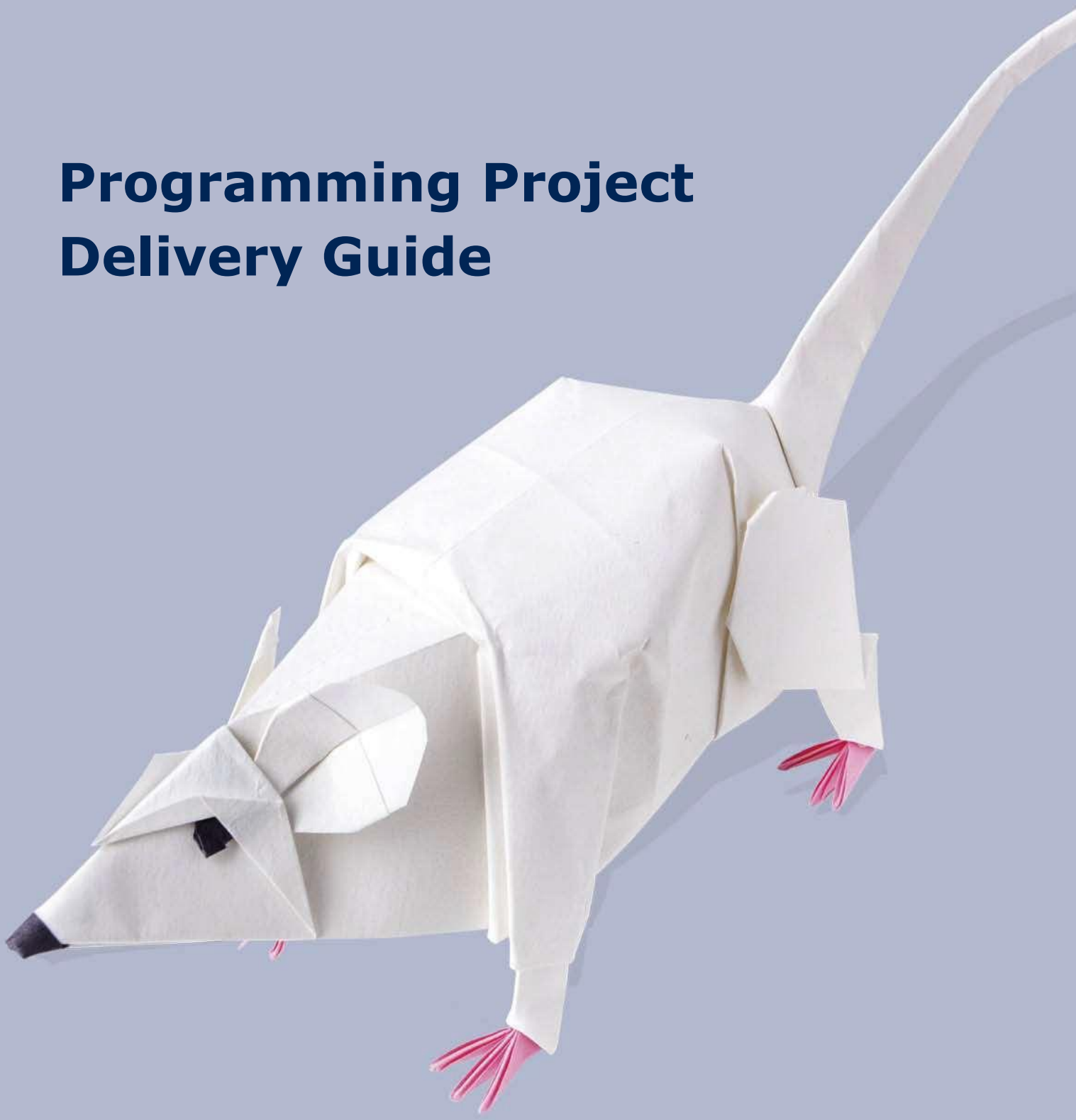
Programming Project Delivery Guide



GCSE (9-1) Computer Science

Pearson Edexcel Level 1/Level 2 GCSE (9-1) in Computer Science (1CP1)
Version 6

Contents:

Programming Project Delivery Guide 3
Requirements for the Programming Project 3
Key dates for the Programming Project 3
Overview 4
Programming languages 4
Scheduling the Programming Project 5
The Secure Environment 7
Student accounts 7
Electronic access outside the student accounts 7
Materials in the supervised environment 7
Additional resources 7
Storing materials securely 8
Security and backups 8
Guidance and feedback 8
Malpractice 9
Submission of work 10
Delivering the Programming Project 11
Stage 1: Analysis 13
Stage 2: Design 13
Stage 3: Implementation 16
Stage 4: Testing, refining and evaluation 18
Appendix 1 Programming Project criteria 20

Programming Project Delivery Guide

The purpose of this document is to help you prepare your students to complete the Edexcel GCSE (9-1) Computer Science project. In January 2018 Ofqual announced that the non-exam assessment (NEA) component of GCSE (9–1) Computer Science will no longer count towards the qualification grade. Nevertheless, students are still required to undertake a programming project. It gives practical advice as well as guidance on the required regulations to follow.

Information on the project is on **pages 14 to 19** in the Edexcel GCSE (9-1) Computer Science specification. This guide is not intended as a replacement to the information already published but provides supporting guidance to assist teaching and learning.

Requirements for the Programming Project

- 20 hours must be set aside in the timetable to allow students to undertake the project. They may not work on their solutions outside of classroom time. Students' work must be individual.

- Students' written accounts of their programming project must represent their individual work, cover each part of the project and reference any resources used or support given.

- The programming language used is a high-level programming language that has a textual program definition.

Key dates for the Project

There will be a bank of project briefs published on the Pearson website, which are valid for the lifetime of the qualification. Students must complete all activities within one project brief. Centres should ensure that students have the necessary knowledge and skills required before undertaking the programming project.

The deadline for submission of all programming project work will be 15th May in the year of certification. See the 'Submission of work' section for further details.

Overview

The purpose of the programming project is to develop students' skills in responding to computer science problems. The GCSE in Computer Science requires each student to undertake a programming project, where they will develop a computer program. The programming project is not assessed and does not contribute towards the final grade.

Pearson will provide a bank of programming project briefs that involve a problem that students will need to solve by developing a computer program.

If a data file is required to complete the programming project, this will be provided by Pearson. Centres should check any data files included with the programming project briefs to make sure they work with the software to be used during Stage 3: Implementation.

Students will submit their program with a written report showing how the program was developed, tested and evaluated. The programming project will require students to create a program that will include the following:

- data input and storage
- processing data
- producing output based on processed data.

There are four different stages in the project. These are:

1. Analysis
2. Design
3. Implementation
4. Testing, refining and evaluation.

Programming languages

There are five language choices for the programming project. These are:

- Python
- Java
- C-derived
- Visual Basic.NET
- Pascal/Object Pascal.

Scheduling the Programming Project

Schemes of Work

The assumption in this document is that students will complete the whole course over a period of two years, with two hours per week timetabled.

- It is recommended that students undertake the programming project in Year 11, having spent Year 10 developing their understanding of the theory topics and programming skills detailed in the specification.
- However, students may undertake the programming project in either Year 10 or Year 11.

A suggested Course Planner and detailed Schemes of Work for a two-year programme are available here:

http://qualifications.pearson.com/en/qualifications/edexcel-gcses/computer-science- 2016.coursematerials.html#filterQuery=category:Pearson-UK:Category%2FTeaching- and-learning-materials

Teachers are free to use their own planners and schemes of work, or adapt the documents provided by Pearson to fit their requirements.

Programming Project Preparation Scheme of Work example

This can be found in the Autumn Year 11 and Spring Year 11 Schemes of Work support materials at https://qualifications.pearson.com/en/qualifications/edexcel- gcses/computer-science-2016.coursematerials.html#filterQuery=category:Pearson-UK:Category%2FTeaching-and-learning-materials&filterQuery=category:Pearson- UK:Document-Type%2FScheme-of-work

Total time allocated to working on the Programming Project

Students must be provided with 20 hours of timetabled time to complete the programming project. Teachers need to consider how to accommodate this within centre timetables. For example, if a class is scheduled for 1 hour, time is taken coming into the classroom, logging in and distributing materials. This could mean that the time spent in each lesson is maybe 45 to 50 minutes. Teachers might find it helpful to time this in a trial, and then build it into their programming project schedule. It is advisable to keep a log of the sessions.

Students who miss lessons are entitled to catch up time, and some students may have extra time allocated to them. This also needs to be timetabled.

Teachers can
Exclude time taken getting students ready to start working in each session from the 20 hours.
Allow extra time for students who are entitled to this or who have missed lessons.

Recommended time for the different stages in the Project

It is important that teachers keep track of their students' progress to ensure that students spend sufficient time on each stage.

Stage 1 Analysis

- It is recommended that students spend 2 hours on Stage 1.

Stage 2 Design

- It is recommended that students spend 6 hours on Stage 2.

Stage 3 Implementation

- It is recommended that students spend 10 hours on Stage 3.

Stage 4 Testing, Refining and Evaluation

- It is recommended that students spend 2 hours on Stage 4.

The Secure Environment

Programming Project Student Account

Centres must be able to authenticate that students have produced their own individual work. There is no requirement for students to be set up with special programming project accounts. However, students can only work on their program within the 20 timetabled hours.

Teachers can
Create an electronic syntax help guide, including basic syntax structure details.
Give the students a printed copy of the syntax help guide.
Give the students a copy of the pseudocode command set booklet.

Electronic access outside the student accounts

Teachers can	**Teachers cannot**
Allow students to use an intranet.	
Allow access to the Internet.	
	Allow students to bring in work to the environment electronically or remove work electronically (e.g. on USB or other portable storage media.)

Materials in the supervised environment

Students can only work on their programming project in the supervised sessions, which means that materials in any form cannot be removed or taken into those sessions.

Students may bring reference materials into the supervised environment. Other information, in addition to the syntax guide and pseudocode command set booklet, can be searched for on the Internet, although these sites must be referenced if material is taken from them. If students are allowed to copy and paste large sections of code into their projects then teachers will find it difficult to authenticate the work as the student's own.

Additional resources

Students should be provided with access to additional resources, such as:

- Flow chart symbols
- The pseudocode command set booklet from the specification
- A syntax guide **

- The testing template provided with the project
- Integrated Development Environment for the programming language selected, including the help facility
- Additional software that does not impact upon the integrity of the project
- A subject specific textbook.

** The syntax guide should be a language reference describing all the facilities of the programming language selected with examples of how to construct each facility using the correct syntax. It may be paper based or provided as a digital resource.

Storing materials

It is vital that students' work is stored securely.

Security and backups

It is the responsibility of the centre to keep secure the work that students have submitted.

Centres are strongly advised to utilise firewall protection and virus checking software and to employ an effective backup strategy, so that an up-to-date archive of students' evidence is maintained.

Centres should keep a secure backup of all students' work.

Guidance and feedback

The work submitted by students must be their own. However, teachers can support students throughout the project to ensure that they get the best from this problem-solving experience.

Teachers can	**Teachers cannot**
Give generic class-wide feedback or instruction to help students understand rubrics, programming project criteria and controlled conditions. This should be of a general nature and be concerned with overriding guidance.	Provide templates, model answers or writing frames. Students should, however, use the test plan template provided with the programming project brief.
Review students' work and provide oral and/or written feedback to support students to progress through each stage of the programming project.	Demonstrate an example of a solution to the current project brief or a closely related problem.
	Use the actual programming project brief in practice sessions.

Co-operation, not collaboration

The project submitted must be the student's own work, as authenticated by the teacher. Students must be supervised throughout the programming project. However, students should be encouraged to discuss ideas with their teacher and each other whilst working independently. In other words, students may co-operate with each other but must not collaborate on a joint solution to the problem.

Malpractice

Teachers should make sure students are aware of what constitutes malpractice and the consequences of malpractice.

Any contravention of the guidelines in this document or relevant JCQ instructions will be considered as malpractice.

Submission of work

The Programming Project Authentication form (PPA) will be used to authenticate the work as being completed by the student. It contains a section for student information and a section for teacher authentication.

This document is available on the Pearson website:

https://qualifications.pearson.com/en/qualifications/edexcel-gcses/computer-science-2016.coursematerials.html#filterQuery=Pearson-UK:Category%2FForms-and-administration

The PPA form must be completed and signed by both the student and the teacher.
A completed copy of the PPA should be included with all submissions.

Presentation

Students must present their work for the programming project electronically.
Each student's work should be saved into a folder called Report. The project brief provides students with instructions for saving their work.

The Head of Centre Declaration form must be signed by the Head teacher to state that all procedures have been followed correctly.

This document is available on the Pearson website:

https://qualifications.pearson.com/en/qualifications/edexcel-gcses/computer-science-2016.coursematerials.html#filterQuery=Pearson-UK:Category%2FForms-and-administration

The signed PPA form(s) and the Head of Centre Declaration form may be sent as hard copies or scanned and included in an electronic format along with the student work. The work for all candidates must be submitted in an approved digital format; that is, on CD-ROM, DVD or USB flash drives.

All work must be sent to:
Unmarked coursework team - GCSE Computer Science - Programming Project
Lowton House, Lowton Way, Hellaby, S66 8SS

The work will not be sent back to centres and copies should be kept at the centre in case an additional copy might be needed. Due to the work not being assessed, we will no longer be supplying centres with labels.

Delivering the Programming Project

Programming Project stages detail

This section gives guidance on how to deliver each stage of the programming project. It should be read in conjunction with the GCSE (9-1) Computer Science specification from Edexcel. Teachers should provide sufficient support to students at each stage of the programming project to ensure that they make progress and gain maximum benefits from the experience.

The programming project requires students to make decisions and assumptions of their own. They will have to work out a number of approaches by themselves, as the programming project brief will not give them all the smallest details. The Edexcel problem is an 'open' problem to encourage creativity when creating the solution, so there is no definitive solution. Please refer to Appendix 1 for the Programming Project marking criteria if you decide to mark students' work and give feedback.

A specimen project brief is available on the Pearson website.

Stage 1: Analysis

Purpose

The purpose of the analysis stage is to identify the requirements of the problem and what the proposed solution will do to meet the requirements.

The analysis tasks are to:

- analyse the given problem and identify the requirements of the program that will be designed, implemented and tested
- decompose the problem into manageable sub-problems, with an explanation of each.

Report contents

The report folder will contain:

- a short introduction to the problem
- a list of the requirements of the problem that will be programmed
- decomposition of the problem into sub-problems, including:
  - a short description of what each of the sub-problems will do
  - a short explanation of the reasoning behind the decomposition submitted.

Delivering the Analysis

The requirements of the problem are key and are followed throughout the programming project from the analysis to the evaluation. By explaining the reason behind the decomposition, students are applying computational thinking

to the problem. During programming project preparation, encourage students to look at other problems, breaking them down into sub-problems to identify the requirements. They could use highlighters to identify these requirements, then practice putting them into succinct prose or bullet points which are measurable, with a short explanation about why they have decomposed like this.
Sample Assessment materials and previous programming projects can be used for this.

For example:

I will create a menu to give the user 3 options which will then give more menu options.

1. *Add data*
    - *Customer data*
    - *Product Data*
2. *Buy*
    - *Choose what to buy and put in basket*
    - *Checkout*
3. *Show account*
    - *Display*
    - *Edit*

I chose to split the menu into 3 main options with sub-menus rather than 9 main options because it will be easier to use by the customers.

Students will need to make some assumptions and decisions of their own and assumptions should be stated at this stage.

The report asks for 'short' entries and this section is therefore unlikely to be more than two pages long.

Teachers can
Provide sufficient support to enable the learner to identify the requirements of the problem and/or data requirements so that they can carry on to the next stage of the programming project.

It is very important that teachers see the complete Analysis before students start on the next stage.

Stage 2: Design

Purpose

The purpose of the design stage is to describe what has to be done when implementing the solution and to suggest an appropriate strategy to test the solution.

There are two sub-stages:

2.1 Solution design

2.2 Test strategy and initial test plan

2.1 Solution design

The solution design task is to:

- Design an algorithm that meets the requirements of the problem using appropriate conventions (flowchart, pseudocode).

Report contents

The report folder will contain:

- the algorithm(s)
- any refinements to the design identified during implementation, with reasons.

Delivering the Solution design

Students should depict each of the sub-problems they have identified in the Analysis in a flowchart, or pseudocode, or a written description. They can use all three conventions if they want to.

Refinements are usually identified during implementation or testing. This reflects 'real life' design and development when changes are required. It would be very unusual for a design to exactly mirror the final implementation.

An 'open' problem will be given which may result in many alternative, correct solutions.

Students must not build the program and then reverse engineer the design.

Teachers can
Show students what the algorithms should include
Remind students that decomposition is one of the requirements identified in their Analysis.
Encourage students to note their thoughts as they go through the decomposition process.
Refer students to the pseudocode booklet.
Provide sufficient support to enable the student to develop algorithms to use in the implementation phase.

2.2 Test strategy and initial test plan

The task is to:

- Devise a test strategy based on meeting the requirements of the problem. The proposed strategy should be followed when creating an initial test plan. This must be completed before implementation and will be updated before the program is actually tested.

Test planning is done twice in the project, the first time is in this stage, and the second time in Stage 4.

Report contents

The report folder will contain:

- the initial test plan, (which will follow the test strategy), using the headings shown.

Test no	Purpose of the test	Test data	Expected result

When constructing test data for the initial test plan, normal data is data that the program will accept. Erroneous data is inaccurate data that the program will not accept. Boundary data is typically on the 'edge' of a range of possible values that may or may not be accepted. Not all tests may require data entry.

This is an initial test plan and more tests can be added later. At this stage the

students should be thinking how to test their solution on how it meets the requirements they have identified.

During preparation work for the programming project and whilst learning to program, students should be reminded that tests to cover normal, erroneous, and boundary data should be included where possible so that it becomes part of good practice.

It is very important that teachers see the Initial Test Plans before students begin to develop the program code.

Teachers can
Teach students the meaning of normal, erroneous and boundary data before the programming project commences.
Remind students that this test plan is based on the requirements identified.
Explain the process of creating the Initial Test Plan before building the solution, then adding refinements to create the Final Test Plan in Stage 4.
Indicate the aspects of the program that need to be tested.
Provide support so that students are not prevented from carrying out some tests in Stage 4.

Teachers should
See the Initial Test Plan BEFORE students start to program the solution
Tell students that it should NOT be worked on once the students start developing the program code in Stage 4.
Ensure that the Initial Test Plan is saved separately from the Final Test Plan.

Stage 3: Implementation

Purpose

The purpose of the implementation stage is to program the solution to the problem.

There are two sub-stages:

3.1 Implementing the design

3.2 Building the solution

The two sub-stages in implementation (3.1 and 3.2) will happen concurrently as the solution develops. It is not possible to build a solution without implementing the design using programming concepts.

Report contents

The report folder will contain:

- the source code and all the files required to execute the program. Any design refinements should be noted as comments in the program code.
- screenshots demonstrating effective use of debugging skills to correct errors.

3.1 Implementing the design

This is where the algorithm(s) from the design are translated into the chosen high-level programming language. Students will apply the programming concepts from that language, e.g. they will apply the correct syntax of a pre-check loop.

Students can be supported at this stage.

Teachers can
Support learners to program their solution to the problem.
Provide support by explaining syntax in general terms.
Provide a syntax guide (electronic or printed) for the chosen language.
Guide students to debug their code.

3.2 Building the solution

The final programmed solution should address all the requirements listed in the analysis stage and be functional. Computing techniques (comments, naming conventions, indentation) should be used.

Students can be supported at this stage.

Teachers can
Support students to build their solution.
Provide a syntax guide (electronic or printed) for the chosen language.
Guide students to debug their code.

Delivering the solution build

Developing programmed solutions is still computational thinking and this is the stage where it is most likely that gaps will be identified which can then be added as refinements. These refinements should be implemented and documented in the program code by using comments as descriptors. A refinement can include many things, for example a more efficient way of programming a sub-problem or an additional component that has not been previously considered but which will add to the functionality of the solution.

Students should be familiar with debugging techniques (hand-tracing, and the chosen language's debugging tools). By the time they get to the programming project, debugging should be a normal part of programming for students and should be used when errors arise during this stage.

Evidence that students can and do use debugging is required here, with screenshots. Students should submit one or two screenshots showing debugging skills in actual use.

Stage 4: Testing, refining and evaluation

Purpose

The purpose of this stage is to show that the final program solution has been tested along with any refinements, and the solutions evaluated against the original requirements.

Report contents

The report folder will contain:

- the updated and completed Test Plan (labelled and saved as 'TestTable')
- the evaluation.

Delivering the Testing, refining and evaluation stage

Columns to show the 'Actual result' and 'Action needed/comments' should be added to the initial test plan. In this stage, the actual tests are then carried out and these last two columns should be completed when each test is run. Any errors should have 'Action Needed/Comments' entries. An attempt should be made to correct and retest all errors.

Tests for any refinements completed in the design and/or implementation stages should be added to the end of the test plan and carried out.

Test no	Purpose of the test	Test data	Expected result	Actual result	Action needed/ comments

A selection of informative screenshots should be provided as evidence of the results of significant tests.

Teachers should remember that they will have to authenticate students' work, which will include testing evidence.

The evaluation should include a thorough and critical evaluation of the program. This should include how successfully the program meets each of the original requirements and the reason for adding refinements to the final solution.

The evaluation should **not** contain a log of what the student has done during the project.

Students can be supported at this stage.

Teachers can
Support students to understand errors in their program.
Provide guidance to students regarding their evaluation of the programming project.

Appendix 1. Programming Project criteria

If teachers mark students' work, please use the following programming project criteria.

Stage 1: Analysis

The purpose of the analysis stage is to identify the requirements of the problem, and what the proposed solution will do to meet the requirements.

The analysis tasks are to:

- analyse the given problem and identify the requirements of the program that will be designed, implemented and tested
- decompose the problem into manageable sub-problems, with an explanation of each.

An introduction to the problem, in prose, will demonstrate an understanding of abstraction. The decomposed list of requirements can be in prose or as a bulleted list, each one clearly identified.

Decomposition requires choices to be made, in this case by breaking the given problem down into sub-problems which will be designed and implemented later. A description of what each sub-problem will do is required, it can be in prose or as a bulleted list. An explanation of the reasons why the decomposition submitted is the most appropriate to meet requirements must also be included, in prose.

Report content for analysis

For this stage, the report should include:

- a short introduction to the problem
- a list of the requirements of the problem that will be programmed
- decomposition of the problem into sub-problems, including
- a short description of what each of the sub-problems will do
- a short explanation of the reasoning behind the decomposition submitted.

Level	Mark	Descriptor
	0	• No awardable material
Level 1	1–2	• Attempts to identify some requirements of the problem, showing limited understanding of abstraction and decomposition. • Generic statements or judgements may be presented for the selection of the sub-

		problems.
Level 2	3–4	• Identifies the requirements of the problem, with some elements missing, showing some understanding of abstraction and decomposition. • An explanation for the judgements made on the selection of the most appropriate sub-problems is given with some occasional links present so that lines of reasoning are partially supported and mostly clear.
Level 3	5–6	• Clearly identifies the requirements of the problem, showing comprehensive understanding of abstraction and decomposition. • An explanation for the judgements made on the selection of the most appropriate sub-problems is given with comprehensive links evidenced so that lines of reasoning are well supported, clear and concise.

Stage 2: Design

The purpose of the design stage is to describe what has to be done when implementing the solution and to suggest an appropriate strategy to test the solution.

2.1 Solution design

An algorithm or algorithms should be designed that meet the requirements of the problem using appropriate conventions (flowchart, pseudo-code, written description). Program code using the chosen language must **not** be included in the design solution.

The algorithm(s) should:

● show detailed decomposition into sub-problems and how they link together (if appropriate)

● demonstrate clear abstraction (for example by including parameterisation, links between components)

● include inputs, processes and outputs

● use all three basic programming constructs: sequence, selection and iteration.

Report content for solution design

For this stage, the report should include:

● the algorithm(s)
● any refinements to the design identified during implementation, with reasons.

Level	Mark	Descriptor
	0	• No awardable material
Level 1	1–3	• In the algorithms there are significant errors in logic and in the use of programming constructs, leading to an overall solution which is non-functional. • Little or no attempt at decomposition has been made. • Limited attempt to address the requirements of the problem.
Level 2	4–6	• In the algorithms some parts of the logic and use of programming constructs are functional, leading to an overall solution which is partially functional. • An attempt at decomposition into subprograms has been made. • Requirements of the problem are partially addressed.
Level 3	7–9	• The algorithms are well designed but there are minor errors in logic and in the use of programming constructs, leading to an overall solution which may not be completely functional. • Full decomposition into subprograms has been made with minor errors so does not lead to clear abstraction. • Most of the requirements of the problem are addressed.
Level 4	10–12	• The algorithms are well designed and there are no errors in logic and in the use of programming constructs, leading to an overall solution which is fully functional. • Full decomposition into subprograms has been made with clear abstraction. • All requirements of the problem are fully addressed.

2.2 Test strategy and initial test plan

A test strategy for the solution should be devised based on meeting the requirements of the problem. The test strategy should explain the approaches that the student will use when testing the program. The proposed strategy should be followed when creating an initial test plan, this must be completed before implementation and will be updated before the program is actually tested.

The test strategy should be in prose.

An example of a table that could be used for the initial test plan is shown below. When constructing test data for the initial test plan, normal data is data that the program will accept. Erroneous data is inaccurate data that the program will not accept. Boundary data is typically on the 'edge' of a range of possible values that may or may not be accepted. Not all tests may require data entry.

Report content for test strategy and initial test plan

For this stage, the report should include:

- the test strategy
- the initial test plan, using the headings shown, with all first four columns completed. It should be labelled 'Initial Test Plan'.

Test no	Purpose of the test	Test data	Expected result	Actual result	Action needed/ comments

Level	Mark	Descriptor
	0	• No awardable material
Level 1	1–2	• Devises a limited test strategy that may not link to the requirements of the problem. Generic comments are made about the approaches for planning the testing of the program. • The test plan shows limited evidence of following the test strategy and includes: - a vague description of the purpose of each test - coverage of normal or erroneous or boundary data - expected outcomes but do not reference their test data.
Level 2	3–4	• Devises a substantially completed test strategy that covers some of the requirements of the problem and a partial explanation about the approaches for planning the testing of the program. • The test plan shows occasional evidence of following the test strategy and includes: - a partial description of the purpose of each test - coverage of two of normal, erroneous or boundary data - expected outcomes with some appropriate reference to their test data.
Level 3	5–6	• Devises a comprehensive test strategy that covers most of the requirements of the problem and a clear explanation about the approaches for planning the testing of the program. • The test plan shows consistent evidence of following the test strategy and includes: - a clear description of the purpose of each test - coverage of normal, erroneous and boundary data - expected outcomes with appropriate reference to their test data.

Stage 3: Implementation

The purpose of this stage is to program the solution to the problem.

It may be that amendments to the original design solution become apparent during this stage and these refinements should be implemented and documented as additions to the design and in the program code by using comments as descriptors. A refinement can include many things, for example a more efficient way of programming a sub-problem or an additional component that has not been previously considered but which will add to the functionality of the solution. Refinements will be awarded in stage 4.

The two sub-stages in implementation (3.1 and 3.2) will happen concurrently as the solution develops. Both sub-stages should, therefore, be marked at the same time using all the submitted code.

Report content for implementation

For this stage, the report should include:

- A copy of the program code. Any refinements should be noted as comments in the final program.
- Screenshots demonstrating effective use of debugging skills to correct errors.

3.1 Implementing the design

The design algorithm(s), as abstract decomposition, need(s) to be translated into the high-level programming language that has been chosen. This process requires applying an understanding of programming concepts when using this programming language during the implementation.

Level	Mark	Descriptor
	0	• No awardable material
Level 1	1–2	• The translation of the design into the programming language shows a limited application of programming concepts.
Level 2	3–4	• The translation of the design into the programming language shows an adequate application of programming concepts.
Level 3	5–6	• The translation of the design into the programming language shows a comprehensive application of programming concepts.

3.2 Building the solution

The final programmed solution should address all the requirements listed in the analysis stage. It should be functional, which will be aided by the choice of appropriate programming concepts. Computing techniques (comments, naming conventions, indentation) should be use to improve readability and aid understanding.

If errors arise during this stage, the language's debugging tools and hand-tracing should be used, as appropriate.

NOTE: formal, recorded testing is carried out in stage 4.

Level	Mark	Descriptor
	0	• No awardable material
Level 1	1–3	• The program has made little or no attempt to address the requirements of the problem. • A limited solution has been built, showing little or no debugging skills. • Programming constructs, data validation and the choice of data types and structures leads to an overall program which is non-functional. • Little or no attempt at decomposition has been made and computing techniques are used but they are ineffective in making the program clear and easy to understand.
Level 2	4–7	• The program addresses the requirements of the problem but there are major omissions. • A limited solution has been built with significant syntax and logic errors, showing limited use of debugging skills. • Subprograms, programming constructs, data validation and the choice of data types and structures lead to an overall program which is partially functional. • An attempt at decomposition into subprograms has been made and computing techniques are used to make some components of the program clear and easy to understand.
Level 3	8–11	• The program addresses some requirements of the problem only. • A partial solution has been built with some syntax and logic errors, showing some use of debugging skills.

		• Subprograms, programming constructs, data validation and the choice of data types and structures lead to a program which may not be completely functional. • The program has been partially decomposed into subprograms and computing techniques are used to make most components of the program clear and easy to understand.
Level 4	12–15	• The program addresses the requirements of the problem with some omissions. • A full solution has been built with some logic errors, showing the use of debugging skills. • Subprograms, programming constructs, data validation and the choice of data types and structures lead to an overall program which is functional with minor omissions. • The program has been fully decomposed into subprograms with minor errors and computing techniques are used to make the program clear and easy to understand.
Level 5	16–18	• The program fully addresses the requirements of the problem with minor omissions. • A full solution has been built with little or no logic errors, showing the effective use of debugging skills. • Subprograms, programming constructs, data validation and the choice of data types and structures lead to an overall program which is fully functional. • The program has been fully decomposed into subprograms and computing techniques are used to make the program unequivocally clear and easy to understand.

Stage 4: Testing, refining and evaluation

The purpose of this stage is to show that the final program solution has been tested along with any refinements, and the solutions evaluated against the original requirements.

Columns to show the 'Actual result' and 'Action needed/comments' should be added to the initial test plan and completed when each test is run. An example is shown. Any errors should have 'Action Needed/Comments' entries. An attempt should be made to correct and retest all errors.

Report content for testing, refining and evaluation

For this stage, the report should include:

- the updated and complete Test Plan (labelled 'Final Test Plan')
- the evaluation.

Test no	Purpose of the test	Test data	Expected result	Actual result	Action needed/ comments

Tests for any refinements completed in the design and/or implementation stages should be added to the end of the test plan and carried out.

The evaluation should include a thorough and critical evaluation of the program. This should include how successfully the program meets each of the original requirements and the reason for adding refinements to the final solution.

Level	Mark	Descriptor
	0	• No awardable material
Level 1	1–3	• Components and some requirements have been tested, using one of normal, boundary or erroneous data with no attempt to test that the subprograms interact. • If errors have been identified, there is little or no attempt to overcome them. • Little or no refinements were identified and implemented in the design or implementation stage. • Isolated comments have been made in the evaluation, showing how successfully the program meets some of the specified requirements.
Level 2	4–6	• Components and some requirements have been tested using two of normal, boundary or erroneous data. Little attempt to test that the subprograms interact. • If errors have been identified, there is some attempt to correct some of them. • Limited refinements were identified and implemented in the design or implementation stage. • An evaluation that identifies some strengths, demonstrating how successfully the program meets the specified requirements.
Level 3	7–9	• All components and the requirements have been fully tested using normal, boundary and erroneous data with minor omissions. Attempts to test that the subprograms interact. • If errors have been identified, the program is corrected to overcome most of them or there is an attempt to suggest how to fix the errors in the test table. • Refinements were identified and implemented in the design or implementation stage and they improve the functionality of the solution. • An evaluation that identifies some strengths and weaknesses, demonstrating how successfully the program meets the specified requirements.

Level 4	10–12	• All components and the requirements have been fully tested using normal, boundary and erroneous data. Testing includes confirmation that the subprograms interact. • If errors have been identified, the program is corrected to overcome them or a clear description is provided on how to fix the errors in the test table. • Refinements were identified and implemented in the design or implementation stage and they improve the functionality and robustness of the solution. • An evaluation that critically reviews how successfully the program meets the specified requirements.