PROGRAMMING

TUTORS

These notes are designed to assist teachers of the course and are in a condensed format. Teachers should also consult the syllabus for this module and adapt these notes accordingly by using extra examples and by filling out this material with detail. Students will be expected to apply the material in a BUSINESS environment and with reference to the particular situation specified in the examination question. Repeating these or any other notes in a generalised form is unlikely to satisfy the required answers for any question.

PRACTICAL REQUIREMENTS

Students are expected to have worked in a high level procedural language. Centres can choose the language but it is essential that it offers all the features listed within this section. These notes will not cover the language teaching elements because this is best achieved using a suitable manual or textbook devoted to that language.

PROGRAMMING METHOD

- Candidates must be familiar a formal algorithmic method of representing logic. Ideally, PSEUDO-CODE is suited for learner programmers because of its compactness. Other methods such Flowcharting, Jackson Charts … are acceptable. Whichever method is used, it must have a standard format.
- Suggested layouts for Pseudo-code (the examples are only intended to illustrate each type and do not represent a solution to a particular problem.

Input (ProductID, QuantityOrdered)	Data input at keyboard
ItemCost QuantityOrdered x UnitPrice	Calculation
Output (GrossTotal, VAT, NettTotal)… 3lines, 2dp	Output (with editing notes)
If ItemCost > 100	Decision – note indentations
then DiscountRate 5	
else DiscountRate 0	
EndIf	
For Count 1 to 12	Loop – Fixed number of repeats
process for each month	Indentation of the repeated process
EndFor	
Repeat	Loop – Variable repeats BUT
process next ordered item	at least once
Until ProductID = 0	
While StockLevel > 0	Loop – Variable repeats BUT
process this order	may not enter first time
EndWhile	
Open File (filename / Input or Output)	Open a file
Close File (name or ALL)	Close 1 or all files
Read (filename, record)	Read record from particular file
Write (filename, record)	Write to a file.
recordname.fieldname can used for individual fields of the record	
T (row,column)	2-D Array access. row/column could vary within a double FOR loop.

The use of indentation as shown above is essential for easy visualisation and to assist during development.

- Top Down = Breaking problems into major parts, which are then subdivided into more detailed logic. Enables modules can be tested separately and errors pinpointed to precise locations. Modules can be written by different people and reused.

- Structure Programming is the step-wise refinement/top-down approach breaking the problem into blocks called CONSTRUCTS. There are 3 main structures. Construct = routine with ONE-entry and ONE-exit.
  Sequence = construct grouping the steps of the tasks one after the other.
  Iteration = construct repeating a series of tasks until an exit condition is met.
  Selection = construct performing different tasks according to a condition.
  Note that Iteration and Selection Constructs must still have one entry/one exit. It is not permitted to exit the construct from within a loop.

REQUIRED PROFICIENCES (by students)

- Defining variables and assigning appropriate data types.
- Input of data from a keyboard including a suitable prompting message.
- Outputting data to screen and printer with good layouts. This includes tabulating data so that decimal points align, output of numeric data to a specified number of decimal places.
- Manipulating string data including extracting parts of a string using appropriate string functions.
- Choosing the most appropriate data structure – strings, arrays, records.
- Applying decision instructions (IF, CASE) with multiple actions in any of the resulting action sections.
- Recognising the need for a loop and deciding which to use. Generally this means the ability to use:
  i. **FOR/ENDFOR** to control the repeated process a fixed number of times. Defining the loop control count variable and incrementing it appropriately by +1 , -1 or other intervals.
  ii. **REPEAT/UNTIL** for a variable number of repetitions which will always occur AT LEAST ONCE. Exit from the loop is at the end.
  iii. **WHILE/ENDWHILE** for a variable number of repetitions with the possibility that the process might not be executed even once. Entry checked at the beginning.
- Applying loops in common standard situations:
  i. Looking at all elements of an array of all records in a serial/sequential file. e.g. valuation of all items held in the file.
  ii. Searching for a particular element in an array or serial/sequential file, stopping the search if found or reporting that the search has failed if the element has not been found when the end of the array is reached.
  iii. Search one array for a target value and if found, displaying a corresponding value from another array, which may hold data of a different type.
  iv. Using double loops, one embedded in another. e.g. invoicing a number of customers, each buying a number of different products.
- Structuring program by means of a top down approach, in particular using procedures.
- Writing and using procedures with input parameters.
- Using standard functions as listed in the syllabus, including string functions.
- Writing and using own functions to perform simple tasks. e.g. accepting three integers and returning the middle-valued number.
- Setting up menus, accepting the user choice and executing the appropriate action.
- Validation of data for range or within a particular set.

TESTING

Candidates will be expected to write about the process of testing and the design of test data. They must be aware of the four elements of testing

1. Test Plan. This is the devising of an overall strategy about how to go about the testing. Consider the simple example of a program, controlled by a menu with the following menu options (there would be other options):
   1 Add new customer
   2 Delete customer
   3 Amend customer
   4 Access and display a customer's details

5 Display all customer balances
0 End Program
A test plan would probably be:

i. Check the menu system works – Options 1 to 5 and 0. The menu and all procedures need to be in place. Each procedure only displays an identifying message to show that it has been accessed, the screen held and then return to the menu. At this stage, NONE of the main procedures perform their required task but the structure is fully in place and the program will run for any option even if it does nothing useful.

ii. Options 1 and 5. New customers are added and proved to be on file by option 5 which displays some details for ALL customer.

iii. Option 3. Customer details changed (address/telephone credit rating..) and checked by option 5.

iv. Option 4. This could be combined with 3 so that a customer's details are displayed with option 4, altered with option 3 and then checked with option 4.

v. Option 2. Finally this option is written and checked with 4 or 5. It is pointless trying this earlier because the program needs a minimum amount of data on file to be checked and to start deleting it before other options are finished is unhelpful.

vi. Bulk data. This is additional data for no other purpose than to check that the program can handle more output than can appear on one screen or one page.

The decision when to test VALID and INVALIDATE situations varies with type of problem. It is usually beneficial to check the program works with valid data before starting to enter invalid situation.

2. Test Data. Test data needs to be carefully designed. Its purpose is not to check that the program works but that it does not fail. If a program is tested with minimum data, it may appear to work but because certain situations have not been checked, logic errors are not seen until that data situation appears later, possibly when the program is running live! Test data should check:

i. valid and invalid situation. Checking for wrong data type is more advanced and is not expected at this level. The question will state this.

ii. boundary conditions. e.g. if better discounts are offered when a customer spends £100 and again at £200, extensive test data is needed. Spending totals of £99.99, £100.00, £199.99, £200.00 should be tested. The first pair show not only that a better discount is earned at £100 but that it occurs at EXACTLY £100. Testing with £50, £150 and £250 WILL give different discounts but these figures do not prove that the changeover occurs exactly at £100 and £200.

iii. decimal values for real data. In designing test data, there needs to be some totals which are not exact round numbers to ensure the print layouts are correct, particularly with money values. However, this need only occur a minimum number of times. Data testing for other features can be simpler e.g. customer orders 2 items at £10 each.

iv. that all arithmetic processes are checked. Two examples where data may appear to give the right answers but the logic could be wrong are:

a. Multipliers. e.g. Ensure at least one customer orders two or more items of a given product otherwise a cost of 1 x 17.50 = 17.50 could have been achieved merely by using the price.

b. 2 x 2 = 2 + 2. Two items at £2 cost £4 but a program error with the "x" accidentally replaced by a "+" will not show this error. Instead check with 2 x 3 or other combinations.

v. that all possible repeating situation occurs: e.g. for a customer ordering system

a. There are at least two customers in the data

b. Some customers order only 1 product while others order several products.

Expected Results. This feature is often overlooked. Write out the expected results for the complete set of test data. All subsequent tests will be compared with these results. If the data is good enough to test the program rigorously, there should be no need to devise new test data later.

Bulk Data. Test data should never be excessive. There is no need to test a particular situation more than once. However, there is a need sometimes to produce more data simply to ensure that outputs run to more than one page- this may mean there are page totals which are then carried forward. This extra data can clearly be simple data (as in iii. above).

3. Testing. This is the actual testing. It clearly does not form a part of the examination but will be tested theoretically. Candidates should be aware of the different types of testing possible.
   These are:
   i. Dry Run. Candidates WILL be expected to perform a dry run on an algorithm with given data. This should be set out in table format with one column for each variable in the routine. Each row of the table should show the progress as data passes through the routine. Only when the value of a variable changes should there be an entry in the table. Leave blank spaces on each row for each variable which does NOT change.
   ii. Module testing
   iii. Program testing
   iv. Link/system tests where the file outputs from one program are input into another
   v. Live data testing
   vi. Acceptance testing – the user supplies data, monitors the outcomes and accepts the programs as they are or requires the programmers to correct some problem area.

IT DOES NOT WORK! A frustrating situation for any programmer is to obtain faulty outputs and the programmer, after much dry running and testing, has no idea where the error is. Candidates frequently suggest asking another programmer to check but this is ignoring the issue. A professional programmer should be able to deal with it. Possible solutions are:

- Using a **trace** program which will display the values of variables at different points in the run and hence give the programmer a better idea of where the problem is.
- Similarly, the programmer can insert his/her own **test bed routines**. These are small coding routines, perhaps a procedure, which can be inserted anywhere in the program to print out variable values at that point. These routines are often de-activated but left in programs for the benefit of subsequent maintenance programmers.

4. Test Log. A programmer should record details of every test undertaken. This will show:
   i date/time,
   ii which set of test data was used
   i. purpose of the test (perhaps the menu option being tested).
   ii. outcome – briefly describe what is successful and what is not
   iii. possible actions or causes of any problems – e.g. variable X should be validated to be 1 to 10.

Candidates undertaking the Programming Project sometimes give the impression that everything worked first time! We all know that did not happen. Admitting problems is a professional approach and indicates that the candidate knows how to proceed. Not admitting the problem is actually missing out part of the required project work and will inevitably earn fewer marks.

DEVELOPMENT PROGRAMMERS / MAINTENACE PROGRAMMERS

Candidates must know the roles of the programming team in a professional situation. Of course it varies depending on the size of the new application and the way the application is being implemented (database?).
A general process which would cover most situations where some parts could be omitted is:

1. Systems analyst gives a presentation to the whole programming team on the overall purpose of the new system and its common processes.
2. He then gives each programmer an individual program specification.
3. After reading the specification, the programmer may need a 1:1 meeting with the analyst to clarify certain elements.
4. Programmer designs test data and expected results. Sometimes the analyst provides this. Programmer and analyst discuss the testing plan. Designing test data early helps to identify the situations that can occur.

5. Programmer develops an algorithm.
6. Dry run test data through algorithm and compares with expected results.
7. As a result of the dry run, some changes may need to be made.
8. The program is coded into the language dictated by the analyst.
9. The program is compiled and syntax errors corrected.
10. The program is tested using the same test data and compared with expected results. A test log is maintained.
11. System and Acceptance testing with the user is undertaken, the programmer may be present.
12. Throughout, the programmer maintains documentation inclusion in the final documentation.
13. A user manual is produced. A technical manual is produced for the maintenance programmer.
14. After acceptance where all programs are completed and working, the programming remains on standby for a limited period to effect immediate corrections when the system is running live. Each programmer will probably now be also beginning for a new project.
15. When, problems seem to have been corrected, the team disbands. If the team consisted of 10 programmers, only a small number of these (perhaps even1) will then be designated as maintenance programmers for the whole system. This means a programmer will be expected to make corrections on some programs that he/she did not actually develop.
16. After reviews perhaps at 6 months/1 year, changes may be needed for business/legal reasons and the maintenance programmer will implement these. If the maintenance program leaves the company, then a replacement should be appointed immediately because of the need for somebody always to be knowledgeable and available at short notice in the event of an emergency – business must still keep going.

DOCUMENTATION

All elements of a business program must be retained as part of the documentation. The candidate needs to know which parts will be used by the maintenance programmer and which will be for the user.

User. Users are NOT programmers and are not expected nor should be allowed to make changes to programs. Hence, all elements of the logic should NOT be available to the user. This includes:

i Algorithm
ii Annotated listing of the code
iii Variables list
iv Original Test data.

The user needs documentation at a simple level to ensure that he/she understands what to do in every situation. This includes knowing what options the system offers, possible restrictions (e.g. salesmen have access only to certain elements of the system), how to run the program, how to install (possibly but this may be restricted more technical staff), possible problems and what to do when they arise.

Maintenance Programmer. He/she would have access to all the user documentation but would prefer more compact information. The four items above would be at his disposal with details of how the whole system links together.

ERRORS

1. **Syntax errors**. Violations of the language rules. e.g. Misspelling a key word such as PRINT. The compiler detects the error.
2. **Technical errors** during the compilation process. The compiler manual probably needs to be consulted.
3. **Logic errors**. Errors which produce the wrong results as shown up by comparing actual with expected results.
4. **Runtime errors**. Errors which stop the program completing. The computer is unable to continue. The simplest is "Divide by Zero". The programmer will not have written an instruction actually doing this but may have coded A/B and B might happen to be zero under certain conditions.

OTHER IMPORTANT FEATURES

- **Parameter** is a value passed to or from a routine indicating how that routine will perform.
  e.g. “A” or “D” indicating whether a sort is to be ascending or descending.
  Procedures/Functions use:
  i **Local** Variables. Their values are only valid within that routine OR in routines called by it.
  ii **Global** Variables. Their values are available throughout the whole program.
- **Automatic file dumps activated by program**. This is a security process to ensure that main files are saved at frequent time intervals, during a long run, along with the current state of certain memory registers. If the program fails before the next dump, the data can be recreated and accurate to the last dump. This overcomes the problem of lengthy file update/print run failing perhaps because of a printer failure. The program is effectively run from the point of the last dump ensuring that all printouts are produced and the file is correct.

- **Iterative Solutions**. Scientific/mathematical problems often do not have a known method of solution. An example is a complex equation. A repetitive method is used where an initial estimated solution is input and a better solution is produced at each phase until successive solutions are within the required accuracy. This is ideal for a computer which is able to repeat a process thousands of times per second and work to many digits of accuracy. The accuracy required is input. The problem is that in some cases, unless the initial estimate is close enough, instead of converging on an answer, the program diverges or converges onto an alternative solution which is not required.
  Candidates are not expected to know the theory of iterative processes but could be asked to program one where the details are given in the question.