

THE INSTITUTE OF COMMERCIAL MANAGEMENT

SUBJECT SYLLABUS



Programming

Unit code: PR-0908

Level: Certificate

Credits: 20

Unit leader: PY

Pre-requisites:

Main Aims of the Unit:

This unit will introduce the computer programming process and enable students to design and produce programs in a high-level language of their choice. The unit will stress the importance of planning the programming approach and provides a grounding for understanding how professional programmes are developed and implemented.

Main Topics of Study:

A. Overview

1. The difference between different types of languages from assembler, through high level languages to 4GLs and Object-oriented languages. An outline history of the development of languages over the years.
2. Differences between Procedural and Declarative languages.
3. Summary of the stages of the translation process - Compiler and Interpreter.
4. An appreciation of assembly language code through a limited set of operation codes. How these build into a program. Writing full programs is NOT required.

B. Program Development

1. Overview of the stages of program development from receiving a program specification to handing over a fully tested and documented program applied to a PROFESSIONAL programmer. Candidates should have an appreciation of the difference of scale of REAL programs compared with classroom exercises.
2. Initial understanding of the problem. Checking details with the systems analyst.
3. Create algorithms for real problems.
4. Comparison of different algorithmic methods - flowcharts, pseudo-code, decision tables, Jackson charts. Advantages and disadvantages of each.
5. Define Structured programming and Construct. The three main constructs – sequence, selection, iteration. Top down programming.
6. Modular programming. How modular program can be implemented. Advantages and disadvantages of their use.
7. Prototyping. Advantages and disadvantages of its use.

C. Input and Output of data

1. Data types and the operations that can be performed with each.
2. Keyboard entry of data. Handling and correcting invalid data.
3. Standard output layouts. Invoices. Account ledgers.
4. Design a layout for a given situation. This must be workable in the real world.
5. Differences between output on a screen and a printer.

D. Filing data

1. The need for files. The consequences of filing systems NOT being available.
2. File structure. Organisation and access methods. An appreciation that a file might be accessed for more than one purpose and so determining the method of organisation.
3. Programming for serial and sequential files. Commands relating to file handling – open, close, read, write, check end of file, append.
4. Programming to locate records from the whole file that satisfy a given condition.
5. Programming to accumulate values from each record in a file.
6. Programming to merge two similar files into one.
7. Sequential Master file update from an UNSORTED Transaction file. Candidates will not be expected to program for this.

E. Handling data in memory

1. Variables – rules for naming variables/procedures/functions.
2. Performing calculations.
3. Decisions – IF, CASE.
4. Loops. Difference between the three main looping features and WHEN to use them.
 - i FOR/ENDFOR for fixed number of repeats.
 - ii REPEAT/UNTIL for variable repeats – the process is always executed at least once – test for exit at the end.
 - iii WHILE/ENDWHILE for variable repeats – the process may not execute even the first time in rare situations – test for exit at the beginning. Use of data TERMINATORS.
5. Arrays – 1-dimensional and 2-dimensional.
 - i Read all data into an array
 - ii Output all data from an array
 - iii Search an array for a given value
 - iv Accumulate values held in some elements of an array
 - v Use one array to find data to access another.
6. Text data – extract character(s) from a string. Joining strings.

F. Procedures/Sub routines and Functions

1. Use of procedures for structuring and re-use processes. Library of procedures/functions.
2. Parameter passing. Calling a procedure. How parameters are handled internally.
3. Difference between procedures and functions.
4. Standard functions such as:
 - Numeric – SQRT, ABS, INT, RANDOM, SIN/COS/TAN
 - String – LEFT, RIGHT, ASC, VALUE. {The names may vary in different languages}

G. Testing

1. Plan the structure of test data for a program.
2. Design test data for valid and invalid situations.
3. Perform a dry run/desk check using test data and an algorithm – lay out the results in table format showing the value of variables ONLY as they change.
4. Programming to test invalid conditions AND subsequent repeated re-entry after data is rejected until it is finally accepted.
5. Testing by program. Recording the results in a test log. Levels of testing – module, program, system, user acceptance.
6. Methods of locating errors of logic – dry run, trace routines, test bed routines, temporary print commands inserted.
7. Difference between syntax and logic errors. How and when syntax and logic errors are detected and corrected.

H. Documentation

1. The need for documentation.
2. Documentation for the maintenance programmer.
3. Documentation for the user.
4. The reasons why user and maintenance programmer documentations are different.
5. Uses and benefits of comments/annotations in program listings.

Learning Outcomes for the Unit:

At the end of this Unit, students will be able to:

1. Develop, write and test high level language programs
2. Appreciate the need for low-level (Assembly) languages
3. Structure a computer program appropriately using looping, arrays and sub-routines
4. Use appropriate measures to thoroughly test programs
5. Provide effective User documentation to support programs

The numbers below show which of the above module learning outcomes are related to particular cognitive and key skills:

Knowledge & Understanding 1-5
Analysis 1, 3
Synthesis/Creativity 1
Evaluation 1, 5
Interactive & group Skills -
Self-appraisal/Reflection on Practice -
Planning and Management of Learning -
Problem Solving 1, 3, 4
Communication & Presentation -
Other skills (please specify) -

Learning and teaching methods/strategies used to enable the achievement of learning outcomes:

Learning takes place on a number of levels through lectures, class discussion including problem review and analysis. Formal lectures provide a foundation of information on which the student builds through directed learning and self managed learning outside of the class. Students will spend a great deal of time involved in practical exercises on the computer to develop programs. The students are actively encouraged to form study groups to discuss course material which fosters a greater depth learning experience.

Assessment methods weightings which enable students to demonstrate the learning outcomes of the Unit:

3 hour examination: 100%
The paper will be split into two parts:
Section A Compulsory section accounting for 40% of the paper. Short questions requiring high-level programming code with good syntax.
Section B Candidates choose 4 questions from 7, each question is worth 15% of the marks. These questions will include the design of FULL test data for a given situation, dry run of a given pseudo-code routine using given data, develop a pseudo-code algorithm for a given situation.

Indicative Reading for this Unit:

Main text

Refer to the ICM website for details of learning material available.

Appropriate manuals for the chosen programming language.

Guideline for Teaching and Learning Time (10 hours per credit)

Lectures / Seminars / Tutorials / Workshops: 50 hours
Tutorial support includes feedback on assignments and may vary by college according to local needs and wishes.

Directed learning: 50 hours
Advance reading and preparation / Class preparation / Background reading / Group study / Portfolio / Computer practical exercises

Self managed learning: 100 hours
Working through the course text and completing practical assignments as required will take up the bulk of the learning time. In addition students are expected to engage with the tutor and other students and to undertake further reading using the web and/or libraries.

Guidelines

- This module is primarily designed to test ability in procedural programming. It is similar to the previous "Computer Programming 1" module but with a few changes including an appreciation of Assembly programming.
- A good structured language is advised such as PASCAL. BASIC may be used but candidates will be expected to initialise variables even though most versions of BASIC perform this automatically. The reason for this is to encourage good programming practice.
- Prior to the examination, candidates should have designed, written and fully tested a series of programs involving input/output, different looping features, decision, arrays, files, functions and procedures. They should have drawn up algorithms for programs with at least double loops and be aware of the difference between different types of looping facilities available in a given language. In the examination, it is expected that language syntax will be correct.
- Candidates should be able to design test data to test a program fully. Dry running/desk checking a program with test data should be practised using a systematic tabular format. An appreciation that dry-running is simulating the actual running of the program by the computer.
- A clear understanding of different types of documentation is required and why different documentation is needed by different people and in different situations particularly in large commercial applications where the documentation may be extensive. Some previous work submitted for printout design has been poor because it was totally unrealistic – did not allow for multiple pages, no headings... Similarly, form designs have omitted obvious features such as the person/address to which the form should be sent.
- Tutors need to "tick off" elements of this syllabus covered as they teach it (which may not necessarily be in the order given below) to avoid missing parts of it.
- The examination will include a compulsory section of 40% for which the answers will require high-level programming code. These questions will be short routines. A candidate who has successfully written programs involving input/output, decisions, loops, arrays, text manipulation and file handling should be able to score close to full marks in this section. In the past, many candidates have rushed into the questions and produced very long routines, some of which was often irrelevant and this would have been obvious had they read the question carefully. It is possible to solve the problems with SHORT routines. Often the question states that the data is ALREADY in variables or arrays and so coding is not required for its input but this is often ignored.
- Students must be encouraged to think HOW they are going to solve a particular problem before actually doing it. Many candidates have scored under 10 out of 40% in the past on the compulsory programming section. Similarly, algorithms have sometimes been ill-thought out. A sample problem is given at the end of this syllabus to show the level of programming required.
- Dry runs should be easy marks for a systematic candidate. Few candidates in past examinations have actually predicted the correct outcomes using a given algorithm and test data.

EXAMPLE OF A LOGIC PROBLEM AND THE EXPECTED ANSWER**QUESTION**

A library allows borrowers to keep books for 14 days. Input a borrowing date (as D, M, Y) and output the date the books are due back (14 days later) taking into account leap years. Integer array MONTH has cells 1 to 12 and holds data that shows the number of days in each of the 12 months of the year for a NON-leap year.

Position	1	2	3	4	5	6	7	8	9	10	11	12
Value	31	28	31	30	31	30	31	31	30	31	30	31

Assume the data is **ALREADY** in the array.

NOTE – questions often state the data is **ALREADY** in the array but in the past many candidates have ignored this and written routines, often incorrect, which earned **NO** marks and wasted valuable exam time. On occasions answers have consisted of **ONLY** this unneeded coding. Candidates are also asked **NOT** to change the variable names given in the question but they frequently do, sometimes leading to confusion when they use the variable name given in the question for a different purpose.

SOLUTION

A pseudo-code algorithmic solution is given here – HIGH LEVEL CODING OF THIS WOULD BE NEEDED IN THE EXAMINATION. Clear indentation is used here to show start and end of multiple-instruction routines such as IF..ENDIF and should be used for loops as well.

INPUT (D,M,Y)

IF Y divisible by 4 (*** see below) testing for leap year

Then MONTH[2] β 29 to allow for leap year or not, reset February

```

Else MONTH[2]  $\beta$  28
ENDIF
D  $\beta$  D + 14           14 days later, but this might be into a new month?
IF D > MONTH [M]     so check against the MONTH table.
    Then D  $\beta$  D – MONTH[M] and if so, move day into next month
        M  $\beta$  M + 1         and move month on by one – but it could be new year?
        IF M > 12
            Then M  $\beta$  1     if so, move month to January
                Y  $\beta$  Y + 1 and move year on to the next year.
        ENDIF
    ENDIF
ENDIF
OUTPUT (D,M,Y)       corrected now date output.
```

***** Might be programmed as** **IF INT(Y/4)*4 = Y** or **IF Y MOD 4 = 0**

Note that this routine is relatively short and would be only slightly longer in a high level language. Approaching this problem, the candidate should note different situations that might occur. In particular is 14 days later

- still in the same month?
- into the next month but still same year?
- into the next month which is then into the next year?

The candidate should recognise at first reading :

- there are no loops
- the month table is already in memory (perhaps by some previously written routine) and the setting up of its initial values is NOT needed.
- decisions (IF statements) WILL be needed to determine
 - a) whether the year is a leap year and what to do about the table,
 - b) how to test for three different date types listed above.

These points will take a few minutes but if thought out in advance should mean that the solution could be written out immediately. The only problem might be that some candidates would not know how to check for divisibility as suggested above. In such situations, it is acceptable for the candidate to write a pseudo-code equivalent to complete the logic AND make a note “I do not know how to do this”. Such honesty would be refreshing and would lose only one mark of the whole. Unfortunately, some candidate’s grasp of syntax has been so poor that the whole solution has been written in note form like this with statements like “add total to the array”. This clearly shows that candidates had little idea how to program for arrays.