bonus payment, when it = 2 the employee is to receive £20, and when it = 3 he is to receive 5% of his gross salary as a bonus payment.

3.4.3 Student Problem 3

Construct a flowchart to produce a customer account report. Input records will contain an account code indicating whether the customer balance is current, over 20 or over 80 days old. A "1" indicates a current account, "2" indicates an account 20-80 days old and a "3" indicates an account over 80 days old. For current accounts no interest is given, for 20-80 day accounts 3% interest is given and 10% interest is given on accounts over 80 days.

Input Data

Field Name	Position	Type	Description
Customer Number	1-4	9	
Customer Name	5-20	x	
Customer Balance	21-26	9	2 decimal places
Account Code	27-27	9	1,2 or 3 only.

Output Required

Output is to consist of the customer number, name and balance, the interest percentage and the balance with interest applied (obtained by multiplying the customer balance by the appropriate percentage). Each customer is to be identified as being "current", "20-80 Days" or "Over 80 Days"

Report Layout

Customer Account List						Page zz9
(Customer Number)	(Customer Name)	(Balance)	(Interest Rate)	(New Balance)		
9999	xxxxxxxxxxxx	zzz9.99		zzz9 99	Current	
9999	xxxxxxxxxxxx	zzz9.99	3%	zzzz9.99	20-80 Days	
9999	xxxxxxxxxxxx	zzz9.99	10%	zzzz9.99	Over 80 Days	

3.4.4 Subroutines

A subroutine is a series of instructions which may be used more than once in the program logic. The subroutine logic will be coded only once, outside the main logic flow, When the subroutine is required, main logic branches to the start of the subroutine, executes it, then returns to the instruction following the branch to the subroutine. Subroutines can have ONLY ONE ENTRY POINT AND EXIT POINT.

In flowcharting a subroutine itself is flowcharted with the oval symbol (previously used for START and STOP) being used to indicate the beginning of the subroutine. It should contain the name of the subroutine. The last symbol should contain RETURN indicating that control passes back to the instruction following the branch to the subroutine.

In illustrative problem 4, three subroutines are used:

```
PRINT HEADINGS  
PROCESS AND PRINT STUDENT DETAILS  
CALCULATE AND PRINT STUDENT AVERAGE.
```

3.4.5 Initial Record Read

This is an alternative to the FIRST RECORD SWITCH which was studied earlier and deals with some problem, without using a switch.

The concept is to provide some unique processing for the first record, so that it will not cause a control break but pass straight into detail processing.

In Illustrative Problem 4 (Figures 19 to 22), it has been assumed that the input file will contain at least one record, so no check for EOF is made on the first read. Normally, in business applications provision should be made for dealing with possible EOF in every read.

Headings are printed and the student name and number are moved to the output area, ready for printing with the first set of marks, After this initial processing, control can be passed to the main “process and read” loop, which will continue to be executed until EOF is reached. At EOF, the students average must be calculated and printed before processing is complete.

This method of catering for any special processing associated with the first record in a file is very much more efficient than using a "first record" switch. The reason being that the initial process is only ever done once, whereas the "first record" switch has to be tested for being equal to "Yes" for every record in the file even though it will only ever be true once.

3.4.6 Illustrative Problem 4

This problem is identical to Illustrative Problem 3 except that the flowchart has been drawn to show some of the techniques that you have just covered.

Flow Chart Worksheet

Programmer: _____	Program No. _____	Date _____	Page <u>1</u>
Module Name: Main Loop	Program Name: Illustrative Problem (4)		

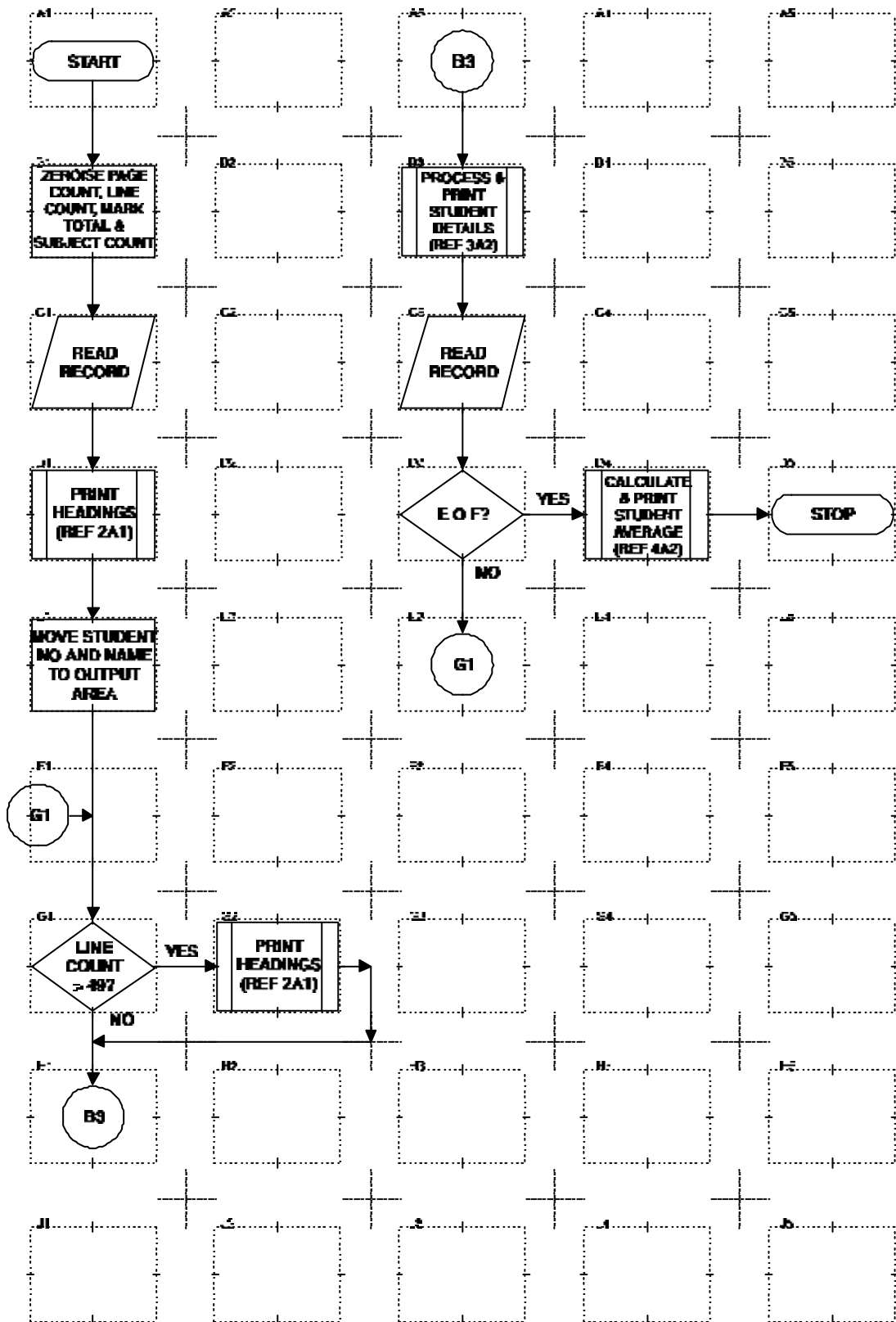


Figure 19

Flow Chart Worksheet

Programmer _____	Program No. _____	Date _____	Page <u>2</u>
Module Name Print Headings	Program Name Illustrative Problem (4)		

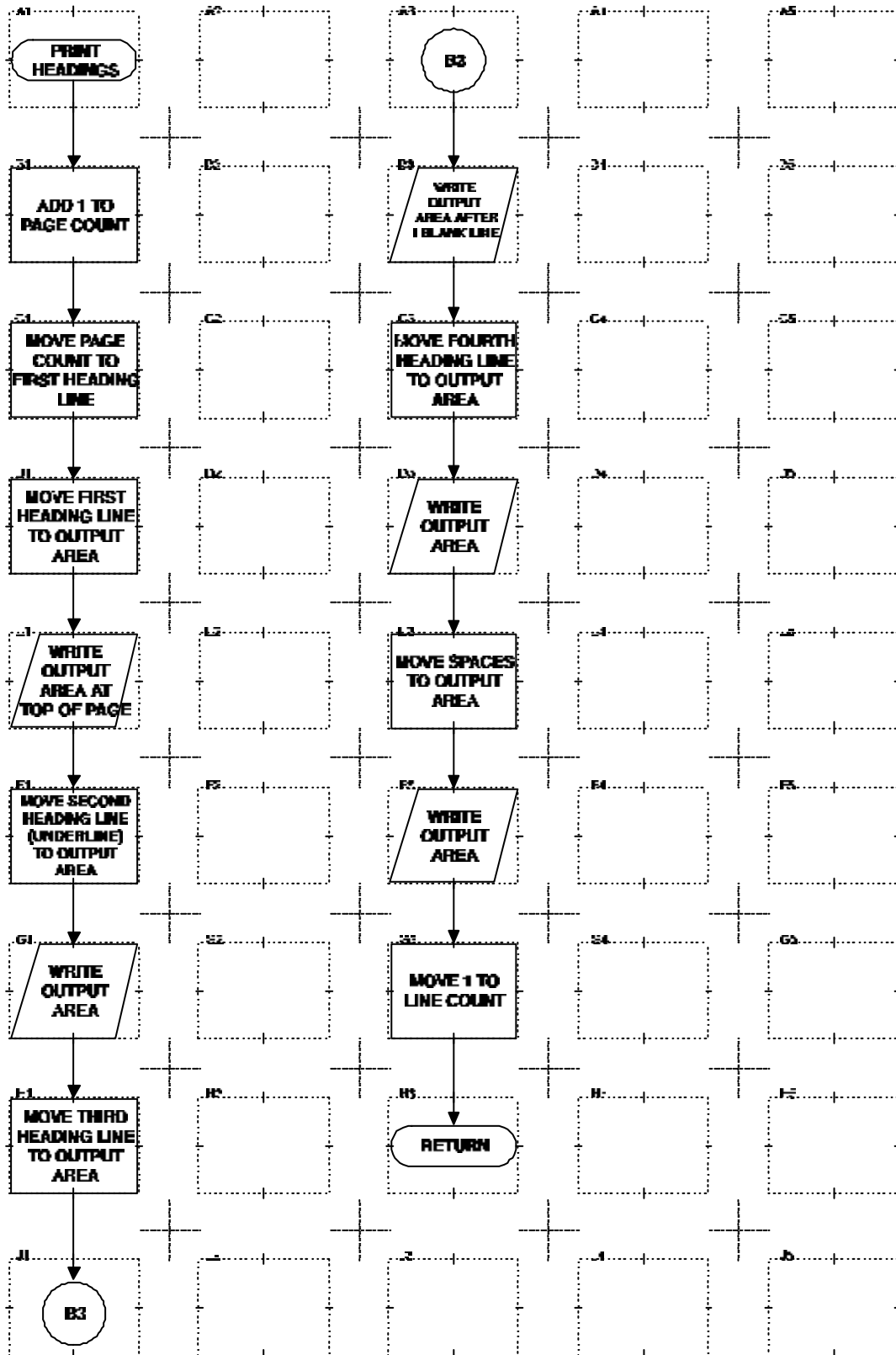


Figure 20

Flow Chart Worksheet

Programmer _____	Program No. _____	Date _____	Page <u>3</u>
Module Name Process & Print Student Details	Program Name Illustrative Problem (4)		

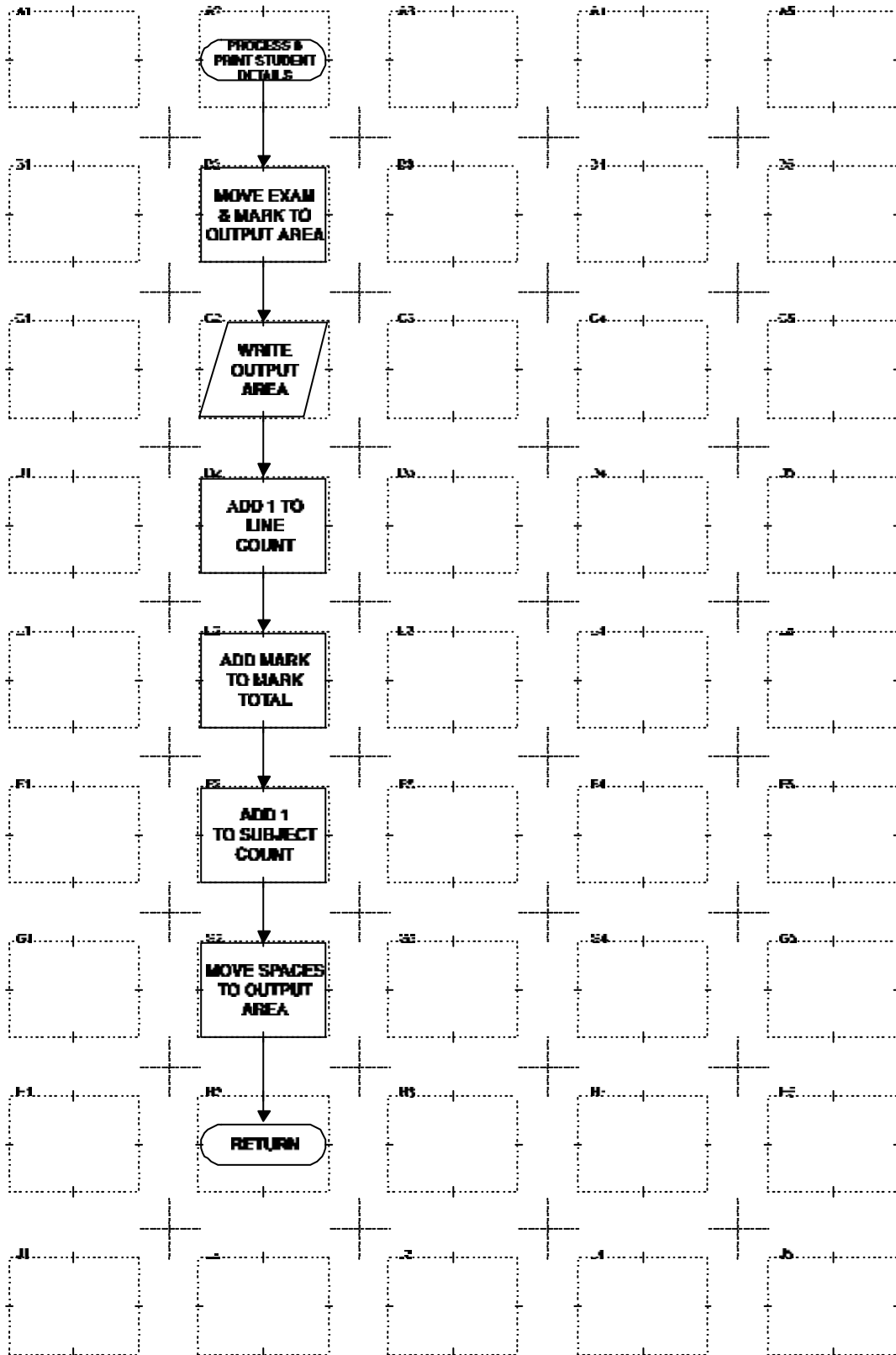


Figure 21

Flow Chart Worksheet

Programmer _____	Program No. _____	Date _____	Page <u>4</u>
Module Name <u>Calculate & Print Student Average</u>	Program Name <u>Illustrative Problem (4)</u>		

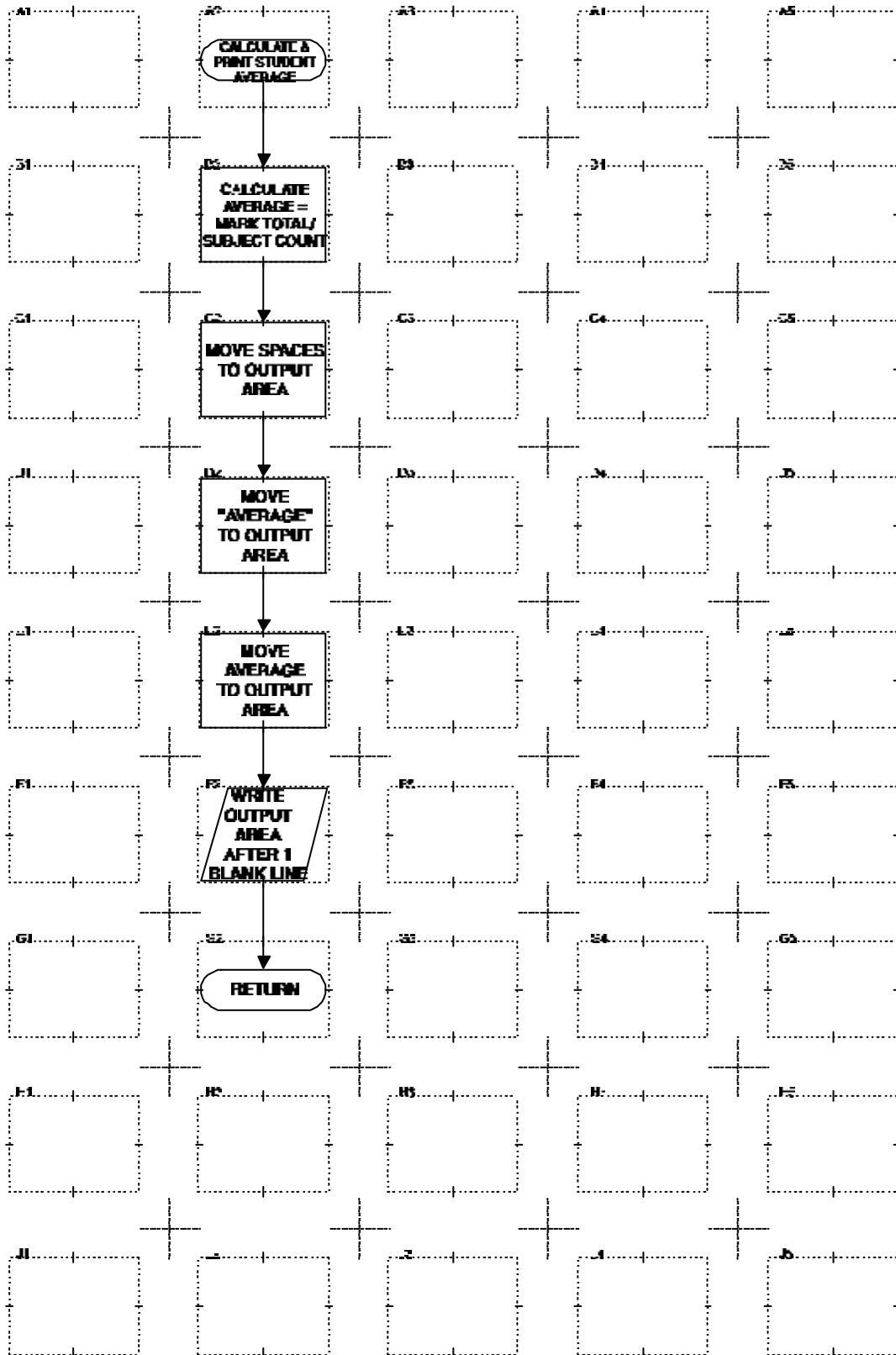


Figure 22

3.4.7 Student Problem 4

Construct a flowchart to read a file of customer names and purchases. Print a report as shown below listing the purchases by customer. Do not print the customer number and name if the customer number is equal to the previous customer number. If the VAT code is 0 then the rate of VAT is 0%, if it is 1 then the rate is 17.5%. The overall total purchases are to be printed at the end of the report.

Input record:

Customer number	999999
Customer name	xxxxxxxxxxxxxxxxxxxxxxxxxxxx
Purchase code	99
Purchase value	99999.99
VAT code	9

Output record - Report Layout

CUSTOMER PURCHASES					Page zz9
CUSTOMER NUMBER	CUSTOMER NAME	PURCHASE CODE	PURCHASE VALUE	VAT RATE	VAT AMOUNT
999999	xxxxxxxxxx	99	99999.99	z9.9	99999.99
		99	99999.99	z9.9	99999.99
999999	xxxxxxxxxx	99	99999.99	z9.9	99999.99
		99	99999.99	z9.9	99999.99
		99	99999.99	z9.9	99999.99
TOTAL PURCHASES			99999.99		99999.99

	T	E	E	E
E			<pre> C B C C D </pre>	<pre> r ; ; </pre>
N			<pre> F A C B F I A C C D </pre>	<pre> A =) ; e ; </pre>
N			<pre> C E L D I B </pre>	<pre> { e ; ; </pre>

