Week Four Lecture Note

INTRODUCTION TO PROGRAMMING & FLOWCHARTS

PRINCIPLE OF GOOD PROGRAMING

a. The program requirements must be specified in full and in writing. These specifications will be prepared by a systems analyst. A programmer has the task of converting these specifications into a written program.
b. In developing a program, a programmer should keep working papers. He can refer back to these papers later to check what he has done in case:
    (i) there is an error in the program for correction;
    (ii) the user of the program asks for a change in the program – e.g. for an extra bit of processing on input data, perhaps to produce an additional report.
c. The working papers might include a decision table or flowchart (or both).
d. When writing a program, the programmer should try to keep it as short as possible, since this will make more efficient use of storage capacity in the CPU. The program should therefore be logically well- structured
e. Program should be tested when they have been written. A programmer should prepared test data and establish whether the program will process the data according to the specifications given by the system analyst.
f. Provision should be made for program amendments. One way of doing this is to leave space in the program instruction numbering sequence for new instruction to be inserted later. For example, instructions might be numbered initially 10, 20, 30, 40, e.t.c instead of 1,2,3,4.
g. A record should be kept of all program errors that are found during 'live' processing of data, and the corrections that are made to the program.
h. Each version of a program should be separated identified to avoid a mix-up about what version of a program should be used.

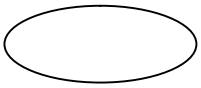
FLOWCHARTS

A flowchart or flow diagram is a diagrammatic representation of the sequence of processing operation. The operations to perform arithmetic computation or make comparison are shown

inside boxes in the flowchart and the sequence of operations is indicated by lines joining up the boxes. The start 'box' in a program flowchart will therefore be joined by an unbroken link or network of lines.

FLOWCHARTING SYMBOLS

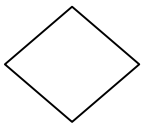
The following symbols are those recommended by the National Computing Center in their standards on systems documentation. Each one represents a distinct concept and has been selected in accordance with a British standard.



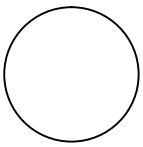
Start or end symbol. A program flowchart has one start and one end symbol.

Operations symbol e.g. arithmetic calculations, e.t.c

A routine of several program instructions not just one. E.g. Subroutine instructions to open or close files.



Decision or Condition Symbol. This must have two (or occasionally three) flow out of it.



Connector. This is a symbol to show when one part of a flowchart connects to another without drawing a connection line to show the flow. These are used when a flowchart goes off the end of the page and is continued on the next page.

You might notice that the unconditional branch instruction makes the flowchart loop round and back to read a record. This will enable the same program instruction to be applied to every record in turn until the record are fully processed. The 'loop' will eventually be broken by a different condition in an Input record.

The general principles you should always bear in mind are as follows:

(i) Make the flowchart clear, neat and easy to follow, so that it has a good visual impact and is an effective mans of communication. You are attempting to communicate to the examiner.

(ii) Ensure that the flowchart has a start and finish.

(iii) Avoid crossing flow lines wherever possible.

(iv) Make comparison instructions simple, i.e. capable of Yes/No answers.

USES OF FLOWCHART

Programmers use flowchart for a number of purposes.

a. To clarify the logic of a problem
b. To analyse the actions resulting from a set of conditions.
c. To sort out the procedural steps in the program.
d. As aids to program construction and coding.
e. As communicating documents –To explain the program to other programmers and the system analyst.
f. Check that the flowchart is logically correct and complete. This can be achieved by passing simple test data through the flowchart.
g. Try to make the flowchart consistence in the level of detail illustrated. If a part of the flowchart has to be shown in greater detail, this could be drawn on a separate sheet and referenced.

However,

(a) Flowchart may be difficult to construct when the logic is complex.

(b) They may become large and difficult to follow or trace back through, mixed levels of detail.

In such circumstances programmers may use decision tables and/or structures programming techniques as aids to preparing the source program.

Advantage of Flowchart

(a) They are an aid to problem definition and writing program. They help to simplify the logic of a program or process.

(b) They are more complete than decision tables for example they include start and end routines and illustrate program loops

(c) They can be used to test whether a program logic works, by running through the flowchart with 'mock data' to see if it will be processed as intended.

(d) They can be included in the specification of the program working papers, so that if there is any requirement in the future to check or amend the program the flowchart will indicate to other programmers the logic of processing as seen by the program written.

Disadvantages of Flowchart

(a) Complicated processing might required flowcharts that stretch on to several pages.

(b) They are not easy to amend. Alterations might involve a complete re-drawing of the flowchart, which could be a time consuming task.

(c) They tend to produce a badly structured program design. The logic shown in flowchart is rarely the best or most efficient logic for writing a program, and if a programmer uses flowcharting, his program is likely to be unnecessarily long. Nowadays, professional programmers do not use flowcharting for this reason.

STRUCTURED PSEUDO CODES

This is a design tool that describes the program tasks in a highly detailed narrative form. This method use English as the language but severely limits the available vocabulary and tries to follow the layout and logical operation of a computer program. Because structured pseudo codes appear to be fairly literal translations of programs, they closely resemble the finished product.

Structure Pseudo codes has the following features:

(a) It is more like spoken English than normal programming language-COBOL, BASIC and so is easily for programmers and non-program to understand.
(b) It is much more limited than normal speech, as it has to follow a strict logical order.
(c) There is a variety of conventions for writing it.

Structure pseudo code uses keywords (e.g. IF, COMPUTE) which by some conventions, are written in capital and have a precise logical meaning in the context of the narrative.

There are three basic logical structure.

- SEQUENCING
- SELECTION
- REPETITION

SEQUENCING INSTRUCTION

Example: MULTIPLY hours worked by Pay rate to get Gross Pay

Computation of Gross pay in a program.

GET Master record

Retrieving Master records from a file for the employee reference.

ADD 1 To Counter	Counting records.

SELECTION INSTRUCTION

Most programs offer a number of 'choices' where the consequent action depends on the choices being made in structured English. A decision follows this structured.

Example, a coy offers discount to trade customers only. How would this be expressed in structured pseudo code.

```
IF
        THEN
ELSE
```

So

```
IF the Customer is a trade Customer
THEN give10%
ELSE (Customer is not trade customer)
```

SO no discount given.

Sometimes, decisions are more complicated. Assume that the Company only offers a 10% discount to trade Customers who have been customer for over one year, but other trade customers receive a 5% discount

```
IF the Customer is a trade Customer
  IF Customer is over 1 year
        THEN 10% discount given
  ELSE 5% discount given
Else (Customer not a trade Customer)
  SO no discount given
```

One particular type of decision is a CASE statement which is an alternative to the IF-THEN-ELSE-SO structure outline above, which is satisfactory for making relatively simple decisions, but can become unwisely when the decision becomes complex. For example, we could have expressed the trade credit policy as follows. ENDIF ends the CASE statement.

```
IF Customer a trade Customer
    CASE Customer for more than one year
        Give 10% discount
    CASE Customer for less than one year
        Give 5% discount
ENDIF
```

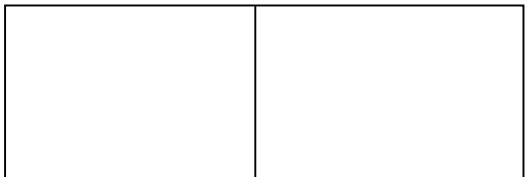
REPETITION

Sometimes a block or set of Instructions may need to be repeated until a final condition is reached. For example assume we have called a given block the name Block 1. We wish that this instruction to be executed until the number of processed records reached 100. This requirement is a condition, which we can call condition 1. In structured pseudo code, this can be written as follows.

```
REPEAT
    Block 1
UNTIL
    Condition
```

DECISION TABLES

Decision tables are used as a method of defining the logic of a process (i.e. processing operations required) in a compact manner. They are more convenient to use than decision trees in situations where a large number of four quadrants divided by intersecting double lines.



CONDITION STUB	CONDITION ENTRY	THE CONDITION STUB is to specify the values of the data that we wish to test for.
ACTION STUB	ACTION ENTRY	THE CONDITION ENTRY specifies what those values might be.

Between them, the condition stub and condition entry show what values an item of data might have that the program should test for. Establishing conditions will be done within a program by means of comparison checks. The condition entry quadrants are divided into columns, each column representing a district 'rule' or unique result of all the conditions.

ACTION STUB contains a list of the possible actions that could apply for any given combination of conditions.

ACTION ENTRY shows the actions to be taken for each combination of conditions

For example: Consider a banking lending policy which says, if an old Customer that has a saving above ₦ 50,000, approve a loan but if a new customer that has a saving less than ₦ 50,000 a loan should not be granted else refer to the manager.

Solution

RULE	1	2	3	4
Old Customer	Y	Y	N	N
Saving> ₦50,000	Y	N	Y	N
Approve	X			
Refer to Manager		X	X	
Reject				X

When constructing a decision table from a narrative, you must approach the position methodically. Work through these steps, applying them to the decision table in the example.

(a) List the conditions in the conditions in the condition stub
(b) List the actions in the Action Stub.
(c) Calculate the number of rules. This will be 2^n, where n equals the number conditions; draw up the required number of columns. If the conditions are not independent of each other, the number of columns can be less than 2^n
(d) Apply the halving rule:
    (i) Write in Y for half the rules and N for the other half, as answers to condition 1.
    (ii) For condition 2, write in Ys and Ns alternatively the number in each group being half the number of each group in condition 1
    (iii) Continue halving for each condition.
(e) Working down for each rule, enter 'X' against ach appropriate action.

The advantages of decision table

(a) It is possible to check that all combinations have been considered
(b) They show a cause and effect relationship
(c) It is easy to trace from action to condition (unlike flowchart)
(d) They are easy to understand and copy as they use a standardized format.
(e) Alternatives can be grouped to facilitate analysis.