Louth Meath Education Training Board - LMETB

Programme Module for

Programming & Design Principles

leading to

Level 5 QQI

Programming & Design Principles 5N2927

Introduction
This programme module may be delivered as a standalone module leading to certification in a QQI minor award. It may also be delivered as part of an overall validated programme leading to a Level 5 QQI Certificate.

The teacher/tutor should familiarise themselves with the information contained in LMETB's programme descriptor for the relevant validated programme prior to delivering this programme module.

The programme module is structured as follows:

1. Title of Programme Module
2. QQI Component Title and Code
3. Duration in hours
4. Credit Value of QQI Component
5. Status
6. Special Requirements
7. Aim of the Programme Module
8. Objectives of the Programme Module
9. Learning Outcomes
10. Indicative Content
11. Assessment a. Assessment Technique(s) b. Mapping of Learning Outcomes to Assessment Technique(s) c. Guidelines for Assessment Activities
12. Grading
13. Learner Marking Sheet(s), including Assessment Criteria

Integrated Delivery and Assessment
The teacher/tutor is encouraged to integrate the delivery of content where an overlap between content of this programme module and one or more other programme modules is identified. This programme module will facilitate the learner to develop the academic and vocational language, literacy and numeracy skills relevant to the themes and content of the module.

Likewise the teacher/tutor is encouraged to integrate assessment where there is an opportunity to facilitate a learner to produce one piece of assessment evidence which demonstrates the learning outcomes from more than one programme module. The integration of the delivery and assessment of level 5 Communications and level 5 Mathematics modules with that of other level 5 modules is specifically encouraged, as appropriate.

Indicative Content
The indicative content in Section 10 does not cover all teaching possibilities. The teacher/tutor is encouraged to be creative in devising and implementing other approaches, as appropriate. The use of examples is there to provide suggestions. The teacher/tutor is free to use other examples, as appropriate. The indicative content ensures all learning outcomes are addressed but it may not

follow the same sequence as that in which the learning outcomes are listed in Section 9. It is the teacher's/tutor's responsibility to ensure that all learning outcomes are included in the delivery of this programme module.

1. Title of Programme Module
Programming & Design Principles

2. Component Name and Code
Programming & Design Principles 5N2927

3. Duration in Hours
150 Hours (typical learner effort, to include both directed and self-directed learning).

4. Credit Value
15 Credits

5. Status
This programme module may be compulsory or optional within the context of the validated programme. Please refer to the relevant programme descriptor, Section 9 Programme Structure.

6. Special Requirements
None

7. Aim of the Programme Module
This programme module aims to introduce the learner to the concepts involved in programming and to equip them with the techniques to design, code and test efficient practical applications on their own and as part of a team.

8. Objectives of the Programme Module

- To explain the historical development of computer programming and differentiate between different programming languages.
- To enable the learner to explore the principles of software design.
- To facilitate the user to understand the basic concepts involved in programming.
- To guide the learner to construct reliable software that has been thoroughly tested.
- To create an awareness of industry standard programming practices.
- To assist the learner to develop the academic and vocational language, literacy and numeracy skills related to **Programming & Design Principles** through the medium of the indicative content.
- To enable the learner to take responsibility for his/her own learning.

9. **Learning Outcomes of Level 5 Programming & Design Principles 5N2927**

Learners will be able to:

1. Demonstrate an understanding of the historical development of computer programming.

2. Demonstrate an understanding of algorithms and their applications in solving real-world problems.

3. Differentiate between programming languages by identifying their distinguishing characteristics.

4. Develop an understanding of the procedural syntax of a modern programming language to include storage (variables and data types), expressions, statements, input/output, reserved keywords and operators.

5. Explain the sequential nature of problem solving and how it relates to the science of computer programming.

6. Summarise a broad range of structured programming and design concepts to include pseudo-code, storage and control structures (selection and iteration).

7. Develop a range of documented computer programs to solve a variety of familiar and unfamiliar specified problems.

8. Utilise a selection of modularisation concepts such as functions, procedures, variable scope and parameter passing.

9. Interpret compiler and linker messages to the extent that an appropriate course of action can be taken to remedy any reported errors.

10. Devise a testing process using structured walk-throughs and debugging tools.

11. Comply with an accepted set of coding standards in their use of comments, indentation and variable naming.

12. Work as part of a team to design, develop, release and review multiple versions of a multi-modular program over an extended period of time.

10. Indicative Content
This section provides suggestions for programme content but is not intended to be prescriptive. The programme module can be delivered through classroom based learning activities, group discussions, one-to-one tutorials, field trips, case studies, role play and other suitable activities, as appropriate.

Section 1: History of programming

Learning outcomes 1 and 3

Explain the historical development of computer programming languages and discuss the need for different languages and their distinguishing features.

- Define a programming language.
- Explain the historical development and evolution of programming languages.
- Explain how programming languages differ from one another and the reasons for different programming languages.
- Explain different programming paradigms, for example, procedural, functional, object oriented, scripting and logic.
- Explain syntax and the need to adhere to the syntax of a programming language.
- Outline the distinguishing characteristics of languages.
    - Outline what they have in common e.g. strict syntax rules, data storage, input statements, output statements, branching, looping.
    - Outline their differences e.g. different syntax, different structures, different focus.
- Explain the different generations of programming languages.
- Explain machine code.
- Explain low level programming, assembly language and an assembler.
- Explain high level languages, compilers, interpreters and translators.

Section 2: Algorithms and pseudocode

Learning outcomes 2, 5 and 6

Enable the user to develop an algorithm and to use pseudcode to design a solution to a problem.

- Explain the sequential nature of problem solving.
- Outline the steps that should be followed when writing a program i.e. design, code and testing.
- Explain how to construct an algorithm.
    - Explain different types of data storage and the range of data that can be stored for each data type.
    - Explain the importance of using pseudocode.
    - Demonstrate pseudocode examples for basic programming problems.
    - Assist the learner in developing good and consistent pseudocode style.

- Explain the importance of developing a comprehensive set of test data to test a program.

Section 3: Basic programming concepts

Learning outcomes 4, 6, 7 and 9

Understand the core principles involved in designing a program. Introduce the learner to the basic concepts involved in programming, including storage, input, output, reserved keywords and mathematical operators. Facilitate the user in creating and debugging programs that use these concepts.

- Familiarise the learner with the structure and basic syntax of a program.
- Explain the sequential nature of program execution.
- Explain reserved keywords to the learner.
- Explain the syntax to output text in a program.
- Create a program that produces simple output e.g. “Hello World”.
- Examine the numeric, character and string data types available in a chosen programming language. For each data type, identify the range of data that it can store and the amount of RAM that it requires.
- Explain the syntax of variable declaration.
- Explain the syntax for reading data from a user.
- Instruct the learner in how to read data from the user and develop a program that reads data from a user.
- Examine the mathematical operators available in the chosen programming language and their precedence.
- Develop programs that use mathematical operators.
- Enable the user to identify and interpret syntax and error messages and to react appropriately.

Section 4: Branching and iteration

Learning Outcomes 6 and 7

Using a chosen programming language, introduce the learner to branching and iteration by creating and debugging programs to include the following:

- List and explain the relational operators available in the chosen programming language and their precedence.
- List and explain Boolean operators.
- Equip the user with the precedence of all operators.
- Explain the syntax of constructing a condition and examine how that condition is evaluated.
- Explain branching/selection (including multi-conditional branching).
- Develop programs that use branching/selection, including multi-conditional branching.

- Explain iteration and list examples where iteration is necessary to solve a given problem.
- Explain the use of post-test, pre-test and counting loops. Identify problems where loops are necessary and discuss the most suitable loop to use.
- Explain the different types of loops available in the chosen programming language and present the syntax of each type of iterative loop.
- Develop programs that demonstrate each type of iterative loop.

Section 5: Modularisation

Learning Outcome 8

Identify the need for modularisation, parameter passing and reusability. Create programs that use functions that can be reused. Examine the scope of variables and determine the most appropriate location for declaration.

- Identify the advantages of modularisation within a program.
- Differentiate between local and global variables and explain variable scope.
- Explain the syntax of a simple module i.e. a module with no parameters and no returned value.
- Develop programs that use modules and use local and global variables according to best practice.
- Explain the advantages/reasons for passing parameters to a module within a program.
- Explain the syntax of a module that accepts parameters.
- Develop modular programs that pass one or more arguments to one or more modules within a program.
- Explain the advantages/reasons for returning a value from a module within a program.
- Explain the syntax of a module that returns a value.
- Develop programs that contain modules that return a value.
- Explain system defined functions.
- Explain how a programmer can access system defined functions.
- Develop programs that use system defined functions.

Section 6: Testing a program

Learning Outcome 10

Facilitate the user to devise suitable test data for each program that they code, to apply this test data to their code and to revise their code until it is accurate and reliable.

- Identify critical test data for each program. Ensure that the test data contains a good mix of values and that incorrect user input is tested and dealt with effectively.
- Develop test results for each item of test data including the output that should result from incorrect/invalid user entry.

- Match test results to actual results.
- Facilitate the user in identifying and correcting any syntax, logical and runtime errors in their code by using structured walk-throughs and debugging tools to locate and correct errors.

Section 7: Best practice

Learning Outcome 11

Provide an awareness of industry standards and best practice in the following areas:

- Coding standards.
- Comments standards.
- Identifier names standards.
- Indentation standard.
- Consistent screen design.
- Ease of use.
- Error trapping and reporting.

Section 8: Programming as part of a team

Learning Outcome 12

Set up teams of between three and five learners to develop one or more programs (defined by the assessor) through all its stages. Facilitate each team in:

- Discussing the problem to be solved.
- Identifying possible solutions to the problem.
- Identifying the best solution to the problem.
- Developing pseudocode for the problem that includes modularisation of the problem.
- Developing test data for the problem.
- Allocating one or more modules to each team member for development with best use of parameters and variable scope.
- Arranging the developed modules into one program.
- Testing the constructed program using the test data developed.
- Revising the developed program until it is reliable, accurate, efficient and adheres to industry standards.
- Releasing the developed program.

11. Assessment

11a. Assessment Techniques

Skills Demonstration	**70% (Practical)**
Exam (Theory)	**30%**

11b. Mapping of Learning Outcomes to Assessment Techniques

In order to ensure that the learner is facilitated to demonstrate the achievement of all learning outcomes from the component specification; each learning outcome is mapped to an assessment technique(s). This mapping should not restrict an assessor from taking an integrated approach to assessment.

Learning Outcome	**Assessment Technique**
1. Demonstrate an understanding of the historical development of computer programming.	Exam
2. Demonstrate an understanding of algorithms and their applications in solving real-world problems.	Skills Demonstration
3. Differentiate between programming languages by identifying their distinguishing characteristics.	Exam
4. Develop an understanding of the procedural syntax of a modern programming language to include storage (variables and data types), expressions, statements, input/output, reserved keywords and operators.	Skills Demonstration
5. Explain the sequential nature of problem solving and how it relates to the science of computer programming.	Skills Demonstration
6. Summarise a broad range of structured programming and design concepts to include pseudo-code, storage and control structures (selection and iteration).	Exam and Skills Demonstration
7. Develop a range of documented computer programs to solve a variety of familiar and unfamiliar specified problems.	Exam and Skills Demonstration
8. Utilise a selection of modularisation concepts such as functions, procedures, variable scope and parameter passing.	Exam and Skills Demonstration
9. Interpret compiler and linker messages to the extent that an appropriate course of action can be taken to remedy any reported errors.	Skills Demonstration
10. Devise a testing process using structured walk-throughs and debugging tools.	Skills Demonstration
11. Comply with an accepted set of coding standards in their use of comments, indentation and variable naming.	Skills Demonstration
12. Work as part of a team to design, develop, release and review multiple versions of a multi-modular program over an extended period of time.	Skills Demonstration

11c. Guidelines for Assessment Activities

The assessor is required to devise assessment briefs and marking schemes for the skills demonstration and an examination paper, marking scheme and outline solution for the examination. In devising the assessment briefs and examination paper, care should be taken to ensure that the learner is given the opportunity to show evidence of achievement of ALL the learning outcomes. Assessment briefs may be designed to allow the learner to make use of a wide range of appropriate media in presenting assessment evidence. Quality assured procedures must be in place to ensure the reliability of learner evidence.

Skills Demonstration	**70%**
Time allocation: 10 hours per each Skills Demonstration to be completed when the relevant learning outcomes have been covered.	

The assessor will devise a minimum of 2 skills demonstrations.

Learning outcomes 2, 7, 9, 10 and 11 are included in both skills demonstrations as they are intrinsic to software development. The assessor should be careful to assess different specific content as indicated in the skills demonstration narratives in order to avoid double assessment i.e. the same aspect of a Learning Outcome being assessed on more that one occasion.

Furthermore, the assessment technique outlines that 3 of the 12 Learning Outcomes are being assessed by both an examination and a skills demonstration. This is intended to give the assessor flexibility when devising assessment briefs and an examination paper. The assessor must ensure that they guard against double assessment and ensure that all learning outcomes are fairly and equally assessed.

<u>Skills Demonstration 1 – 30%</u>

The assessor will devise a practical skills assessment that allows the user to demonstrate their achievement of learning outcomes 2, 4, 5, 6, 7, 9, 10 and 11.

The assessor will develop a problem that requires the learner to:

- Develop an algorithm using pseudocode or a flowchart to solve the given problem.
- Use the developed algorithm to implement a solution to the program. The solution will require the use of:
    - Different data types
    - Input and output statements
    - Mathematical, relational and Boolean operators
    - Selection statements
    - Iteration statements
- Develop test data that thoroughly tests the given problem and apply that test data to the coded solution.

The learner will submit an algorithm, documented code and evidence (e.g. screen shots) of testing the program using the compiled test data. Evidence for this assessment technique will take the form of written, visual and digital evidence~~.~~

All instructions for the learner must be clearly outlined in a skills demonstration brief.

<u>Skills Demonstration 2 – 40%</u>

The assessor will devise a practical skills assessment that allows the user to demonstrate their achievement of learning outcomes 2, 7, 8, 9, 10, 11 and 12.

The assessor will develop a problem that requires the learner to work as part of a team (between 3 and 5 learners inclusive). The learner teams will be required to:

- Develop a top down algorithm using pseudocode or flowchart(s) to solve the given problem.
- Use the developed algorithm to implement a program that uses:
    - Modularisation (with appropriate use of local and global variables).
    - User defined functions designed to encapsulate single specific tasks.
    - Parameter passing between functions where appropriate.
    - At least one user defined function that returns a value.
    - At least one system defined function.
- Develop test data that thoroughly tests the given problem and apply that test data to the coded solution.
- Release a solution to the devised problem.

The learner team will submit an algorithm, documented code (and the released solution) and evidence (e.g. screen shots) of testing the program using the compiled test data. Evidence for this assessment technique will take the form of written, visual and digital evidence~~.~~

All instructions for the learner must be clearly outlined in a skills demonstration brief.

Exam (Theory)	**30%**
Time allocation: 2 hours	

The format of the Theory Exam will be 4 Structured Questions.

(Structured questions are divided into a number of related parts and generally require the learner to demonstrate more in-depth knowledge and understanding of a topic.)

Learning outcomes assessed: 1, 3, 6, 7 and 8.

Evidence for this assessment technique will take the form of written evidence.

All instructions for the learner must be clearly outlined in an examination paper.

12. Grading

Distinction:	80% - 100%
Merit:	65% - 79%
Pass:	50% - 64%
Unsuccessful:	0% - 49%

At levels 4, 5 and 6 major and minor awards will be graded. The grade achieved for the major award will be determined by the grades achieved in the minor awards.

Programming & Design Principles 5N2927	**Learner Marking Sheet 1 Skills Demonstration 1 (Practical) 30%**

Learner's Name: ________________________________ Learner's PPSN: ________________

Assessment Criteria	**Maximum Mark**	**Learner Mark**
Skills Demonstration 1		
• Algorithm ○ Pseudocode or flowchart detailing all tasks to be completed. ○ Complete data dictionary compiled.	5	
• Accurate programming ○ Program compiles. ○ Appropriate data types chosen for variables. ○ Correct use of input statements with suitable input prompts. ○ Correct use of output statements with output appropriately labelled. ○ Correct use of mathematical, relational and Boolean operators. ○ Correct use of selection structures. ○ Correct use of iteration structures.	15	
• Appropriate testing ○ Suitable test data compiled. ○ All computation on test data shown. ○ Correct results shown for each piece of test data. ○ Screen shots included that show results of compiled test data used on coded solution.	5	
• Accepted industry standards for coding ○ Logical sequence to program. ○ Code suitably commented. ○ Indenting conforms to industry standard. ○ Clear and consistent input prompts given to user. ○ Clear and consistent output from the program, suitably displayed.	5	
Total Mark	**30**	

Assessor's Signature: __________________________ Date: ____________________

External Authenticator's Signature: __________________________ Date: ____________________

Programming & Design Principles 5N2927	Learner Marking Sheet 2 Skills Demonstration 2 (Practical) 40%

Learner's Name: ________________________________ Learner's PPSN: ________________

Assessment Criteria	**Maximum Mark**	**Learner Mark**
Skills Demonstration 2		
• Algorithm o Top level algorithm with a suitably detailed algorithm for each task. o Complete data dictionary compiled.	8	
• Accurate programming o Program compiles. o Appropriate data types chosen for variables. o Correct use of functions. o Correct and appropriate use of parameter passing. o Correct and appropriate use of a function returning a value. o Use of a system defined function.	16	
• Appropriate testing o Suitable test data compiled. o All computation on test data shown. o Correct results shown for each piece of test data. o Screen shots of compiled test data used on coded solution.	6	
• Accepted industry standards for coding o Logical sequence to program. o Code suitably commented. o Indenting conforms to industry standard. o Clear and consistent input prompts given to user. o Clear and consistent output from the program, suitably displayed.	5	
• Evidence of good team participation o Evidence of each learner's contribution.	5	
Total Mark	40	

Assessor's Signature: ___________________________ Date: ____________________

External Authenticator's Signature: ___________________________ Date: ____________________

Programming & Design Principles 5N2927	**Learner Marking Sheet 2 Examination Theory 30%**

Learner's Name: ________________________________ Learner's PPSN: ________________

Assessment Criteria	**Maximum Mark**	**Learner Mark**
4 Structured Questions Answer all questions (15 marks each) Question No.: __________ __________ __________ __________	 15 15 15 15	
Subtotal	60	
Total Mark (Divide Subtotal by 2)	30	

Assessor's Signature: __________________________ Date: ____________________

External Authenticator's Signature: __________________________ Date: ____________________