Grade 12

2021

### Performance Assessment Task

#### Programming Project

The programming project represents the development cycle of a product. You are to create a single programming application for a real-world application based on any topic which has relevance for you. The project must demonstrate knowledgeable use of the programming language used to produce the application.

#### Topic & Content

You are required to find a sample selection of future users of the software and research what their requirements are. The content should indicate that you have mastered what has been taught in the programming lessons i.e. data structures, objects, file manipulation, program structures, etc. If a text file has been used as a data source then the program must have sufficient complexity with Objects and Classes to be of the correct standard. The programming project must include Objects and Classes and/or a Database as a data source, and a GUI. It must use several Interlinked screens that satisfies the principles of good UI design. The project topic may be a game, a business management system or a solution to a real world problem such the school tuck shop including ordering, storage of appropriate (see POPI Act) student data and calculations of profits

The program needs to meet the following criteria:

* **User friendly**
* Interface - the interface must be user friendly (preferably a GUI), easy to use and task appropriate. If the program does not use a GUI there has to be a very good, task appropriate reason for this (which is adequately explained in the design document). This GUI should satisfy the criteria listed the following Assessment Standards: Use of the most appropriate component for input / output for the problem.
* Uncluttered screens with effective use of colour.
* Appropriate user prompts and error messages based on exceptions caught.

The error message returned should indicate a solution.

* Metaphors or images (e.g. a picture of a printer on a print button)
* Consistent behaviour, e.g. always using F1 for Help or ESC to stop a process.
* An effective help system - Online*/electronic Help / Context sensitive help*
* Shortcuts
* Use of a system of several interlinked screens which can successfully

be navigated by end-users

* Data Flow / program operation – the learner must ensure that the sequence of steps required to use the program and complete a task are clear, easy to follow and logical.
* **Storage / Data persistence** – data must be stored and retrieved from session to session (this can be in the form of conventional files OR a database OR both). Database work is strongly encouraged, as is the use of SQL. Ensure that the storage is appropriate to the program (e.g. a game should have the ability to save & load a game & have high scores, etc)
* **Separation of interface & engine** – the ‘working code’ **must not be embedded in the interface** (i.e. it must be in separate classes / units). Communication between the interface & the working code is in the form of parameters & typed methods (functions). A limited amount of code in the interface is acceptable only if suitably justified in the planning.
* **Good data internal structures** – There has to be some form of internal representation of data (i.e. classes / records / arrays – or any combination of these). Data structures must be logical and task appropriate. Classes should be used.

N.B. In order to reduce the risk of plagiarising an existing project, it is advisable to design and code a complete new game, or design and code a solution for a new situation. It is vital that any external code used does not comprise more than 20% and is suitably referenced. The project can be implemented on one or more desktop computers, as a mobile application or on a separate hardware device such a Raspberry PI or Arduino, or any combination thereof. You must have written a substantial amount of the solution and must not have not relied too heavily on utilities and functions provided by the solution development tool such as Unity in game development.

The most important aspect of this project is that the program should work properly. Aim for a shorter, simpler program that works 100%. You are expected to research existing solutions to the project to determine whether there is an existing solution. Any project that has a complete solution in existence should be avoided. It is recommended that you consider solutions to national or global problems described in the National Development Plan 2030 (NDP 2030). These problems described in the NDP are related to the 17 global goals listed in the Sustainable Development Goals (SDGs) set by the United Nations.

**It is vital that there is evidence of planning in your project.**

The following items need to be handed in at the end of the project:

* Project Specification Document
* Design Document
* Technical Document
* Testing Document
* The program code, compiled program and all documentation, as well as comprehensive and valid data files.

Referencing

You may make use of other sources such as books, Internet, friends, teachers. These sources must be referenced using the Harvard Standard in the appropriate document.

Authentic work

You will be required to sign a ***Declaration of Authenticity document***.

Documentation Requirements

1. Project Specification Document
   1. A title page: Include the project name, ‘Project Specification Document’ and

your name.

* 1. A summary of the features of the project together with the target user group.
  2. A brief discussion of similar projects as a result of research. Research must be correctly referenced and cited. A motivation is included for the project indicating how this project differs from existing projects. This section should NOT be longer than a page.
  3. Project Functions, Justification, and Success Criteria – a description of the goals

that are to be achieved by the project, a justification for choosing this

project, and a list of criteria against which the project can be measured to

determine the success of the project. Features of the project – an explanation of your success criteria and what the program actually does. These features should be very specific.

* 1. Specifications of help - what sort of help is going to be included and how is the user going to access it. A variety of Help options must be completely described. Specify how help related data will be stored. Help provision must be appropriate for the type of application being developed.
  2. Specification of Permanent Data Storage – specify and correctly describe all data fields and the types of storage being used.
  3. Hardware, Software and Installation Requirements- provide a complete list of hardware and software required to run your program including development software i.e. IDE software, secondary software such as database software, SQL connectors. Also include software versions.

1. Design Document (N.B. no code is needed for this section)
   1. A title page: Include the project name, ‘Design Document’, your name.
   2. A table of contents.
   3. Interface Design - ALL interfaces linking the program to the outside environment including the user must be completely specified in this section. This section should describe each screen of the project. Data source and description (e.g. keystroke from user, joystick movement from user, server providing other players' data, temperature gauge providing temperature). NB: Screen capture from a design tool (including IDE), sketches and mock-ups are acceptable.
   4. Program Flow – describe the flow of events in the program using either flow charts or pseudocode to show how the program works internally in relation to the interface(s). Sequencing must cover all aspects of the functions and features listed in the Specification Document. The flow should be clear, well represented and easy to understand. There should be no logical gaps.
   5. Class Design and OOP Principles - provide class designs represented as a UML class diagram with class name, fields and methods demonstrating the application of OOP principles including possible inheritance. Only provide backend classes NOT user interface or GUI classes. Class design must be thorough; fields and methods must be protected sensibly. Must be reflective of good use of OOP principles where necessary.
   6. Secondary Storage Design - show how data structures in primary memory will be permanently stored. NB: Storage design should be done using tables in a database, text files, JSON files or a combination thereof. For a database, screenshots of tables with record structure and field types from database software are acceptable along with sample data for each table. For text files, an explanation of the structure of the file must be explained together with sample data. Storage can be local, remote or cloud based. Storage design must be well described – fields must be listed, typed and described. Storage design must be appropriate to its purpose and match the Specification Document. There must be no missing elements.
   7. Explanation of Secondary Storage Design - provide an explanation of the secondary storage design. For example, a text file may have been a better solution than a database as the data to be stored is small in value and simple. The explanation must demonstrate a justification of the secondary storage design and an understanding of the implications of the chosen design.
   8. Explanation of Primary Data Structure related to Secondary Storage – provide a description of how the primary data structure described in the class diagrams (assessed in 2.5) will represent the secondary storage design (assessed in 2.6). There should be a description for each backend class listed in 2.5 that will translate the data to and from secondary storage.
2. Coding
   1. Comments - Code is commented using an API and/or comments which are placed inside source code to explain code, parameters and return types. Only backend classes need to have APIs, however all code needs to be commented. Comments need not be provided if descriptive names are used for methods, fields and variables.
   2. Separation of UI from Working Code - different classes are separated in the backend from the UI and other interfaces. The backend can be 'plugged into' a different UI that uses all the methods appropriately.
   3. Inter-Code Communication: Typed Methods and Parameters - Inter-code communication must occur between classes and within a class between methods. There should be effective and conceptually correct use of parameters and typed methods/functions.
   4. Good General Techniques - code required is present and is technically perfect. Indentation is immaculate. Variable names are all descriptive and follow conventions. Programming structures are appropriate e.g. switch instead of if statements, duplication of code is eliminated using appropriate structures such as arrays or methods.
   5. Querying and Manipulation of Data in Secondary Storage - this section refers to the implementation of the primary data structure (section 2.8), the secondary data storage (section 2.6) and the representation of the secondary storage in the primary data structure. Primary data structure is Implemented fully as described. Secondary storage is implemented fully as described. Secondary storage representation is implemented fully as described previously
   6. Defensive Programming - data validation, exception handling, error messages– consider all interfacing elements (e.g. database elements), not just user interface elements. All appropriate data is controlled and validated using code and appropriate interface elements with relevant exception handling. All error messages must be descriptive and easy to understand.
   7. Fulfilment of Specifications - the project will be tested against the functions listed previously Specifications of Program Function. This can only be assessed by running the compiled program. Also, cross reference this with the functional testing done later. All specifications must be complete and working 100%. N.B. there should be no new unspecified functions.
   8. User Experience – will be assessed by running the compiled program. Should be easy to use, completely easy to understand and to navigate. Must provide a wonderful user experience.
   9. Technical Document
      1. A summary of the contents of the document.
      2. Externally Sourced Code – provide references to any code that you used that was not your own. NB: This must be present even if you only declare that no external code has been used. No more than 20% of the code may be from an external source.
      3. Explanation of Critical Algorithms i.e. algorithms that have a significant effect on the

performance of the program – explain the core algorithms that are critical to the correct functioning of the program. There may be only a few, or even only one critical algorithm. Programming code is NOT acceptable. Each algorithm must be included and the function of the algorithm explained.

* + 1. Advanced Techniques - Advanced techniques include a minimum of TWO code/algorithm/features that are NOT part of the syllabus. List and explain how they have been used. An advanced technique is anything that falls outside of what has been taught in class. Inheritance and complex SQL with more than three tables may be included. NB: This must be present even if you only declare that no advanced techniques have been used.
  1. Testing document
     1. Evaluation of Solution against original problem description – provide an objective report on how the solution meets the original problem. You should include suggestions on how to address failure if goals are not met. Address all shortfalls and discuss all possible suggestions. Should the project satisfy all goals, then possible alternate solutions or improvements must be described.
     2. Functional Testing - At least TWO sets of functional testing evident together with the tester’s name, the date the testing was performed and the result of each functional test. Each test should indicate whether the project satisfies the functions listed previously in Specifications of Program Function. It is acceptable if some of the functions are not working as long as progression is seen through the testing process.
     3. Test Plan and Result – provide a test plan and results for TWO input variables. The TWO input variables must be clearly identified. Testing should be done using standard, extreme and abnormal data. Screenshots showing before and after of each test for each variable must be included.

1. The program – important points
   1. Your project should not depend on specialised equipment. It is important that the

programs run on computers available at school.

* 1. Comments: The program should be sensibly and consistently commented i.e.

complicated sections of code need explanation and each method

should have a brief description directly following or preceding the

method name. Eye-catching use of asterisks or dashes enhances

the readability of the code. It is wise to build comments into the program

during development, and not leave it until the end.

* 1. Structure: Structure is the division of a program into modules/classes, with

parameters, that each perform a single task. It is also the use of control

structures such as loops, if and switch..case statements. The program must be well

structured so that its logic is easy to follow. Good structure comes from

good planning.

* 1. User-friendliness: The program should be as easy and pleasant to use as

possible. Example : A built-in Help system; Use of Hints for input etc.

* 1. Output: Keep the output simple. Cluttered screens with many graphics

and/or too much text are off-putting. The output of data should use

appropriate display methods and should be clearly and logically arranged.

1. Other important points:

* There must be no duplication of code unless this is unavoidable.
* The most efficient solution should be used.
* On-line Help that is provided should be contextual and general.
* Screens should be well designed and clear. The design should be

consistent.

* The method of input of data should ensure that as much automatic

input as possible is used, and that this input is validated where possible.

* When using data files include a reasonable amount of data that is all valid.
* Include comments for API documentation that explain the use of each

Class.

The final due date is **31 July 2021**. You are required to meet each deadline as per the project schedule.

Note: Late submissions will be penalised with a 5% deduction for each day late.

**To ensure a good product at the conclusion of the project it should be completed**

**(and will be assessed) in 5 phases.**

Your project will be marked in 5 phases.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Phase** | | **Description** | **Marks** | **Date to be submitted** |
| **What must be submitted by the candidate:** | 1. Project Specifications | Specification of the problem, user interface, data storage and hardware requirements. List (and describe) the functions that your program needs to achieve in order to be a ‘success’ | 17 | 23/02/2021 |
| 1. Design Document | Design the user interface, sequencing (data flow), class and persistent storage of the program in detail. | 30 | 24/03/2021 |
| 1. Coding | Write the program following good programming techniques. | 38 | 01/06/2021 |
| 4.1. Technical Document | Document the project by printing the code & explaining critical algorithms. | 8 | 21/07/2021 |
| 4.2. Testing Document | Document what is to be tested, the test data used and the results of the testing. | 7 | 23/07/2021 |

**TEMPLATES** that show the type of content required in the ***Project Specifications***, ***Design***, ***Technical*** and ***Testing*** documents **have been PROVIDED**. **You must** **use** these templates to produce your documentation for each of the above phases.

**PERFORMANCE ASSESSMENT TASK – Example Mark Sheet**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1 SPECIFICATION DOCUMENT** | | | | | | | | | | | | **17 MARKS** |  |
| **1.1 Problem Summary**  A brief description of the project including the purpose of the project, summary of functions and description of target user group(s) | | | | | | | | | | | **3 MARKS** | |  |
| [0] – No summary or completely inadequate. | [1] – Only one well described item or part of each is not fully complete. | | | | [2] – Most of the items well described but at least one is not completely specified. | | | | [3] – All items well-described and every item completely specified. | | | | |
| **1.2 Motivation and Research**  A brief discussion of similar projects as a result of research. Research is correctly referenced and cited. A motivation is included for the project indicating how this project differs from existing projects. This section should NOT be longer than a page. | | | | | | | | | | | | **3 MARKS** |  |
| [0] – No motivation or research provided. | [1-2] – Similar projects are described, cited and referenced. A motivation is supplied to include an explanation of how the project will differ from existing projects. Deduct one mark for each incomplete aspect of research, citing, referencing and insufficient motivation. | | | | | | [3] – Similar projects are described, cited and referenced. A motivation is supplied to include an explanation of how the project will differ from previous projects. | | | | | | |
| **1.3 Specification of Program Function** | | | | | | | | | | | **3 MARKS** | |  |
| [0] – No functions listed | [1] – The function list is four or less points. The project is too simplistic. | | | | [2] – The function list is a substantial list of appropriate outcomes but is insufficient in complexity or the functions list is not complete. | | | | | [3] – Function list is complete and detailed. The project has sufficient complexity (see taxonomy 2.6.2). | | | |
| **1.4 Specification of Interface (including User Interface)** Specify the user interface and if relevant, the input/output from external hardware such as sensors. | | | | | | | | | | | **2 MARKS** | |  |
| [0] –Interface not specified or incorrectly specified. | | | | [1] – One or two items are inadequately specified. | | | | [2] – Interface completely specified. | | | | | |
| **1.5 Specification of Help** | | | | | | | | | | | **2 MARKS** | |  |
| [0] – Help not discussed. | | | [1] – Help partially discussed with omissions and/or errors. Help is not appropriate for the type of project (e.g. printed manual is not required for mobile applications). | | | [2] – Help completely described and a variety of help is described. Storage of help related data has been specified. Help provision appropriate for application type. | | | | | | | |
| **1.6 Specification of Permanent Data Storage** | | | | | | | | | | | **2 MARKS** | |  |
| [0] – No information given. | | [1] – Some data fields are described with a few errors or data is not appropriately grouped (i.e. related fields should be listed together). | | | | [2] – All data fields have been correctly described and grouped appropriately. | | | | | | | |
| **1.7 Hardware and Software Requirements (including additional hardware)** | | | | | | | | | | | **2 MARKS** | |  |
| [0] – Hardware and software not discussed. | | [1] – Hardware and software is incomplete for development (IDE software, secondary software such as database software, SQL connectors) or missing details of required software or versions missing. | | | | [2] – Hardware and software list is complete including development (IDE software, secondary software such as database software, SQL connectors) and includes software versions. | | | | | | | |

|  |  |  |
| --- | --- | --- |
| **2 DESIGN DOCUMENT** | **30 MARKS** |  |

Taking the template as an example, this document should be around 14 -20 pages (including title page & table of contents). Its main task is to detail the actual design elements of the program, namely:

* User interface design (what the screens look like & what happens on them)
* Program flow (how the program works – linked to the interface)
* Class design (what the classes are, their fields & methods)
* Database/ Storage design (what the persistent storage structure is)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **2.1 Interface Design (See 10.2.5, 10.4.12, 11.4.12)**  ALL interfaces linking the program to the outside environment including the user (barring the program's own data storage – this is dealt with in section 2.4 of this mark sheet) must be completely specified in this section.  Data source and description (e.g. keystroke from user, joystick movement from user, server providing other players' data, temperature gauge providing temperature)  NB: Screen capture from a design tool (including IDE), sketches and mock-ups are acceptable.  No code is needed for this section. | | | | | | | | | | | | **6 MARKS** | | | | |  | |
| User Interface [0-2]  No user facing interface present or the majority of the interface listed is not user facing.  Other Interfaces (where relevant)  No interface design provided or interface description/design is superficial. | | | User Interface [3-4]  Sufficient level of user interface design present but no consideration has been given to good design principles for an effective user interface.  Other Interfaces (where relevant)  Data source and description partially discussed. Specifications are incomplete. | | | | | | User Interface [5-6]  Sufficient level of user interface design present. Evidence of good design principles for an effective user interface.  Other Interfaces (where relevant)  All interfaces (data source and description) completely described. | | | | | | | | | |
| **2.2 Program Flow (See 10.4.1, 11.4.1, 12.4.1, 10.4.8, 11.4.8, 12.4.8))**  Describe the flow of events in the program using either flow charts or pseudocode to show how the program works internally in relation to the interface(s). | | | | | | | | | | | | | | **5 MARKS** | |  | | |
| [0-1] – No sequencing evident or sequencing is superficial with little detail and large logical gaps. | | [2-3] – Sequencing is substantial but still has logical gaps with areas lacking in detail. | | | | | [4-5] – Sequencing covers all aspects of the functions and features listed in the Specification Document. Flow is clear, well represented and easy to understand. No logical gaps are evident. | | | | | | | | | | | |
| **2.3 Class Design and OOP Principles (see 11.4.3, 12.4.3, 11.4.5, 12.4.5)**  The candidates must provide their class design represented as a UML class diagram with class name, fields, and methods demonstrating the application of OOP principles including possible inheritance. Only provide backend classes NOT user interface or GUI classes. | | | | | | | | | | | | | | **8 MARKS** | |  | | |
| Class Design  [0] – No class design or class design is incorrect or is rudimentary with little detail. Fields are incomplete, methods are minimal/not well thought out/not well described.  OOP Principles  [0-1] – No attempt to separate into classes. Some attempt at a class diagram but there is no organisation. | | | | | Class Design  [1-2]– Class design is substantial but shows obvious gaps in missing fields/methods or has minor errors. Method descriptions are more thorough but with missing elements.  OOP Principles  [2-3] – Fields and methods are separated logically into classes. Some instances of incorrect or inappropriate use of OOP principles. | | | Class Design  [3] – Class design is thorough – all fields and methods are present and correctly described. Private methods are present. Methods and fields clearly relate back to the Specification Document.  OOP Principles  [4-5] – Fields and methods are protected sensibly. Good use of OOP principles where necessary. | | | | | | | | | | |
| **2.4 Secondary Storage Design (see Appendix G – 11.4.7, 12.4.7, 10.4.10, 12.4.10, 10.4.11, 11.4.11, 12.4.11)**  Candidate must show how data structure in primary memory described in 2.3 will be permanently stored.  NB: Storage design should be done using tables in a database, text files, JSON files or a combination thereof. For a database, screenshots of tables with record structure and field types from database software are acceptable along with sample data for each table. For text files, an explanation of the structure of the file must be explained together with sample data. Storage can be local, remote or cloud based. | | | | | | | | | | | | | **5 MARKS** | |  | | |
| [0-1] – No design evident/storage design is rudimentary or superficial (e.g. 'uses a database'). | [1-2] – Storage design is evident, but descriptions are, however, superficial/vague/incomplete or with errors. | | | | | [3-4] – Storage design is well described but with missing elements | | | | | [5] – Storage design is well described – fields are listed, typed and described. Storage design is appropriate to purpose and matches the Specification Document. There are no missing elements. | | | | | | |
| **2.5 Explanation of Secondary Storage Design (See Appendix G - 12.4.7, 12.4.10)**  The candidate must provide an explanation of their secondary storage design. For example, a text file may have been a better solution than a database as the data to be stored is small in value and simple. The explanation must demonstrate a justification of the secondary storage design and an understanding of the implications of the chosen design. | | | | | | | | | | | | | **3 MARKS** | |  | | |
| [0-1] – No explanation of secondary storage design is provided or no evidence of understanding of the storage design | | | | [2] – Explanation is substantial, but it is not completely justified. There are some areas of confusion or lack of understanding of the implications of the storage design. | | | | | | [3] – Explanation shows in-depth understanding of the implications of the secondary storage design and is completely justified. | | | | | | | |
| **2.6 Explanation of Primary Data Structure related to Secondary Storage (See Appendix G – 10.4.3, 11.4.3, 12.4.3, 11.4.7, 12.4.7)**  Description of how the primary data structure described in class diagrams (assessed in 2.3) will represent the secondary storage design (assessed in 2.4). There should be a description for each backend class listed in 2.3 that will translate the data to and from secondary storage. | | | | | | | | | | | | | **3 MARKS** | |  | | |
| [0] – No specification on how this secondary storage will be represented in primary memory. | | | | [1-2] – Some form of description present but there are missing elements (e.g. which class relates to which secondary storage data, how the data will be represented, when the data is read from or written to secondary storage). | | | | | | [3] – All clear, detailed representation of secondary storage in primary memory is provided. | | | | | | | |

|  |  |  |
| --- | --- | --- |
| **3 CODING** | **38 MARKS** |  |

Taking the template as an example, this document can be anything from 10 – 100+ pages depending on the complexity and extent of the code. Emphasis must be placed upon:

|  |  |  |  |
| --- | --- | --- | --- |
| Comments for all the methods (these can be copied & pasted from the Design Document) | Communication using typed methods (functions) and parameters | Defensive programming | Good use of persistent storage |
| Separation of UI from working code | Good general programming techniques (naming, indentation, appropriate data structures, etc) | Fulfilment of Design Specifications | User Experience |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CODING NB: This is assessed by examining the source code. The project must be run to determine whether it achieves the specifications listed** | | | | | | | | | | | | | |
| **3.1 Comments** Code is commented using an API and/or comments which are placed inside source code to explain code, parameters and return types. Only backend classes need to have APIs, however all code needs to be commented. Comments need not be provided if descriptive names are used for methods, fields and variables. | | | | | | | | | | | **4 MARKS** |  | |
| [0] – Most of the required code is not submitted and/or there are no comments. | | [1-2] – Not all required code is submitted but there are some with comments. Not all methods are commented. Comments are brief and contain little relevant detail. Parameters and return types are not all commented. | | | | | | [3-4] – Majority of the required code is present with comments. All methods have comments describing what they do. Comments include the data they return (for typed methods/functions) and the data they receive (parameters). Steps in complex algorithms are commented. | | | | | |
| **3.2 Separation of UI from Working Code** | | | | | | | | | | | **5 MARKS** |  | |
| [0] – Most of the required code is not submitted and/or no separation with all code in the interface class/unit. | | [1-3] – Some separation. There are separate classes/units but work is still done in the UI. Insufficient further breakdown and separation in the backend. This includes SQL statements for database centric programs. | | | | | | [4-5] – Majority of the code required is present and there is complete separation. Different classes are separated in the backend from the UI and other interfaces. The backend can be 'plugged into' a different UI that uses all the methods appropriately. | | | | | |
| **3.3 Inter-Code Communication Typed Methods/Functions and Parameters**  Inter-code communication occurs between classes and within a class between methods. | | | | | | | | | | | **5 MARKS** |  | |
| [0] – No inter code communication, no typed methods/functions or parameters. | | [1-3] – Some use of parameters/typed methods/functions. Marks can be deducted (-1 per error type – multiple instances of the same error do not accumulate deductions).  Errors include:  unnecessary use of parameters, incorrect parameters types, parameters specified but not used, incorrect typed method/functions types, failing to return values in typed methods/functions, failing to use the results returned by typed methods/functions, using variables/fields where the value is best returned by a typed method/function. | | | | | | [4-5] – Majority of the code required is present and there is effective and conceptually correct use of parameters and typed methods/functions. | | | | | |
| **3.4 Good General Techniques** | | | | | | | | | | | **5 MARKS** |  | |
| [0] – No techniques. | | [1-4] – Errors in techniques (-1 per error type – multiple instances of the same error do not accumulate deductions).  Errors include:  No indentation, single level indentation, inconsistent or inaccurate indentation, variable names do not clearly indicate what the variable is used for, multiple variables used instead of arrays, multiple if statements instead of switches or case statements, repetition of code (instead of using a typed method/function/void method/procedure arrays). | | | | | | [5] – Majority of the code required is present and is technically perfect. Indentation immaculate. Variable names are all descriptive and follow conventions. Programming structures are appropriate e.g. switch instead of if statements, duplication of code is eliminated using appropriate structures such as arrays or methods. | | | | | |
| **3.5 Querying and Manipulation of Data in Secondary Storage**  This section refers to the implementation of the primary data structure (section 2.3), the secondary data storage(section 2.4) and the representation of the secondary storage in the primary data structure (section 2.6). | | | | | | | | | **6 MARKS** |  | | | |
| Primary data structure implementation *evaluate against what was specified in 2.3*  [0] – No implementation of specification.  Secondary storage implementation *Evaluate against what was specified in 2.4*  [0] – No implementation of specification.  Implementation of secondary storage representation *evaluate against what was specified in 2.6*  [0] – No implementation of specification. | | | | Primary data structure implementation *evaluate against what was specified in 2.3*  [1] – Implemented but not fully as described.  Secondary storage implementation *Evaluate against what was specified in 2.4*  [1] – Implemented but not fully as described.  Implementation of secondary storage representation *evaluate against what was specified in 2.6*  [1] – Implemented but not fully as described. | | | Primary data structure implementation *evaluate against what was specified in 2.3*  [2] – Implemented fully as described.  Secondary storage implementation *Evaluate against what was specified in 2.4*  [2] – Implemented fully as described.  Implementation of secondary storage representation *evaluate against what was specified in 2.6*  [2] – Implemented fully as described. | | | | | | |
| **3.6 Defensive Programming**  Data validation, exception handling, error messages– consider all interfacing elements, not just user interface elements. | | | | | | | | | **4 MARKS** |  | | | |
| [0] – No data validation. | | | [1-2] – Superficial data validation/error trapping. Only focussed on limited areas of code (such as file handling). Interface elements/components are poorly selected. Important validation checks not correctly implemented. | | [3] – Aspects are complete – potential major IO errors protected with exception handling, interface elements used, potential maths errors trapped. There are, however, a few areas where the candidate has not implemented defensive programming. Vague or insufficient error messages. | | | | [4] – All appropriate data is controlled and validated using code and appropriate interface elements with relevant exception handling. All error messages are descriptive and easy to understand. | | | | |
| **3.7 Fulfilment of Specifications**  The project must be tested against the functions listed in Specifications of Program Function.  NB: This can only be assessed by running the compiled program. Also, cross reference this with the functional testing done. | | | | | | | | | **5 MARKS** |  | | |
| [0] – Not achieved or project does not compile. | | | [1-3] – Basic implementation of specifications. Missing functions or significant number of functions do not work as specified. | | | [4] – 90% of specification achieved. but do not all work correctly or almost all functions are there but those that are there work 100%. One or more new functions introduced without specification. | | | [5] – All specifications complete and working 100% and no new unspecified functions. | | | |
| **3.8 User Experience**  NB: This can only be assessed by running the compiled program. | | | | | | | | | **4 MARKS** |  | | |
| [0] – Program does not execute or there is no user facing interface. | [1] – The user is lost – does not know where to start or how to achieve anything when using the program. or user interface present but not at a sufficient level | | | [2-3] – Sufficient level of user interface present, most of which provides a good user experience. Navigating to some screens/functions is unnecessarily complex. Any aspect of the design/ interaction is confusing or unsatisfying. | | [4] – Easy to use, completely easy to understand and to navigate: a wonderful user experience. | | | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **4.1 TECHNICAL DOCUMENTATION** | | **8 MARKS** | | |
| **4.1.1 Externally Sourced Code**  NB: This must be present even if the candidate only declares that no external code has been used. No more than 20% of the code may be from an external source. | | **1 MARK** | | |
| [0] – Not Present | [1] – Candidate has declared used code. Confirm this with interview incorporating oral review of code and techniques. | | | |
| **4.1.2 Explanation of Critical Algorithms**  NB: The core algorithms that are critical to the correct functioning of the program. There may be only a few, or even only one critical algorithm. Programming code is NOT acceptable. Each algorithm must be included and the function of the algorithm explained. | | | **3 MARKS** |  |
| [0] – Not present | [1–2] – Algorithm(s) present but with errors or is incorrect. | [3] – A good, clear description of why these algorithms are critical. Correct flowchart/pseudocode. | | |
| **4.1.3 Advanced Techniques**  Advanced techniques include a minimum of TWO code/algorithm/features that is NOT part of the syllabus. Inheritance and complex SQL with more than three tables may be included.  NB: This must be present even if the candidate only declares that no advanced techniques have been used. | | | **4 MARKS** |  |
| [0] – Not Present | [1–2] – Superficial, candidate has listed techniques in the syllabus or techniques not clearly explained. Only one significantly complex advanced technique is included. | [3–4] – A good explanation of at least TWO features not in the syllabus. | | |
| **4.2 TESTING DOCUMENTATION** | | | **7 MARKS** |  |
| **4.2.1 Evaluation of Solution – against original problem description:**  An objective report on how the solution meets original problem. Should include suggestions on how to address failure if goals are not met. Do not penalise if goals are not met. Award marks for evaluating solution and presenting suggestions for failed solution. Should the project satisfy all goals, then possible alternate solutions or improvements must be described. | | | **2 MARKs** |  |
| [0] – No evaluation, or original problem description was insufficient (or not present). | [1] – Some shortfalls discussed but incomplete, or suggestions are insufficient. | [2] – All shortfalls completely addressed and/or suggestions completely discussed. | | |
| **4.2.2 Functional Testing:**  At least TWO sets of functional testing evident together with the tester’s name, the date the testing was performed and the result of each functional test. Each test should indicate whether the project satisfies the functions listed in Specifications of Program Function. It is acceptable if some of the functions are not working as long as progression is seen through the testing process. | | | **3 MARKS** |  |
| [0] – Not testing, not indicated or original function list was insufficient (or not present) | [1–2] – Not all requirements were tested and/or not sufficiently described: missing details such as when, with whom and result) | [3] – TWO sets of functional tests and all requirements tested with all details present. | | |
| **4.2.3 Test Plan and Result for TWO input variables**  The TWO input variables must be clearly identified. Testing should be done using standard, extreme and abnormal data. Screenshots showing before and after of each test for each variable must be included | | | **2 MARKS** |  |
| [0] – No test plan and no result present | [1] - Some test plan and result present but at least one element is missing. | [2] – Full test plan and results present for TWO input variables which are clearly identified together with screenshots. | | |
| **TOTAL** | | | **100 MARKS** |  |

**All tasks must be handed in on or before the due dates stipulated. Under no circumstances will any extensions be granted.**