Internal assessment: program dossier (35 marks) 35%

The program dossier is an individual piece of work completed during the course. The dossier must address a **single problem** that can be solved using computer systems and which has an **identified end-user**. The analysis, design and production of the final system must be **well documented**.

The emphasis is on the use of a logical approach and analytical thinking from definition and decomposition of the problem through to its solution by constructing appropriate classes implementing algorithms and data structures in the **Java programming language**.

The program dossier is internally assessed by the teacher and externally moderated by the IBO following procedures provided in the *Vade Mecum*.

Time allocation

Standard level

It is expected that approximately 25 hours' teacher contact time will be devoted to the program dossier, including guidance on format, presentation and content. Some of the time teaching the syllabus content will also involve work connected with the program dossier, but this does not include the time required by students to work on their own to develop and complete their dossiers.

Higher level

It is expected that approximately 35 hours' teacher contact time will be devoted to the program dossier, including guidance on format, presentation and content. Some of the time teaching the syllabus content will also involve work connected with the program dossier, but this does not include the time required by students to work on their own to develop and complete their dossiers.

Choice of problem

The role of the teacher is crucial in advising the student in their choice of problem. Overly ambitious problems should be avoided as should overly simplistic ones.

Students are free to choose problems generated by themselves or their teacher, but the problem chosen must have an identified end-user. Students may share the same problem to be solved or the same initial scenario, but **collaborative work is forbidden.**

Teachers are expected to give educational guidance at each stage of the design process. In particular the prototype should be fully explored by the teacher and student to ensure that the requirements of the user can be met within the programming abilities of the student and within the time available. If this is not the case then another problem should be chosen or a restricted solution should be offered to the end-user.

The scope of mastery aspects available in the problem needs to be considered. The level of difficulty of the problem should be consistent with the ability of the student.

Approach

The internal assessment criteria imply that the work on the dossier falls into four main stages:

A—Analysis

B—Detailed design

C—The program

D—Documentation.

Preferably students will complete these stages in the order given. However, with stages B and C it may occasionally be necessary for students to return from C to B one or more times to refine their detailed design in a "spiral" of design and development. This will also depend on the nature of the problem (open or closed) and on the ability of the student. Teachers should not allow students to produce stages A and B following development of the solution.

Teachers are advised to set deadlines for the ends of stages A, B, C and D so that students are helped to achieve success.

Students are free to choose a design methodology (structured, top-down or object-oriented) that is flexible and extensible. Therefore, it may be necessary for them to retain design documentation from earlier stages to present as evidence in the final dossier in support of awards for criteria B1–B3. Teachers may wish to design their own methods of collecting such design documentation (design logs, portfolios containing CRC cards, UML style diagrams and so on). Examples will be provided with teacher support material for this course.

When the student's program is complete, the teacher should run it in the presence of the student to confirm that it functions, and has produced the hard copy output submitted with the program dossier.

Automated development systems

Some programming systems, such as visual IDEs, provide interactive development environments with a wide range of extra facilities, such as visual design, object manipulation, and automatic code generation. However, the use of these is beyond the scope of this syllabus.

Within the program dossier such facilities may be used by the student, but must not be used for mastery tasks. For example, SL students would be expected to write their own algorithms for sorting an array, rather than simply executing a library function that sorts the array. Similarly, HL students would be expected to write their own algorithms that maintain a linked data structure, rather than using a system library that already contains all the required algorithms.

Any program listing that includes code automatically generated by the development system must have this code clearly identified and distinguishable from the code written by the student.

Teacher assessment

Teachers assess students' performance by using level descriptors against the relevant criteria, which are related to the objectives. The criteria and achievement levels must be applied to the work in the program dossier **regardless** of the number of aspects in which mastery is demonstrated. After this a "mastery factor" is applied. This factor depends on the number of different aspects in which mastery is demonstrated. (See the section on mastery in this guide). The assessment of the program dossier is moderated externally.

Only the code **designed** and **written** by the student must be taken into account when applying the assessment criteria, and awarding marks.

If teachers add comments to dossiers as well as marking them ready for moderation, this facilitates the moderation process. In addition, if teachers write a report for each student that justifies the achievement level awarded for each criterion, this also will facilitate the moderation process and make the feedback forms from the moderator more focused.

Format of the dossier

All the student's work must be submitted together as a single document. The work can be stapled, put into a ring-binder or inserted into a folder. All information required for the program dossier must appear as hard copy. Diskettes, CD-Roms, and so on must **not** be included within the program dossier or sent to the moderator.

There must be a table of contents and all written documentation should be word processed, except where it is felt necessary to include rough notes. Program runs and sample screens may be annotated by hand.

All the pages must be numbered. The numbering can be sequential (1, 2, 3, and so on) throughout the entire program dossier or it can be done according to the items numbered in the following table. (For example, if the design process is the third item, then these pages can be numbered 3–1, 3–2, 3–3, and so on.) This may be easier, since each item can be numbered sequentially as it is completed. The page numbering can be done by hand if the available computer systems do not permit automatic page numbering.

The number of pages associated with each item may vary according to the nature and complexity of the problem being solved as well as its programmed solution. However, as a guide, an approximate number of pages is given in the following table. This is included mainly to ensure that students include the appropriate amount of material.

Items to be included in the program dossier

All of the items listed in the following table must be included in the program dossier.

Items to be included in the program dossier	Suggested number of pages
Table of contents	
Analysis of the problem	2–3
Criteria for success	1–2
Prototype solution	Variable
Data structures	2–5
Algorithms	2–5
Modular organization	3–5
Handling errors	1–2
Code listing	Variable (500–3,000 lines)
Annotated hard copy	Variable
Evaluation of solutions	2
Documentation of mastery aspects	2
Total	Approx 60–100

Internal assessment criteria

Using the assessment criteria and descriptors

The method of assessment used by the IBO is criterion related. That is to say each student is assessed against identified assessment criteria and not against other students.

- There are **twelve** assessment criteria for the program dossier. For each assessment criterion, achievement level descriptors are defined that concentrate on positive achievement, although for the lower levels (1 = the lowest level of achievement) failure to achieve may be included in the description.
- The aim is to find, for each criterion, the descriptor that conveys most adequately the achievement level attained by the student.
- Having scrutinized the work to be assessed, read the descriptors for each criterion, starting with level 1, until you reach one that describes a level of achievement that the work being assessed has **not** reached. The work is therefore best described by the preceding achievement level descriptor and you should record this level.
- Use only whole numbers, not partial marks such as fractions and decimals. If a student does not achieve a standard described by any of the descriptors, then 0 (zero) should be recorded.
- The highest descriptors do not imply faultless performance and teachers should not hesitate to use the extremes, including zero, if they are appropriate descriptions of the work being assessed.
- Descriptors should not be considered as marks or percentages, although the descriptor levels are ultimately added together to obtain a score out of 35. It should not be assumed that there are other arithmetical relationships; for example, a level 4 performance is not necessarily twice as good as a level 2 performance.
- A student who attains a particular level of achievement in relation to one criterion will not necessarily attain similar levels of achievement in relation to the others. Do not assume that the overall assessment of the students will produce any particular distribution of scores.

Stage A—Analysis

Criterion A1: Analysing the problem

The documentation should be completed first and contain a thorough discussion of the problem that is being solved. This should concentrate on the **problem** and the goals that are being set, not on the method of solution. A good analysis includes information such as sample data, information and requests from the **identified end-user**, and possibly some background of how the problem has been solved in the past.

0	The student has not reached a standard described by any of the descriptors given below. For example, the student has simply described the programmed solution.
1	The student only **states** the problem to be solved **or shows** some evidence that relevant information has been collected.
2	The student **describes** the problem to be solved.
3	The student describes the problem **and provides evidence** that information relating to the problem has been collected.

This section of the program dossier would typically be two to three pages in length. It should include a brief statement of the problem as seen by the end-user. A discussion of the problem from the end-user's point of view should take place, including the user's needs, required input and required output. For example, evidence could be sample data, interviews and so on, and could be placed in an appendix.

Criterion A2: Criteria for success

This section of the program dossier will clearly state the objectives/goals of the solution to the problem.

0	The student has not reached a standard described by any of the descriptors given below.
1	The student **states some** objectives of the solution.
2	The student **describes most of** the objectives of the solution.
3	The student **relates all of** the objectives of the solution to the analysis of the problem.

This section of the program dossier would typically be one to two pages in length. Objectives should include minimum performance and usability. These criteria for success will be referred to in subsequent criteria, for example criterion C3 (Success of the program) and D2 (Evaluating solutions).

Criterion A3: Prototype solution

The prototype solution **must** be preceded by an initial design for some of the main objectives that were determined to be the criteria for success. A prototype of the solution should be created.

A prototype is: "The construction of a simple version of the solution that is used as part of the design process to demonstrate how the system will work."

0	The student has not reached a standard described by any of the descriptors given below.
1	The student includes an initial design **and a prototype**, but they do not correspond.
2	The student includes an initial design and a prototype that **corresponds**.
3	The student includes an initial design and a complete prototype that corresponds to it **and documents user feedback** in evaluating the prototype.

The prototype need not be functional, it could be constructed using a number of tools such as: Visual Basic, PowerPoint, Mac Paint, Corel Draw for a simple Java program. The intent is to show the user how the system is expected to operate, what inputs are required and what outputs will be produced. A number of screenshots will be required for the user to be able to evaluate the solution properly. The prototype, at its simplest, could be a series of clear, computer-generated drawings, a hierarchical outline of features in text mode, or a series of screenshots.

Documentation of user feedback could be, for example, a report of the user's comments on the prototype.

Stage B—Detailed design

The ordering of the criteria B1–B3 does not imply that this is the sequence in which students should develop or document their designs. This will vary according to the methodology adopted.

Criterion B1: Data structures

Students should choose data structures, **at the design stage**, that fully support the data-storage requirements of the problem, and that allow clear, efficient algorithms to be written. The data structures must fully support the objectives of the solution (criterion A2). The classes chosen should be logical in that the data is sensible for the objects in question and the methods are appropriate for the data given. This section of the program dossier could include class definitions, file structures, abstract data types (particularly at higher level) and some consideration of alternatives.

0	The student has not reached a standard described by any of the descriptors given below.
1	The student has **outlined some** of the data structures/types to be used in the solution.
2	The student has **described some** of the data structures/types to be used, and provided sample data.
3	The student has **discussed** and **clearly illustrated all** of the data structures/types to be used to solve the problem, and provided sample data for **all** of them.

This section would typically be two to five pages in length.

Data structures and data members that are to be used in the programmed solution should be discussed here. Sample data, sketches/illustrations, including discussion of the way data objects will be changed during program execution should be demonstrated to achieve a level 3 in criterion B1.

Criterion B2: Algorithms

Students should choose algorithms, **at the design stage**, that fully support the processes needed to achieve the objectives of the solution (criterion A2), and provide sufficient support for the required data structures. The classes chosen should be logical in that the methods are appropriate for the data given. Students must include parameters and return values.

0	The student has not reached a standard described by any of the descriptors given below.
1	The student has **outlined some** of the algorithms to be used in the solution.
2	The student has **described most** of the algorithms to be used, with details of parameters and return values.

This section would typically be two to five pages in length.

This can be a list or outline of all the algorithms, presented as text, possibly in outline format. Standard algorithms (such as search or sort) can simply be named (with parameters), but non-standard algorithms must be described in more detail.

Criterion B3: Modular organization

Students should choose modules, **at the design stage**, that incorporate the data structures and methods required for the solution (criteria B1 and B2) in a logical way. The data structures must fully support the objectives of the solution (criterion A2). Students must present this organization in a structured way that clearly shows connections between modules (hierarchical decomposition or class dependencies). The connections between modules, algorithms and data structures must also be presented.

0	The student has not reached a standard described by any of the descriptors given below.
1	The student has **outlined some** of the modules to be used in the solution.
2	The student has **described most** of the modules to be used, **showing connections** between them.

This section would typically be three to five pages in length.

A variety of presentations are possible here. Some possibilities are:

- a top-down hierarchical decomposition chart containing the names of modules, showing connections between modules and showing details of which data structures and methods are connected with (or part of) which modules
- a text outline showing hierarchical decomposition (equivalent to above)
- a hard copy of CRC cards showing dependencies between collaborating classes, with details of which data structures and methods are connected with (or part of) which classes.

The design is assessed independently from the programming stage (stage C). The design should be complete, logical and usable, but the student may deviate from it or expand it during stage C, without penalty.

Stage C—The program

Program listings must contain **all** the code written by students and, if a program listing displays code that was automatically generated by the development system or copied from another source, then this code must be clearly identified and distinguishable from that code written by the students. Only the code **designed and written** by students must be taken into account when applying the assessment criteria.

Criterion C1: Using good programming style

Good programming style can be demonstrated by program listings that are easily readable, even by a programmer who has never used the program. These would include small and clearly structured Java methods, sufficient and appropriate comments, meaningful identifier names and a consistent indentation scheme.

0	The student has not reached a standard described by any of the descriptors given below.
1	The program listing demonstrates **some** attention to good programming style.
2	The program listing **mostly** demonstrates attention to good programming style.
3	All parts of the program listing demonstrate **considerable** attention to good programming style.

A typical program should be approximately 1,000–3,000 (HL) or 500–2,000 (SL) lines of code in length.

Comments should be included to describe the purpose and parameters of each method, and also when code is difficult to understand.

The program should demonstrate the use of good programming techniques. It should include:

- an identification header indicating the program name
- author, date, school
- computer used, IDE used, purpose.

The program should possess good internal documentation, including:

- constant, type and variable declarations that should have explanatory comments
- identifiers with meaningful names
- objects that are clearly separated and have comments for their parameters
- suitable indentation that illustrates various programming constructs.

Generally, achievement level 2 will be appropriate where two or more of these have been demonstrated. Then, achievement level 3 will be appropriate for three or more being demonstrated.

Criterion C2: Handling errors

This refers to detecting and rejecting erroneous data input from the user, and preventing common run-time errors caused by calculations and data-file errors. Students are not expected to detect or correct intermittent or fatal hardware errors such as paper-out signals from the printer or damaged disk drives, or to prevent data-loss during a power outage.

0	The student has not reached a standard described by any of the descriptors given below.
1	The student **includes** documentation that shows a **few** error-handling facilities in the program, or documents only **one** type of input or output.
2	The student includes documentation that shows **many** error-handling facilities in the program, and documents **more than one** type of input or output.
3	The student **fully** documents the error-handling of **each** input and output method within the program.

This section would typically be one to two pages in length.

For this criterion, students must attempt to trap as many errors as possible. The documentation in the dossier can take a variety of forms.

For example, students could highlight relevant comments within the program listing or they could produce a table with two columns, one that identifies any error possibilities, and one that shows the steps taken to trap the errors. It is not expected that extra output is produced for this section.

Criterion C3: Success of the program

Evidence here refers to hard copy output in criterion D1.

0	The student has not reached a standard described by any of the descriptors given below.
1	The student **includes** evidence that the program **functions partially**. The student successfully achieved **some** of the objectives from criterion A2.
2	The student includes evidence that the program functions **well**. The student successfully achieved **most** of the objectives from criterion A2.
3	The student includes evidence that the program functions well. The student successfully achieved **all** of the objectives from criterion A2.

The teacher should run the program with the student to confirm that the program functions, and that it produces the hard copy output submitted with the program dossier.

Stage D—Documentation

Criterion D1: Including an annotated hard copy of the test output

The hard copy of test output should demonstrate that the program fulfills the criteria for success in criterion A2. The output **must** be **annotated** (this may be done by hand). The teacher must confirm that each student has actually completed the testing as claimed in the documentation. (See the *Vade Mecum*.)

0	The student has not reached a standard described by any of the descriptors given below.
1	The student includes an **incomplete** set of sample output.
2	The student includes an incomplete set of **annotated** sample output.
3	The student includes a **mostly complete** set of annotated sample output.
4	The student includes a **complete** set of annotated sample output, testing all the objectives in criterion A2.

Hard copy output from one or more sample runs should be included to show that the different branches of the program have been tested; testing one set of valid data will not be sufficient. The hard copy submitted should demonstrate the program's responses to inappropriate or erroneous data, as well as to valid data. Thus the usefulness of the error-handling routines mentioned above should become evident. While at least one complete test run must be included in the dossier, it is not necessary that the hard copy reflect every key stroke of every test run. Cutting and pasting of additional test runs should be done to illustrate the testing of different aspects of the program.

All test runs should be annotated in such a way that the student is stating what aspect of the program is being tested. Sample output must **never** be altered by hand, erased or covered up.

Sample output can be "captured" and combined electronically with explanatory annotations into a single document. However, it is forbidden to alter or reformat sample output in any fashion (except to add page numbers or annotate in order to highlight user friendliness or error-handling facilities as discussed above), especially if these alterations would give an unrealistic impression of the performance of the program. Examples of such "abuse" include: lining up text that was not originally aligned; adding colour or other special effects; changing incorrect numerical output; erasing evidence of errors.

Criterion D2: Evaluating solutions

The evaluation/conclusion section should be a critical analysis of the resulting solution. Effectiveness should be discussed in relation to the original description of the problem and the criteria for success that were stated in criterion A2. Efficiency may be discussed in general terms, for example BigO notation is not required. Suggested improvements and possible extensions should be realistic, for example suggestions should not include statements such as "the program would be a lot better if it incorporated some artificial intelligence techniques such as speech recognition and natural language parsing".

0	The student has not reached a standard described by any of the descriptors given below.
1	The student only **outlines** the solution.
2	The student **outlines** the solution and **partly** considers effectiveness, efficiency and possible improvements.
3	The student **discusses** the effectiveness and efficiency of the solution **and suggests** alternative processes and improvements.
4	The student **suggests** alternative approaches to the solution and the design process.

This section of the dossier would typically be two pages in length.

The evaluation/conclusion should include reflections on the effectiveness of the programmed solution of the original problem. It should discuss answers to the following questions.

- Did it work?
- Did it address the criteria for success?
- Did it work for some data sets, but not others?
- Does the program in its current form have any limitations?
- What additional features could the program have?
- Was the initial design appropriate?

A thorough evaluation should also discuss possible future enhancements that could be made to the program.

Stage E—Holistic approach

Criterion E: Holistic approach to the dossier

The program dossier should be an ongoing process involving consultation between the student and teacher. The student should be aware of the expectations of the teacher from the beginning of the process and each achievement level awarded should be justified by a written comment from the teacher at the time of marking. The examples given below for each criterion level are teacher-orientated and each teacher should use discretion when judging the levels.

0	The student showed **no** commitment. For example, the student did not participate in class discussions on dossier work, did not submit the required work in progress, and/or missed many deadlines.
1	The student showed **minimal** commitment. For example, the student participated minimally in class discussions on dossier work, kept to most deadlines, had some discussion initiated by the teacher and/or did not exploit the available opportunities for the development or improvement of the dossier.
2	The student showed **good** commitment. For example, the student participated in class discussions on dossier work, initiated discussions with the teacher and/or the rest of the class and/or became fully involved in the development of the dossier.

In order to obtain the highest achievement level for this criterion the student should have excelled in areas such as those listed below. This list is not exhaustive and teachers are encouraged to add their own expectations.

The student:

- actively participated at all stages of the development of the dossier
- demonstrated a full understanding of the concepts associated with his/her dossier
- demonstrated initiative
- demonstrated perseverance
- showed insight
- prepared well to meet deadlines set by the teacher.

MASTERY

Students must demonstrate mastery of various aspects of Java by documenting evidence in their program dossiers.

Mastery aspects

Standard level

To achieve a mastery factor of 1.0, students must have mastered at least **10** of the following 15 aspects.

1. Arrays
2. User-defined objects
3. Objects as data records
4. Simple selection (**if–else**)
5. Complex selection (nested **if, if** with multiple conditions or **switch**)
6. Loops
7. Nested loops
8. User-defined methods
9. User-defined methods with parameters (the parameters have to be useful and used within the method body)
10. User-defined methods with appropriate return values (primitives or objects)
11. Sorting
12. Searching
13. File i/o
14. Use of additional libraries (such as utilities and graphical libraries **not included** in appendix 2 Java Examination Tool Subsets)
15. Use of sentinels or flags

It is anticipated that this list will provide students with the option to choose algorithms and data structures appropriate to the problem rather than contriving a solution to fit the mastery aspects.

Where one aspect includes others, all are credited, for example aspect 10 will also satisfy aspects 8 and 9 (always provided that the use is **non-trivial**, **well-documented** and **appropriate**).

Higher level

To achieve a mastery factor of 1.0, students must have mastered at least **10** of the following 19 aspects.

1. Adding data to an instance of the **RandomAccessFile** class by direct manipulation of the file pointer using the **seek** method
2. Deleting data from an instance of the **RandomAccessFile** class by direct manipulation of the file pointer using the **seek** method. (Data primitives or objects may be shuffled or marked as deleted by use of a flag field. Therefore files may be ordered or unordered).
3. Searching for specified data in a file.
4. Recursion
5. Merging two or more sorted data structures
6. Polymorphism
7. Inheritance
8. Encapsulation
9. Parsing a text file or other data stream
10. Implementing a hierarchical composite data structure. A composite data structure in this definition is a class implementing a record style data structure. A hierarchical composite data structure is one that contains more than one element and at least one of the elements is a composite data structure. Examples are, an array or linked list of records, a record that has one field that is another record, or an array.
11. The use of any five standard level mastery factors—this can be applied only once

12–15. Up to four aspects can be awarded for the implementation of abstract data types (ADTs) according to the table entitled "Implementation of ADTs".

An ADT may be implemented as a class or interface containing data members and methods appropriate to that ADT. The number of mastery aspects to be awarded will depend on the thoroughness and correctness of the student's implementation. Examples are given in the following table.

16. Use of additional libraries (such as utilities and graphical libraries not included in appendix 2 Java Examination Tool Subsets)
17. Inserting data into an ordered sequential file without reading the entire file into RAM.
18. Deleting data from a sequential file without reading the entire file into RAM.
19. Arrays of two or more dimensions.

"Non-trivial" means that the programmer must demonstrate that the program benefits from the use of the aspect.

Where one aspect includes others, all are credited (always provided that the use is **non-trivial, well documented** and **appropriate**).

Implementation of ADTs

ADT name	One aspect	Two aspects	Three aspects	Four aspects
General criteria.	An incomplete ADT is implemented.	An ADT is implemented with all key methods implemented.	An ADT is implemented that includes some error checking.	An ADT is implemented completely and robustly.
Lists, implemented using references (that is, a dynamically linked list).	A node style class with appropriate constructors and methods to set and get data elements.	Methods are implemented to add at/remove from the tail and the head of the list.	Proper checks are made for errors such as attempting to get an element from an empty list or inserting the same element twice.	All error conditions are checked for, and all appropriate methods are implemented. For a doubly linked list these could be: size isEmpty first last before after insertHead insertTail insertAfter insertBefore.
Tree (simple, ordered, binary tree is sufficient using arrays or dynamically linked object instances).	A class or interface with appropriate constructors and methods to set and get data elements.	Methods are implemented to add at/remove from the correct point in the tree.	Proper checks are made for errors such as attempting to get an element from an empty tree or not inserting the same element twice.	All error conditions are checked for, and all appropriate methods are implemented. For a simple, ordered, binary tree these could be: size isEmpty root parent leftChild rightChild.

ADT name	One aspect	Two aspects	Three aspects	Four aspects
Stack implemented dynamically or statically.	A class or interface with appropriate constructors and methods to push and pop items.	Methods to test for full and empty stack are added.	Proper checks are made for errors such as attempting to get an element from an empty stack.	Probable methods: push pop top isEmpty isFull size.
Queue implemented dynamically or statically.	A class or interface with appropriate constructors and methods to enqueue and dequeue items.	Methods to test for full and empty queue are added.	Proper checks are made for errors such as attempting to get an element from an empty queue.	Probable methods: enqueue dequeue front rear isEmpty isFull size.
Hash table implemented in an array.	A class or interface with appropriate constructors and methods to insert and remove items.	Methods to test for full table and duplicate keys are added.	Proper checks are made for errors such as attempting to get a non-existent key, clashes are dealt with properly.	Probable methods: hashFunction insertKey removeKey isDuplicate isEmpty isFull size.

It is not possible to provide an exhaustive list here and teachers will need to exercise some degree of judgment in implementing this mastery aspect. It is considered highly unlikely that teachers will encourage students to develop graphs, heaps, dictionaries, priority queues and ADTs of similar complexity.

"Complete and robust" means that all the needs of the solution are solved without failure.

The mastery factor

Mastery judgments must be made in the same way for both SL and HL. Therefore the criteria should be applied in the same way to the work in both SL and HL program dossiers.

Both SL and HL students are required to demonstrate mastery of **at least 10** aspects. The criteria and level descriptors should first be applied to the work in the dossier **regardless** of the mastery aspects demonstrated.

The appropriate mastery factor should then be determined from the table below. After applying the mastery factor, the student's final score should be rounded to the nearest whole number (0.5 or above rounds up to the next whole number).

Students must also document their dossiers thoroughly. To show mastery of an aspect, it is **not** sufficient for students only to **use** it within a program. In the written documentation, students must include information about **why** a particular data structure is appropriate, **how** it is used (for example, how nodes are added, deleted and searched for) and **where** it is used in the program. In other words, students must provide cross-references between the documentation and specific procedures within the program.

Number of aspects in which the student demonstrates mastery	Mastery factor
10 or more	1.0
9	0.9
8	0.8
7	0.7
6	0.6
5	0.5
4	0.4
3	0.3
0, 1 or 2	0.2

Examples

1. A student achieves 29, as judged by applying the assessment criteria.

   Mastery was demonstrated in eight different aspects.

   So the final mark awarded is $29 \times 0.8 = 23.2 = 23$.
2. A student achieves 32, as judged by applying the assessment criteria.

   Mastery was demonstrated in twelve different aspects.

   So the final mark awarded is $32 \times 1.0 = 32$.

Documentation of mastery aspects

The mastery aspects should be listed with:

- the page number(s) where they occur in the code listing
- a brief description of how their use benefits the solution.