COMPUTER SCIENCE

Overall grade boundaries

Higher level

Grade:	1	2	3	4	5	6	7
Mark range:	0-13	14-26	27-38	39-50	51-63	64-75	76-100

Standard level

Grade:	1	2	3	4	5	6	7
Mark range:	0-15	16-30	31-41	42-52	53-63	64-74	75-100

Higher level internal assessment

Component grade boundaries

Grade:	1	2	3	4	5	6	7
Mark range:	0-5	6-10	11-17	18-25	26-33	34-41	42-50

The range and suitability of the work submitted

As this was the first session of a new set of criteria for the dossier and a new set of mastery aspects, there were inevitably some misunderstandings and misinterpretations. This report will attempt to clarify what is required for a computer science dossier.

It is important for schools to be aware that moderators were given instructions to be generous in their interpretation of many of these criteria and mastery aspects, but that this will not be the case in future sessions.

The range of topics chosen was quite broad, as the new mastery aspects were intended to allow. Games continue to be handled poorly by all but the most able candidates. The best choices were usually those that had a clear, real intended user. For the weaker candidates some kind of database program is generally open-ended enough for them to demonstrate all of their skills and provides the most straightforward choice of problem.

As ever a number of candidates concentrated on the solution and hardly ever discussed the problem they were attempting to solve. Interaction with users often appeared to be token or minimal and this clearly hampered the analysis and design part of the dossier.

While it must be recognised that this is a classroom exercise and the candidates' final programs are extremely unlikely to be used in the real world, the concept is to try to simulate a real-world design process as closely as possible.

It often appeared that candidates produced the analysis and design sections after the program had been written. This is a disadvantage to candidates, as careful attention to analysis and design can actually reduce time and work required for the solution, whereas writing this documentation afterwards is a time-consuming task with no value as a learning experience.

The new mastery aspects related to OOP caused some problems and were approached in a trivial manner in many cases, often relying on library or utility routines that are part of the Java SDK. The relevant sections of the subject guide (in Topic 5) describe the knowledge that candidates at this level are expected to have regarding inheritance, polymorphism and encapsulation. This issue is discussed in depth in later sections of this report.

Other new mastery factors such as parsing a text stream were also widely misunderstood and will be clarified by example.

The implementation of ADTs was adequate or even good in many cases. There were some clear-cut examples of plagiarism and some not so obvious. Candidates must reference any code that is taken directly or indirectly from another source – this will be the usual case for an ADT as it is extremely unlikely that a candidate will write an ADT from scratch. Their sources of information should be included in the design section B1 Data Structures and the teacher should ensure that the candidate has adapted the ADT to their particular problem and understands all the code that is included and is being claimed for a mastery aspect.

Candidates should clearly indicate which code they are using for a mastery claim and which they have borrowed and are including to improve some aspect of their solution. Teachers should use professional judgement and maintain integrity in this area, as the border between adapted and merely borrowed but poorly understood code can be a narrow one.

Sample runs including hard copy output were again a problem in many cases. The Subject Guide clearly states that at least one complete test run of the program should be included. This should involve normal data and, where possible, demonstrate that the program functions as claimed. For example, if a mastery factor of adding to a file or linked list is claimed, there should be a part of the listing that shows the record being added and subsequently included in the data set. Many sample runs only show what happens when abnormal data is entered.

Candidate performance against each criterion

Criterion A1

In many cases the problem was not analysed. The students were keen to start demonstrating what their program will accomplish without any consideration of why it will be doing this. The starting point should be a clear statement of the information problem that is to be solved. This information problem can ONLY be properly understood by talking to the user or users involved in using the current system. Students who start the solution with only a hazy idea of the problem to be solved inevitably do poorly in the dossier.

A real (not imaginary) user is required so that the students can obtain the views of a non-technical person and gather information from them. It is not essential but preferable that this user is a mature adult with a real problem rather than a classmate or sibling with a problem the candidate made up for them. In the former case, the quality of evidence collected is likely to make it more useful to the candidate. That said, not all mature adults are good communicators, nor all young people incoherent, the candidates and teachers should still use their best judgement.

Often evidence of data collection was absent or superficial in the dossiers seen this session.
The user need not have a problem that requires a real-life, all-singing, all-dancing computer program at the professional level – ideally it will be relatively simple. Students should be actively discouraged from attempting to create feature-rich or overly ambitious solutions.

In this context, the real user exists to bring the student out of their own narrow perspective in which they develop only for themselves. There is no reward (in IB terms) for a really first class program or

a program that is actually used afterward. Part of the analysis and design process needs to limit the scope of the solution to an achievable size. Teacher guidance is essential at this stage.

The "creeping featurism" danger (i.e. the temptation to add extra “cool” features during coding) that leads to missing deadlines and experiencing too many programming difficulties can probably be avoided if the students understand that this is a classroom exercise, not a commercial product. Where students insist that they can produce such programs a good teaching strategy is to get them to do the core requirements of an IB dossier first and let them attempt any sophisticated additions at the end (if there is time).

Criterion A2

Many students did not provide a clear connection between the objectives and the problem analysis as is required by the criterion. Some students wrote descriptive paragraphs while others wrote bullet points – the latter probably being more effective. Large numbers of objectives are undesirable for the reasons discussed in A1.

Objectives should be SMART: specific, measurable, achievable, realistic and time-constrained. At this point the teacher and candidate should satisfy themselves that at least 10 (and preferably 12) mastery aspects can be achieved and have a good idea which they will be. A development plan at this stage would help candidates to create a realistic timeframe for completing the work to be done – this need not be included in the dossier but should be referred to frequently during development. Teachers should consider rigorously enforcing deadlines to ensure that candidates gain maximum credit for the work that they have done.

The objectives should relate to the analysis explicitly. Stating as an objective “I am going to organize the CDs” is meaningless unless it is clear what needs to be demonstrated to prove that the objective has been achieved.

Objectives were given but rarely referred to in later sections. In the later sections the objective should be stated, then the evidence that supports the accomplishment of that objective should be presented.

Students should avoid having a large number of objectives as this dramatically increases the need for testing and sample runs without really illustrating additional skills of the candidate but merely adding to bulk. As a rough and ready guide, 5 or 6 serious, attainable, well described, non-trivial objectives ought to be plenty.

Another suggestion is to write **user-centric** goals rather than **program-centric** goals. As the students are trapped in their programmer-centric viewpoint, their objectives are always written about the program.

User-centric goals describe what the user will be able to accomplish, rather than what "features" the program will supply. It should be easier to test user-centric objectives. And it is obvious that testing a user-centered goal requires sample output, rather than reading some lines of a program listing. For example, consider rewriting the goals this way:

Program-oriented	rewritten as	User-oriented
Program saves CD titles in a data file	→	User can save CD titles and retrieve and view them later
Program can sort CD titles alphabetically	→	User can choose to view titles sorted alphabetically or chronologically (date order)

Some dossiers in the current session accepted user data input and "stored" it in an array, but never copied it into a file. The user would definitely want to keep the data from day to day, but the student programmers apparently thought it was okay to get the array mastery mark and ignore the file I/O.

This might make sense in a proto-type, but it is hard to imagine writing any kind of data processing software that loses all its data when the machine is turned off. Again, a real user who has real needs and who is actively consulted should be able to identify these objectives for the candidate. The candidate, as stated earlier, is not obliged to do everything the user wants but should "cherry pick" the 5 or 6 that fit the constraints mentioned above.

Criterion A3

Once the major objectives and boundaries of the proposed solution to the simple problem have been clearly and unambiguously identified, then it should be possible to identify the major elements of the solution as well as its planned appearance.

This leads to an initial modular design for a prototype which can then be shown to the user for comment. The user's comments should be documented but need not always be followed to the letter – particularly if their suggestions would lead to an overly extended solution.

A prototype will help the candidate form a clear picture of the "product" in their own minds.

In this session many candidates did not include an initial design or user feedback on the prototype. Non-functional prototypes created using various graphics packages were often presented in this section and are useful.

It can be beneficial for a non-functional prototype to be written in Java as the candidates will gain insights which are useful for algorithm design and it also satisfies their craving to get on and write some code at an early stage – clearly this should not be taken to extremes. A Java prototype with method stubs for the main functions is a useful starting point for coding later on.

The prototype should be used to break up the program into logical sections from a user-viewpoint. This probably involves choosing menu items and data fields on a form. The initial design can be more program-centric, involving data-structures and algorithms. If the "initial-design" and the "prototype" are approached from these two different viewpoints, it makes sense to ask whether the two connect sensibly (criteria A3).

Criterion B1

At Higher Level, some discussion of alternative data structures is expected, but this can be fairly superficial. It is more important for candidates to explain how their selected data structures will actually work with the data from the problem domain.

As in the previous version of the dossier criteria, illustrations are extremely valuable and required for the top marks. Illustrations must use data from the problem the candidate is attempting to solve; generalized diagrams of the operation of linked lists or trees in the abstract do not satisfy this criterion.

Candidates may use data structures taken from any source as long as the source is acknowledged. The data structure can only be used to claim mastery if it is discussed in this section and usually there will be a need to adapt the structure for the proposed solution.

Students must program their own ADTs (e.g. Linked List). Use of the Java.util LinkedList will not achieve ADT mastery marks. Also, simply copying a standard LinkedList library from another author does not achieve mastery marks, and constitutes plagiarism when not clearly referenced.

At Higher Level the principal problem in this respect is code to implement Linked Lists, however, other copied code for ADTs was also encountered in the session.

Criterion B2

The algorithms should be sufficiently detailed that the solution can be created from them – they do not have to be perfect nor complete – but all of the essential modules identified in B3 or A3 should be present, as it remains a requirement of the IB Computer Science programme.

Many candidates in this session neglected to provide sufficient algorithms at the required level of detail (pre-conditions, post-conditions and parameters) and were penalized for that. Some schools presented algorithms on coding sheets which was a very clear method. Other schools presented algorithms which were close to Java syntax and this is acceptable – if teachers prefer to use a pseudocode of their own choosing or devising that is also good.

As ever, some candidates presented code that was clearly derived from the solution and were penalized accordingly. Moderators find this practice very easy to detect.

Criterion B3

As indicated in the Subject Guide, a range of presentations is acceptable for this session – often dependent upon the design approach used. However, all these variations should show the connections from modules or classes to algorithms and data structures. This section often provides the "big picture" of the proposed solution whether it is completed as a top-down modular design or an Object Oriented Design or indeed a "bottom-up" approach.

A variety of presentations were used here but the most common failing was to forget to identify the connections to data structures and algorithms.

Criterion C1

This was mostly well addressed by the candidates who often used good indentation and clear identifier names. However, in many listings it was hard to find out where Classes began and ended and to determine the relationship between them. This is more difficult when the solution departs from the design, which is permitted.

Ideally a Class should start on a new page or have a clear separation from preceding Classes and should have a header statement describing its function. A list of Classes with page numbers at the start of the code is extremely useful. Some IDEs provide relationship diagrams which are also helpful to those finding their way around the code.

Candidates should ensure comments are readable if printed in black and white; some IDEs have comments in grey which are hard to read.

Criterion C2

This section was generally done well by candidates. Many of them chose to use a summary approach, referring to hard copy printout supplied as part of section D1. This worked better than descriptive text without illustrations or specific examples.

This section should reference the A2 criteria explicitly and this was not done in very many cases. Again, the idea behind the A2 usability criteria is write them such that this section can clearly demonstrate that they have been achieved. Thus "good usability" is hard to demonstrate whereas "every menu will have a help option" is easier to demonstrate.

Criterion C3

Again this section was generally well done; it has not altered significantly from the previous programme except that criteria are a little more specific. The best candidates produced examples of error-handling code together with references to the hard copy output showing evidence that these were trapped.

Candidates often also produced tables of test data that were either included in this section or included in D1 and then referenced in this section – both these approaches tended to score very well and illustrated that candidates were well organised.

Criterion C4

There were a few cases where teachers awarded high marks but the candidate failed to produce output for criterion D1. If there is no evidence of a working program, the appropriate mark for this criterion is 0. It is certainly not sufficient for the teacher to merely state that that the program functions properly – evidence must be produced in all cases.

To avoid getting into this situation, teachers should allow time before final submission for candidates to produce the test runs and demonstrate those parts of the solution that do work. Candidates may find it useful to produce some partial test runs during coding as a fallback. These could be included in a development blog or written log. Clearly this is easier if candidates are able to select a problem amenable to incremental development (most problems are).

The output referenced in this section should also be related to the objectives stated in A2 which should be explicitly re-stated here along with any supporting evidence to show that they have been met.

Criterion D1

In the ideal case, the test runs are systematic, well planned and executed in a structured way. The annotations should indicate which part of the solution (as outlined in the objectives) is being tested. This need not be repeated from section C4 but should appear either in C4 or D1. It is feasible to consider a section which incorporates both C4 and D1 under a section entitled “Annotated hard copy” as indicated in the Subject Guide and on form 5/PDCS.

Again, this section is difficult to complete effectively unless there are clear and specific objectives listed under section A2.

This section should aim to demonstrate the working of the program under mostly “normal” situations; many candidates place too much emphasis on error detection to the exclusion of all else. Where feasible, runs should be included that demonstrate that sections of code claiming mastery do actually work. A common example at HL would be the proper function of additions to and deletions from files and other ADTs.

Criterion D2

The candidate is expected to reflect upon the achievements of the project, although not to the exclusion of all else. As indicated in the introduction, the process is more important than the sophistication of the application actually produced. Thus the effectiveness of the solution should be honestly discussed – how well it worked for the user (in relation to criteria stated in A2), rather than how good a solution it was in coding terms (this can be discussed under efficiency).

As well as feasible improvements, the candidate should examine the whole process gone through. This is helped by keeping notes during the project, either electronically (eg a blog) or in a notebook. Shortcomings of the design process and any lesson learned may well be the most valuable outcome of

an IB Computer Science dossier. Candidates only rarely considered alternatives to the design process itself.

Criterion D3

Many teachers appeared to miss the reference to "running the program on another machine without using the IDE" for an award of 3 marks for this section. On the whole, the best candidates are doing clear and illustrated instructions on program use. Weaker candidates show evidence of having rushed this section. With careful planning of test runs, screen shots can be taken that provide documentation for both D1 and D3.

Criterion E

The holistic mark did seem to be awarded fairly by the teachers with very few schools opting to give candidates a blanket maximum 3 as might have been feared. Often, teachers were kind enough to explain the basis for their award of this mark and this gives moderators confidence in the integrity of schools and teachers.

Mastery Aspects

The 2006 session revealed some variation in interpretation of mastery aspects across teachers, schools and candidates. In particular, some aspects are not well known or it is not appreciated that aspects such as encapsulation and inheritance are to be used as defined in the Subject Guide, Topic 5.

This report is designed to illustrate the way in which the mastery aspects should be approached by candidates and how teachers should give awards.

It should be stressed however, that there are many ways to program and many ways to achieve mastery. None of the examples presented here should be taken as templates to be applied by teachers or candidates to their own problems.

Perhaps the biggest issue with mastery aspects is the frequent lack of documentation or inaccuracies in documentation. These cover several aspects such as administrative errors, lack of hard copy output and lack of discussion of the need of the solution for certain mastery aspects.

Page numbers written down by students are sometimes incorrect and should be carefully checked by the teacher prior to submitting the dossier and form 5/PDCS. Moderators are not expected to look through pages of code listing in order to locate relevant mastery factors.

Candidates and teachers may refer to more comprehensive lists of mastery aspects within the dossier if they prefer; the location of such pages must be clearly indicated on form 5/PDCS.

Where the teacher has not completed the forms properly, dossiers may be returned to schools for correction resulting in considerable delays to candidate diploma results.

Design stage documentation of aspects

Typical examples of problems here include mastery aspects not being discussed in the data structures section, no algorithms or modules identified as manipulating a data structure. Yet there will often be an aspect mysteriously appearing in the code. This is often a sort where there is no apparent reason to sort and this may be claimed as mastery of recursion (Quicksort, Mergesort) at Higher Level.

Sometimes ADTs appear in the listing and are awarded mastery aspects by the teacher when they have not been described at all. The subject guide is absolutely clear that such use is considered trivial and cannot be awarded mastery aspects.

Use of standard algorithms

The use of standard algorithms is permitted, even encouraged in the completion of a dossier. Any such standard algorithms from published sources must be acknowledged in the usual way. However, moderators need to be sure that the candidate understands such algorithms. This can be demonstrated in several ways:

- The candidate discusses the modifications required to the standard algorithm to fit their own requirements or clearly explains the function of such algorithms in the context of their particular problem/solution.
- The candidate demonstrates sufficient understanding to the teacher who confirms this by making observations on 5/PDCs or 5/IACS and signing the forms.

As for any other IB subject, candidates who simply copy code from a book, website or other source (such as a colleague) will be penalised for plagiarism, probably resulting in a zero mark or fail for the component and thus the subject.

Libraries and Java utilities

Java has extensive built-in collections and library routines and equally many are available from 3rd party sources. While these may be used in solutions, no mastery aspects can be claimed from them.

This applies in particular to structures such as Linked Lists, Hash Tables and other ADTs.

Students are also not permitted to claim mastery where they wrap library methods inside trivial Classes and methods of their own.

Illustrative examples and further comments

All of the following aspects need to be documented in the design section and, where appropriate, demonstrated to work via hard copy output.

Adding data to an instance of the RandomAccessFile class by direct manipulation of the file pointer using the seek method.

Trivial examples of this include:

- seeking to the start of file – seek(0) – and writing a stream of data to the file.
- seeking to the end of the file seek(file.length) and appending data to the file.

If a Random Access File type is going to be used in a dossier, it really needs to be used as such. The data may be of a primitive type (eg int) but then some calculation must be made to write the int to a specified position within the file. Writing an array of ints, one value after the other will not satisfy this aspect.

Deleting data from an instance of the RandomAccessFile class by direct manipulation of the file pointer using the seek method.

Primitives or objects may be marked for deletion. Parts of the file marked in this way must be re-used, available for "undelete" operations or compacted at some time so that the file access is not significantly affected. If this is not done, because it is felt it is not needed for the particular problem at hand, the student should discuss this in section B1.

Components of the file may also be removed by shuffling records (by manipulation of the file pointer) over the item to be deleted, if the file is to be maintained as an ordered file.

Overwriting an existing record with a modified record may be claimed as deletion for this aspect. However, the modification may not be made in an external structure (such as a list or tree) and then the contents of this structure written to the file sequentially.

Searching for specified data in an instance of the Random AccessFile class.

This may be combined with addition or deletion as appropriate.

Recursion

Recursion must be documented and implemented correctly. Trivial use of recursion is a use where an iterative solution would work equally well (eg counting nodes or searching in a linked list).

Recursion should not be claimed when using a standard sort algorithm (mergesort or quicksort) that has not been documented at the design stage. Such documentation would normally be expected to justify the use of these algorithms as opposed to a simple O(n^2) for example. For these sorts the likely distribution of data in the unsorted list should be discussed. The teacher should also confirm that the student understands the algorithms in their notes on 5/PDCS or 5/IACS.

A common misuse of recursion is in input routines:

```
public int getPercent()
{
  int theInt = inputInt("Input a number between 1 and 100");
  if ( (theInt < 0) || (theInt > 100) )
  {
     getPercent();
  }
  return theInt;
}
```

This is trivial since the method could have been written iteratively equally well. It is a misuse of recursion since there is no clearly defined terminating condition. Note that this code doesn't work (although it will appear to if a return statement is added following the call to `getPercent()`).

This example does not illustrate recursion either:

```
void write()
{
  for (int i = 0; i < 10; i++)
    write(a[i]);
}
void write(int num)
{
  System.out.println(num);
}
```

Merging of two or more sorted data structures

The most common mistake here is to have a mergesort and then claim mastery of this aspect. Merging requires two data sets which are already sorted.

The distinction also needs to be made between merging and appending to a list structure.

Polymorphism

The three OOP mastery aspects (Polymorphism, Inheritance and Encapsulation) are included for candidates/teachers who prefer to adopt a proper Object Orientated approach to the design process. Therefore they will risk being seen as trivial in a dossier which adopts a structured design approach and simply includes extends or implements keywords. However, it is sometimes appropriate to award these aspects in other approaches where the candidate is able to explain the need for the aspect.

Polymorphism in Java is usually implemented either via interfaces or by inheritance. In a subclass or a class that implements a given interface, methods in the class can override methods in the parent class to produce different behaviours appropriate to that class.

Therefore implementing a toString() method that exists in Class Object (from which all Java classes derive) is technically polymorphism.

Similarly, when implementing the ActionListener interface, it is a requirement that the method actionPerformed(ActionEvent) is implemented by the Class which implements the interface – it is not a design choice of the student to do this.

Candidates may not use library code in their mastery claims and candidates have not written either of these superclasses. Therefore the above implementations of polymorphism are considered trivial.

To claim mastery of polymorphism, candidates must have designed, documented and implemented the superclass/interface and the subclass or class that implements the interface. The OOP design must be properly discussed in stage B. In this case it may be appropriate to depart from the suggested order of Criteria as indicated on page 57 of the Subject Guide.

Polymorphism can also be used to refer to relatively simple constructs such as method (including constructor) overloading. It could be appropriate for a student to claim this as an example of polymorphism if:

- The alteration in method parameters leads to a proper behavioural change in an algorithm
- The requirement for the approach is well documented in section B

To illustrate the first point, consider a method sort( ) which might be used to put the list in alphabetical order. Polymorphism might be used to implement methods such as:

- sort(boolean direction) where true/false can choose ascending/descending order, and
- sort(int start, int stop) to sort only part of the list.

As with other mastery aspects, the onus is on the candidate to demonstrate that they know why they are using a particular aspect and to state the benefits the aspect brings to their code. It is never acceptable to include such aspects without describing the need for them in the problem solution.

Inheritance

Similarly, inheritance cannot simply extend a built-in or library class and expect to claim this mastery aspect “by default”. A class which extends Object is both trivial and unnecessary.

To satisfy inheritance, the candidate must document and demonstrate how the subclass uses the methods and data members of the parent Class. For example, it is acceptable for the candidate to build “special” Panels, JPanels, JFrames, TextFields and the like as long as they document the need for the newly derived Class.

Simply using setSize(int, int), setVisible(boolean) or similar utility methods is trivial (and necessary for code to actually produce visible results).

However, the following would be good examples (as long as they are used by other Classes and properly documented):

- Creating custom Exception classes
- Creating Custom GUI objects:
    - J/Buttons with built-in mouse over events
    - J/TextFields which handle only strictly defined types of data (numbers, names, boolean values)
    - J/Panels/ which have a pre-set array of standard Buttons (Close, Cancel, Help) or other features

Thus, Higher Level candidates who are not using an OOP approach can still implement this aspect discretely if desired.

Encapsulation

Most Higher Level candidates should be able to implement a class which has both data members and methods. The data members should usually be private or protected so that other classes do not access the data members via dot notation but through appropriate public methods in the encapsulated class. Again, the candidate should be able to explain why such "data hiding" techniques are appropriate to the classes they are setting up to solve the problem they have identified.

Parsing a text file or other data stream

Using the built-in wrapper classes such as Integer and Double with their methods parseInt(String) and parseDouble(String) is considered a trivial use.

An input data stream, such as from a file, the keyboard or a network, which consists of character bytes representing text, should be split into separate components in some way. The use of String.split() or java.util.StringTokenizer is acceptable. The components must then be recognised as belonging to a particular set, each with a particular processing requirement. Taking lines from a text file in csv or similar formats and creating record-style object instances is an acceptable use. In this type conversion using Integer.parseInt() etc is fine. The formats and the process of interpretation must be discussed in the data structures and algorithms of stage B.

Implementing a hierarchical composite data structure

This has not changed from the previous Subject Guide and seems reasonably well understood. The need for the structure and the sample data it will store should be described in B1.

The use of any 5 standard level mastery factors

This does not require additional documentation – apart from indicating where these factors occur in the listing.

12-15 Up to four aspects can be awarded for the implementation of abstract data types (ADTs)

This is another mastery aspect that has been introduced to provide flexibility in designing solutions and to prevent too prescriptive an approach to program construction.

Normally, the candidate would select an ADT which is appropriate to their particular problem and data set and explain why that is so. It is common practice, but not acceptable, for stage B to deal with these ADTs in a theoretical way, rather than how the ADT meets these specific requirements.

Some ADTs are more complex than others, a list or stack is fairly simple to implement and candidates should thus be prepared to explain why that particular structure has been selected. As with sorts, standard book or website algorithms can be used, but their sources must be acknowledged in the usual way. The hard copy must demonstrate that these ADTs work and the teacher needs to write a note on 5/PDCS or 5/IACS confirming that the candidate understands the code used.

Where one or more candidates are using the same ADT the teacher must make a note regarding each candidate and the candidates must individually justify their use of the data structure – this will be different for different problems.

There is a risk that a "template" approach whereby everyone in the Class has a very similar problem with a very similar solution using the same ADT will raise the question of plagiarism. Thus candidates should be encouraged to "think through" the design for themselves. Collaborative work on the dossier is forbidden.

Candidates may gain some credit for all ADTs which have been properly documented, implemented and demonstrated to work via hard copy. Four such ADTs which all work only partially may gain the full 4 marks (although it does seem a hard way to go about it).

Candidates cannot simply wrap a library class with its associated methods in their own trivial class structure. For example:

```
import java.util.LinkedList;
public class TrivialList
{
  // example of a trivial ADT
  private LinkedList myList = new LinkedList();

  public static void main(String[] args)
  {
    new TrivialList();
  }
/**
 * Constructor for objects of class TrivialList
 */
  public TrivialList()
  {
    super();
  }

  public void addToHead(Object o)
  {
    myList.addFirst(o);
  }
  public void addToTail(Object o)
  {
    myList.addLast(o);
  }
  public boolean isEmpty()
  {
    return myList.isEmpty();
  }
}
```


Notice that a trivial stack, queue or deque could easily be built using the LinkedList and/or other java.util classes.

A common problem is when insufficient sample runs are presented to show that a program can write to and read from a random access data file or other ADT and can update these structures. This frequently requires more than one run of the program to have been recorded.

All of these aspects need to be documented in the design section and, where appropriate, demonstrated to work via hard copy output.

Recommendations for the teaching of future candidates

Many recommendations will be apparent from the foregoing section on specific criteria, however all sections of the dossier should benefit from candidates ensuring that their choice of topic:

- Includes an intended user
- Involves something the candidate understands
- Is solvable with the Java language
- Is approved by the teacher
- Has the potential to satisfy sufficient HL mastery aspects
- Is researched and analyzed BEFORE they start writing a program.

As in previous sessions, many teachers did not complete the reverse of Form 5/IACS and no comments were included with the samples sent. Comments help the moderator to evaluate the mark given by the teacher – the moderator wishes to confirm the mark awarded by the teacher whenever it is reasonable to do so.

In some cases teachers appeared not to award mastery factors where there was evidence in the dossier that these might have been achieved. The moderator will generally take the view that the teacher is best placed to make these awards and will not increase them.

Comments by the teacher and proper documentation by the candidate may avoid penalties in cases where the mastery aspect was simply not well understood and thus not claimed when it could have been.

Some mistakes were made when calculating the Final Mark in Form 5/IACS, teachers should carefully read the instructions in the Vade Mecum on calculating and recording marks.

Teachers should encourage students to show mastery in all aspects, as many marks can be lost if they do not do so.

The discussion in the evaluation should be from the programming point of view, not their personal achievements as programmers.

Teachers must discuss the assessment criteria and assessment guidelines with the candidates before beginning the program dossier. It should be made clear to the students what each criterion requires of them. At Higher Level, teachers should also be careful to outline the required mastery aspects and to discuss trivial versus non-trivial use.

The development of the dossier should follow the stages:

- analysis
- design

- development (coding)
- documentation

In particular, as noted above, the analysis and design should not be completed after the solution as this is a waste of time and effort on the part of the candidate.

An excellent strategy for the teaching of candidates is to insist on the production of sections A and B to a high standard before any development is done. Not only does this create a basis for a high grade in the dossier but also will improve the candidates' coding of the actual solution. In mathematical terms, given the 24 marks for these sections, the candidates would already have done enough to gain a grade 4 and still not have to achieve perfection on the subsequent criteria to get a top grade (6/7). This is extremely difficult to achieve unless the programme is being run over 2 years, however.

Mastery aspects may not be confirmed by moderators and candidates may be penalized if:

- Mastery aspects are not properly documented on 5/PDCS (or elsewhere within the dossier)
- Mastery aspects are not documented in Section B
- Mastery aspects are not documented in Section D1 – if appropriate
- The candidate has used algorithms from a book, utility library or website
- The teacher has not confirmed the candidates understanding of standard algorithms

Standard level internal assessment

Component grade boundaries

Grade:	1	2	3	4	5	6	7
Mark range:	0-6	7-13	14-20	21-27	28-35	36-42	43-50

The range and suitability of the work submitted

As this was the first session of a new set of criteria for the dossier and a new set of mastery aspects there were inevitably some misunderstandings and misinterpretations. This report will attempt to clarify what is required for a computer science dossier at Standard Level.

It is important for schools to be aware that moderators were given instructions to be generous in their interpretation of many of these criteria and mastery aspects but that this will not be the case in future sessions.

Dossier presentation was variable – the format is indicated on page 54 of the Subject Guide – but some candidates chose not to follow it. While presentation was often good, organisation and structure was often poor. To be fair to the students, it does not appear that the information in the guide was shared with them. Some few schools and candidates submitted work using the old format which inevitably lost them marks.

A number of students put items in appendices that should have gone elsewhere. Examples of this included feedback regarding the prototype and unlabelled, apparently randomly selected screen shots that might have been part of sample runs or user documentation. It is really only necessary to include surveys from the user(s) in a separate appendix when these are bulky – the results individual application forms and the like can easily be included in section A1.

The range of topics chosen was quite broad, as the new mastery aspects were intended to allow. Games continue to be handled poorly by all but the most able candidates. The best choices were usually those that had a clear, real intended user. Some schools submitted dossiers that did not demonstrate mastery of at least 10 aspects even though the solutions could potentially have done so – indicating poor pre-planning.

For weaker candidates some kind of database program is generally open-ended enough for them to demonstrate all of their skills and provides the most straightforward choice of problem. A number of simulations were presented for this session and these generally proved to be suitable topics.

In common with HL, many candidates concentrated on the solution and hardly ever discussed the problem they were attempting to solve. Interaction with users often appeared to be token or minimal and this clearly hampered the analysis and design part of the dossier.

While it must be recognised that this is a classroom exercise and the candidates' final programs are extremely unlikely to be used in the real world, the concept is to try to simulate a real-world design process as closely as possible.

It often appeared that candidates produced the analysis and design sections after the program had been written. This is a disadvantage to candidates, as careful attention to analysis and design can actually reduce time and work required for the solution, whereas writing this documentation afterwards is a time-consuming task with no value as a learning experience.

Sample runs including hard copy output were lacking in many cases. The Subject Guide clearly states that at least one complete test run of the program should be included. This should involve normal data and, where possible, demonstrate that the program functions as claimed. For example, if a mastery factor of adding to an array is claimed, there should be a part of the listing that shows the record being added and subsequently included in the data set. Many sample runs only show what happens when abnormal data is entered.

Where the program has a fatal flaw such that it cannot compile or enters an infinite loop sample output cannot be produced and coverage of mastery aspects is doubtful.

It is difficult for a moderator to distinguish this from a case where there is a working solution but no sample output has been produced for some other reason. Although moderators may make an effort to carefully evaluate the code listing this should not be relied upon by schools. Teachers should comment explicitly on such cases using the 5/PDCS form provided.

Candidates should take the opportunities provided by the language and the various free IDEs to structure code listings more intelligibly. Class diagrams and/or tables of contents enable both teachers and moderators to get a good overview of the solution and will also help in the documentation of mastery aspects.

Candidate performance against each criterion

Criterion A1

In many cases the problem was not analysed. The students were keen to start demonstrating what their program will accomplish without any consideration of why it will be doing this. The starting point should be a clear statement of the information problem that is to be solved. This information problem can ONLY be properly understood by talking to the user or users involved in using the current system. Students who start the solution with only a hazy idea of the problem to be solved inevitably do poorly in the dossier.

A real (not imaginary) user is required so that the students can obtain the views of a non-technical person and gather information from them. Often evidence of data collection was absent or superficial. Data collection was often treated as a "box to be ticked" by candidates and the data collected was never discussed or explained further.

The user need not have a problem that requires a real-life, all-singing, all-dancing computer program at the professional level – ideally it will be relatively simple. Students should be actively discouraged from attempting to create feature-rich or overly ambitious solutions.

In this context, the real user exists to bring the student out of their own narrow perspective in which they develop only for themselves. There is no reward (in IB terms) for a really first class program or a program that is actually used afterward. Part of the analysis and design process needs to limit the scope of the solution to an achievable size. Teacher guidance is essential at this stage.

The "creeping featurism" danger that leads to missing deadlines and experiencing too many programming difficulties can probably be avoided if the students understand that this is a classroom exercise, not a commercial product. Where students insist that they can produce such programs a good teaching strategy is to get them to do the core requirements of an IB dossier first and let them attempt any sophisticated additions at the end (if there is time).

Criterion A2

Many students did not provide a clear connection between the objectives and the problem analysis as is required by the criterion. Some students wrote descriptive paragraphs while others wrote bullet points – the latter probably being more effective.

The objectives should be few in number at this level, 6-8 is ideal, more simply add to documentation burden without demonstrating additional capabilities; for example 12 objectives will double the number of tests to be done, the output required to demonstrate success, the user instructions and make it much less likely that criteria such as C2 and C3 will achieve full marks. Thus, the selection of objectives requires careful thought.

Objectives should be SMART: specific, measurable, achievable, realistic and time-constrained. At this point the teacher and candidate should satisfy themselves that at least 10 (and preferably 12) mastery aspects can be achieved and know which they will be. A development plan at this stage would help candidates to create a realistic timeframe for completing the work to be done – this need not be included in the dossier but should be referred to frequently during development. Teachers should consider rigorously enforcing deadlines to ensure that candidates gain maximum credit for the work that they have done.

The objectives should relate to the analysis explicitly. Stating as an objective "I am going to organize the CDs" is meaningless unless it is clear what needs to be demonstrated to prove that the objective has been achieved.

Objectives were given but rarely referred to in later sections. In the later sections the objective should be stated, then the evidence that supports the accomplishment of that objective should be presented.

The limitations of the system should cover more than the available hardware although of course this is a consideration. As noted above, the candidate will frequently elect to solve only a small part of the user's problem and this can be included as a limitation here.

Another suggestion is to write **user-centric** goals rather than **program-centric** goals. As the students are trapped in their programmer-centric viewpoint, their objectives are always written about the program.

User-centric goals describe what the user will be able to accomplish, rather than what "features" the program will supply. It should be easier to test user-centric objectives. And it is obvious that testing a user-centered goal requires sample output, rather than reading some lines of a program listing. For example, consider rewriting the goals this way:

Program-oriented	rewritten as	User-oriented
Program saves CD titles in a data file	→	User can save CD titles and retrieve and view them later
Program can sort CD titles alphabetically	→	User can choose to view titles sorted alphabetically or chronologically (date order)

Some dossiers in the current session accepted user data input and "stored" it in an array, but never copied it into a file. The user would definitely want to keep the data from day to day, but the student programmers apparently thought it was okay to get the array mastery mark and ignore the file I/O.

This might make sense in a proto-type, but it is hard to imagine writing any kind of data processing software that loses all its data when the machine is turned off. Again, a real user who has real needs and who is actively consulted should be able to identify these objectives for the candidate. The candidate, as stated earlier, is not obliged to do everything the user wants but should “cherry pick” the 5 or 6 that fit the constraints mentioned above.

Criterion A3

Once the major objectives and boundaries of the proposed solution to the simple problem have been clearly and unambiguously identified, then it should be possible to identify the major elements of the solution as well as its planned appearance.

This leads to an initial modular design for a prototype which can then be shown to the user for comment. The user’s comments should be documented but need not always be followed to the letter – particularly if their suggestions would lead to an overly extended solution.

A prototype will help the candidate form a clear picture of the “product” in their own minds.

In this session many candidates did not include an initial design or user feedback on the prototype. Non-functional prototypes created using various graphics packages were often presented in this section and are useful.

It can be beneficial for a non-functional prototype to be written in Java as the candidates will gain insights which are useful for algorithm design and it also satisfies their craving to get on and write some code at an early stage – clearly this should not be taken to extremes. A Java prototype with method stubs for the main functions is a useful starting point for coding later on.
Often the feedback collected from users was really trivial or was not incorporated to modify the prototype. The candidates’ response to user feedback on the prototype was usually not given.

The prototype should be used to break up the program into logical sections from a user-viewpoint. This probably involves choosing menu items and data fields on a form. The initial design can be more program-centric, involving data-structures and algorithms. If the "initial-design" and the "prototype" are approached from these two different viewpoints, it makes sense to ask whether the two connect sensibly (criteria A3).

Criterion B1

At Standard Level, the main data structures will be arrays and/or files. Candidates using records should illustrate these, provide sample data from the problem domain and justify their selection of data types.

As in the previous version of the dossier criteria, illustrations are extremely valuable and required for the top marks. Illustrations must use data from the problem the candidate is attempting to solve; generalized diagrams of the operation of array operations or file i/o in the abstract do not satisfy this criterion.

Candidates may not use any library or utility functions or code that they have not written or modified themselves in order to claim a mastery aspect. At Standard Level the principal problem in this respect is code using Java.util.arrayList which cannot be used to claim mastery of arrays as a static type is expected. Files must be used for both input and output. Specific mastery aspects are discussed further in a later section.

Criterion B2

The algorithms should be sufficiently detailed that the solution can be created from them – they do not have to be perfect nor complete – but all of the essential modules identified in B3 or A3 should be present. Not everyone agrees that this is a useful way to approach the writing of code; nevertheless it remains a requirement of this Computer Science programme.

As ever, some candidates presented code that was clearly derived from the solution and were penalized accordingly. Moderators find this practice very easy to spot.

Criterion B3

As indicated in the Subject Guide a range of presentations is acceptable for this session – often dependent upon the design approach used. However, all these variations should show the connections from modules or classes to algorithms and data structures. This section often provides the "big picture" of the proposed solution whether it is completed as a top-down modular design or an Object Oriented Design or indeed a "bottom-up" approach.

A variety of presentations were used here but the most common failing was to forget to identify the connections to data structures and algorithms.

Criterion C1

This was mostly well-addressed by the candidates who often used good indentation and clear identifier names. However, in many listings it was hard to find out where Classes began and ended and to determine the relationship between them. This is more difficult when the solution departs from the design, which is permitted.

Ideally a Class should start on a new page or have a clear separation from preceding Classes and should have a header statement describing its function. A list of Classes with page numbers at the start of the code is extremely useful. Some IDEs provide relationship diagrams which are also helpful to those finding their way around the code.

Candidates should ensure comments are readable if printed in black and white; some IDEs have comments in grey which are hard to read.

A complete listing of the final functioning program **must** be submitted in this section; it is not sufficient to use the algorithm section B2 or other parts of the documentation as the code listing.

Criterion C2

This section was generally done well by candidates. Many of them chose to use a summary approach, referring to hard copy printout supplied as part of section D1. This worked better than any amount of descriptive text lacking illustrations or specific examples.

This section should reference the A2 criteria explicitly and this was not done in very many cases. Again, the idea behind the A2 usability criteria is write them such that this section can clearly demonstrate that they have been achieved. Thus "good usability" is hard to demonstrate whereas "every menu will have a help option" is easier to demonstrate.

Criterion C3

Again this section was generally well done; it has not altered significantly from the previous programme except that criteria are a little more specific. The best candidates produced examples of error-handling code together with references to the hard copy output showing evidence that these were trapped.

Candidates often also produced tables of test data which were either included in this section or included in D1 and then referenced in this section – both these approaches tended to score very well and illustrated that candidates were well-organised.

Criterion C4

There were a few cases where teachers awarded high marks but the candidate failed to produce output for criterion D1. If there is no evidence of a working program, the appropriate mark for this criterion is 0. It is certainly not sufficient for the teacher to merely claim that the program functions properly – evidence must be produced in all cases.

The output referenced in this section should also be related to the objectives stated in A2 which should be explicitly re-stated here along with any supporting evidence to show that they have been met.

Criterion D1

In the ideal case, the test runs are systematic, well-planned and executed in a structured way. The annotations should indicate which part of the solution (as outlined in the objectives) is being tested. This need not be repeated from section C4 but should appear either in C4 or D1. It is feasible to consider a section which incorporates both C4 and D1 under a section entitled "Annotated hard copy" as indicated in the Subject Guide and on form 5/PDCS.

Again, this section is difficult to complete effectively unless there are clear and specific objectives listed under section A2. Even where this was the case, many candidates did not test them explicitly in this section.

This section should aim to demonstrate the working of the program under mostly "normal" situations; many candidates place too much emphasis on error detection to the exclusion of all else. Where feasible, runs should be included that demonstrate that sections of code claiming mastery do actually work. A common example at SL would be the proper function of additions to and deletions from files and arrays.

Criterion D2

The candidate is expected to reflect upon the achievements of the project, although not to the exclusion of all else. As indicated in the introduction, the process is more important than the sophistication of the application actually produced. Thus the effectiveness of the solution should be

honestly discussed – how well it worked for the user (in relation to criteria stated in A2), rather than how good a solution it was in coding terms (this can be discussed under efficiency).

As well as feasible improvements, the candidate should examine the whole process gone through. This is helped by keeping notes during the project, either electronically (eg a blog) or in a notebook. Shortcomings of the design process and any lesson learned may well be the most valuable outcome of an IB Computer Science dossier. Candidates only rarely considered alternatives to the design process itself.

Criterion D3

Many teachers appeared to miss the reference to "running the program on another machine without using the IDE" for an award of 3 marks for this section. On the whole, clear and illustrated instructions on program use are being done by the best candidates. Weaker candidates show evidence of having rushed this section. With careful planning of test runs, screen shots can be taken that provide documentation for both D1 and D3.

Criterion E

The holistic mark did seem to be awarded fairly by the teachers with very few schools opting to give candidates a blanket maximum 3 as might have been feared. Often, teachers were kind enough to explain the basis for their award of this mark and this gives moderators confidence in the integrity of schools and teachers.

Mastery Aspects

These are guidelines only; teachers are expected to use their professional judgement in the award of mastery aspects.

The 2006 session revealed some variation in interpretation of mastery aspects across teachers, schools and candidates. In particular, some aspects are not well known or not sufficiently well-defined in the Subject Guide.

This section of the report is designed to illustrate the way in which the mastery aspects should be approached by candidates and how awards should be given by teachers.

It should be stressed however, that there are many ways to program and many ways to achieve mastery. None of the examples presented here should be taken as templates to be applied by teachers or candidates to their own problems.

Perhaps the biggest issue with mastery aspects is the frequent lack of documentation or inaccuracies in documentation. These cover several aspects such as administrative errors, lack of hard copy output and lack of discussion of the need of the solution for certain mastery aspects.

Page numbers written down by students are sometimes incorrect and should be carefully checked by the teacher prior to submitting the dossier and form 5/PDCS. Moderators are not expected to look through pages of code listing in order to locate relevant mastery factors.

Candidates and teachers may refer to more comprehensive lists of mastery aspects within the dossier if they prefer; the location of such pages must be clearly indicated on form 5/PDCS.

Where the teacher has not completed the forms properly, dossiers may be returned to schools for correction resulting in considerable delays to candidate diploma results.

Design stage documentation of aspects

Typical examples of problems here include mastery aspects not being discussed in the data structures section, no algorithms or modules identified as manipulating a data structure. Yet there will often be an aspect mysteriously appearing in the code. This is often a sort where there is no apparent reason to sort and this may be claimed as mastery at Standard Level.

Use of standard algorithms

The use of standard algorithms is permitted, even encouraged in the completion of a dossier. Any such standard algorithms from published sources must be acknowledged in the usual way. However, moderators need to be sure that such algorithms are understood by the candidate. This can be demonstrated in several ways:

- The candidate discusses the modifications required to the standard algorithm to fit their own requirements or clearly explains the function of such algorithms in the context of their particular problem/solution.
- The candidate demonstrates sufficient understanding to the teacher who confirms this by making observations on 5/PDCs or 5/IACS and signing the forms.

As for any other IB subject, candidates who simply copy code from a book, website or other source (such as a colleague) will be penalised for plagiarism.

Libraries and Java utilities

Java has extensive built-in collections and library routines and equally many are available from 3rd party sources. While these may be used in solutions, no mastery aspects can be claimed from them.

This applies in particular to structures such as ArrayLists.

Arrays

The java.io.ArrayList Class or various other vector-style implementations should be avoided as this aspect refers to traditional static arrays. The students are expected to be able to control the updating of array elements while avoiding common pitfalls such as accessing null elements or elements outside the array bounds.

Therefore candidates should use standard arrays with subscripts in [square brackets].

A good, common example might be to input a list of names and store these in a String array. Then let the user edit the array and eventually save it into a text file.

User-defined objects

The student is expected to set up a Class definition and to use an instance of this Class and its data members and/or methods in another Class. It is not necessary for Standard Level candidates to understand the classic OOP concepts, such as inheritance and encapsulation, however.

Creating the main program as a class does not count, creating objects from standard Java classes does not count. An example might be creating a Class that can store (and use a method to validate) a phone number which is then used within the program to store and validate phone numbers.

Objects as data records

This involves using a Class with data members corresponding to the fields of a record – to avoid a trivial approach, the fields should be of different types as appropriate to the data being stored. Such a Class might include validation either in the constructor or in mutator (setter) methods if the students have got that far (but isn't strictly necessary).

This aspect has the advantage of fulfilling the previous mastery aspect at the same time.

Storing data in the main program does not satisfy this aspect. The Class which is written needs to be used within the main program. An example might be creating a CD class containing fields such as title, artist and length/number of tracks – use of data types other than String is usually a requirement. The CD class must be used to create CD objects in the main program.

Simple selection

This involves the use of a number of simple if or if else statements to control code execution. The exact number is hard to specify but more than one would be desirable.

```
if (scores[x] < passMark)
{
  result[x] = "fail";
}
```

Selection constructs appear repeatedly in almost any program. It is not necessary to document all the examples - documenting a couple of clear examples is sufficient.

A try .. catch construct does not count as selection. More than one example of an if statement in a program would be needed to demonstrate mastery.

Complex selection

Complex selection would be using an if statement with multiple conditions (and logical operators), nested if statements, multiple chained if else statements or a switch.

Students who can handle complex selection can claim mastery of simple selection as well.

Example 1

```
if ( (x < 8) && (x >0) )
{
  ok = true;
}
```

Example 2

```
if (x < 8)
{
  if (x > 0)
  {
    ok = true;
   }
}
```

Example 3

```
if (opt == 1)
{
  addItem(Food item)
}
else if (opt == 2)
{
```



```
    deleteItem(Food item)
}
```

Although `switch` is not part of JETS it can equally well be used in the last example.

Normally, this aspect will occur without detailed documentation. A try .. catch statement does not count as complex selection.

Loops

Any kind of (logically workable) loop will serve for this criterion. This aspect will usually occur frequently in non-trivial dossiers. A try – catch block cannot be counted as a loop. More than one example will be needed within a program in order to claim mastery of this aspect.

It is recommended that candidates avoid complex conditions in for loops as these can easily produce infinite loops (which then show non-mastery). A sequential search through an array is a good use of looping and demonstrates the searching aspect too (when properly documented in section B). Loops which never execute do not count as mastery, for example:

```
for(x=0;x<0;x++)
```

Nested loops

Nested loops are loops which are correctly placed inside other loops and produce a desired result, which has been documented at some point.

```
while ( cp != (SIZE - 1) )  //while not at end of array
{
  cp = cp + 1    // increment current to first in unsorted
  pt = cp;       // pointer into sorted part - starts at top
  tm = data[cp]; // temp store for element to insert
  // while not at start, and next value is still too big,
  // shuffle up current element by 1
  while ( (pt > 0) && (data[pt-1] > tm) )
  {
    data[pt] = data[pt - 1];
    pt = pt - 1;
  }
  // insert the temp value into the sorted part
  data[pt] = tm;
}
```

Notice that this loop is part of a sort routine and thus would serve for 3 mastery aspects. However, the **need** for sorting **must** have been established in the **design** of the solution and **must** be demonstrated to work via **hard copy**.

The example below shows simple looping and simple selection, NOT nested loops or complex selection:

```
for (int pos = start + 1; pos <= end; pos = pos + 1)
      {
          if (data[pos] < minSoFar)
          {
            // found a new mimimum
            minSoFar = data[pos];
            minPos = pos;
```



```
            }
        }
        return minPos;
    }
```

User-defined methods

Good programming techniques will lead naturally to the use of short segments of code performing a single task and combined into a method. This method can be void.

The documentation for a method **must** be found in the algorithm design section B2 for this mastery aspect to be awarded.

The use of public static void main does not count as a user-defined method and neither does a simple constructor method for a main program. Methods which are created by the IDE or required by implementing standard Java interfaces (eg ActionListener) do not count.

A method that is only called once does not count as mastery of this aspect. Examples of good uses for methods are: searching, sorting, validating user input and re-formatting a String.

User-defined methods with parameters

Parameters will be passed into a method and used within the body. A classic example might be to print blank lines to the output terminal:

```
public void printBlanks(int n)
{
  for (int x = 0; x < n; x++)
    System.out.println("");
}
```

Again, this ought to be documented somewhere in B2. If the student subsequently uses multiple statements such as:

```
System.out.print("\n\n\n\n\n\n\n\n");
```

instead of

```
printBlanks(8);
```

It suggests that perhaps they have yet to master the concept of methods with parameters.

If the above is the only example of a method in the program it is certainly trivial since most non-trivial programs could be expected to make good and effective use of methods of all kinds.

As moderators we hope not to see almost every dossier in the next session using this method – these are illustrative examples, not templates to be used by candidates.

Methods which are created by the IDE or required by implementing standard Java interfaces (eg ActionListener) do not count.

User-defined methods with appropriate return values

These methods are not void but return a value of some kind via a return statement. However, more than a simple return statement is required, for example this is trivial (and terrible programming):


```
public void returnFourtyTwo(int y)
{
  d = 42;
  return;
}
```

The parameter is not used, d is non-local and the return statement is redundant.

Again, in a non-trivial dossier problem that a Standard Level student is expected to tackle should provide ample opportunity for students to develop suitable methods that meet these criteria.

Methods which are created by the IDE or required by implementing standard Java interfaces (eg ActionListener) do not count. A method that is only called once does not count.

Sorting

One of the reasons for having more than the 10 required mastery aspects is that students do not have to find problems that require a given aspect. Therefore it is expected that students will introduce a sort for a reason that is related to the problem domain rather than one introduced merely to satisfy a mastery requirement.

A related problem is that a sort is potentially worth 3 mastery aspects and therefore the moderator will need to be convinced of this need otherwise the candidate may put more than one mastery aspect at risk.

Sorting must use loops or recursion, not for example, a series of selection (if) statements on a small data set.

Searching

There are no particular restrictions on the need for searching other than those pointed out for sorting. A simple linear search will require a loop and a selection statement, and possibly use of a sentinel also.

This is a rather simple process to potentially gain 4 mastery aspects and therefore the design documentation must be adequate and the hard copy must demonstrate success. Also if the search is the only point in the program where selection, looping and flags appear, this would be considered trivial.

Searching must use loops or recursion, not for example, a series of selection (if) statements on a small data set.

File I/O

Data must be both written to and read from a file (may be a simple text file or ObjectOutputStream) and must persist between runs of the program – and this has to be demonstrated in hard copy too.

Standard Level candidates are permitted to use an SQL database to satisfy this requirement (for example, Access) and java.sql functions to update the database (additions, edits and deletions must all be present). This will also satisfy the criteria for Use of additional libraries.

In this case, SQL cannot be used to demonstrate mastery of searching. The use of an SQL database does not imply mastery of objects as data records (the candidate has used the database, not Java, to create the record set).

A fixed SQL command string must not be used, values must be retrieved in a flexible way, e.g. the SQL command to update the data set must be generated in a flexible way such that any values can be supplied – not a programmatically pre-defined set.

A good way of using the aspect in this way might be to create a method for saving an array of values into an SQL database, and another method for retrieving the values and copying them back into the array

In common with all other mastery aspects, this cannot be awarded if no reason for use or explanation of its operation is given in section B1 – along with sample data and illustrations. The solution must have a real requirement for long term storage (most applications do) and this requirement needs to be explicitly stated.

Use of additional libraries

Math and String are part of JETS so the libraries must be others such as Abstract Windows Toolkit, Swing, a built-in Collection or a utility like StringTokenizer. An interface may be implemented as well as a library imported, thus ActionListener is a valid example (provided that the listeners are actually set and detected).

It is important that the need for such libraries is demonstrated and documented at the design stage. If the utility is providing storage or other functions, its successful operation needs to be demonstrated by hard copy output as well.

Using classes created by the student does not count

Only using fully automated IDE code (e.g. for Swing) is not sufficient - the student must also write some code themselves that uses the library, and this code should be clearly identified as such.

In the current session, this mastery aspect was sometimes not claimed when it appeared that it might have been.

Use of sentinels or flags

The sentinel may be the end marker of a data set, for example – it is not part of the data itself. It is not a value typed in by the user (eg “yes” or “no”). It is not a Boolean value returned by some method.

A flag is often set in a search or returned by a particular method (eg a validation check). Normally, this aspect will form part of most non-trivial dossier programs and will most likely be incorporated at more than one point.

The flag need not be a boolean data type.

A good example could be marking the end of a list of data in an array with a phony value ("xxx" or -999) and programming a search to stop when it reaches this value.

Recommendations for the teaching of future candidates

Many recommendations will be apparent from the foregoing section on specific criteria, however all sections of the dossier should benefit from candidates ensuring that their choice of topic:

- Includes an intended user
- Involves something the candidate understands
- Is solvable with the programming language available to them

- Is approved by the teacher
- Will satisfy the required SL mastery aspects
- Is researched and analyzed BEFORE they start writing a program.

As in previous sessions, many teachers did not complete the reverse of Form 5/IACS and no comments were included with the samples sent. Comments help the moderator to evaluate the mark given by the teacher – the moderator wishes to confirm the mark awarded by the teacher whenever it is reasonable to do so.

In some cases teachers appeared not to award mastery factors where there was evidence in the dossier that these might have been achieved. The moderator will generally take the view that the teacher is best placed to make these awards and will not increase them.

Comments by the teacher and proper documentation by the candidate may avoid penalties in cases where the mastery aspect was simply not well understood and thus not claimed when it could have been.

Some mistakes were made when calculating the Final Mark in Form 5/IACS, teachers should carefully read the instructions in the Vade Mecum on calculating and recording marks.

Teachers should encourage students to show mastery in all aspects, as many marks can be lost if they do not do so.

The discussion in the evaluation should be from the programming point of view, not their personal achievements as programmers.

Teachers must discuss the assessment criteria and assessment guidelines with the candidates before beginning the program dossier. It should be made clear to the students what each criterion requires of them. At Higher Level, teachers should also be careful to outline the required mastery aspects and to discuss trivial versus non-trivial use.

The development of the dossier should follow the stages:

- Analysis
- Design
- Development
- Documentation

In particular, as noted above, the analysis and design should not be completed after the solution, as this is a waste of time and effort on the part of the candidate.

An excellent strategy for the teaching of candidates is to insist on the production of sections A and B to a high standard before any further development is done. Not only does this create a solid platform for a high grade in the dossier but also will improve and inform the candidates' development of the actual solution. In mathematical terms, given the 24 marks for these sections, the candidates would already have done enough to pass and still not have to achieve perfection on any subsequent criterion to get a top grade. This is extremely difficult to achieve unless the programme is being run over 2 years, however.

Moderators may not confirm mastery aspects and candidates may be penalized if:

- Mastery aspects are not properly documented on 5/PDCS (or elsewhere within the dossier)

- Mastery aspects are not documented in Section B
- Mastery aspects are not documented in Section D1 – if appropriate
- The candidate has used algorithms from a book, utility library or website without proper referencing
- The teacher has not confirmed the candidates understanding of standard algorithms

Higher level paper one

Component grade boundaries

Grade:	1	2	3	4	5	6	7
Mark range:	0-15	16-30	31-41	42-51	52-61	62-71	72-100

Range of achievement

There was a large range of marks with fewer at the very low end than in previous years as well as many good papers. There was no marked difference between performances in sections A and B. There was no evidence that candidates were short of time. Only a handful answered more questions than demanded by the rubric of the paper. In each case only the first four questions in section B were marked.

The areas of the programme and examination which were successful or unsuccessful for candidates

Candidates generally competently answered straightforward questions, such as fitting data to a binary search tree, defining DMA and tracing an algorithm.

Floating point representation, the machine instruction cycle, OMR and *polling* are basic computing terms that were unfamiliar to some candidates.

The OOP terms *encapsulation*, *inheritance* and *polymorphism* were evidently well known as terms and many were able to give a definition but the attempts to relate these definitions to an application demonstrated that they were not fully understood. Similarly the examples given for *interrupts* were valid uses of *interrupts* but bore little relation to the application.

The strengths and weaknesses of the candidates in the treatment of individual questions

In general the paper was of appropriate standard with few questions causing major problems for ALL candidates or markers.

The distribution of marks resembled a normal distribution, thus spreading students at the top end over a reasonable range, distinguishing clearly a 6 and 7 from lower levels of performance.

Areas of the programme which proved difficult for candidates

In section A Q2 and Q13 proved difficult, as did Q16 in section B. These questions deal with binary and hex representation of data along with issues of arithmetic and associated errors. This is a clear area that teachers and students need to concentrate on.

Question 8 (c) was outside the scope of the course and few students answered it correctly.

A number of candidates performed poorly on the algorithm parts of Q15 in section B. These algorithms were reasonably standard and it is disappointing, given the requirements of the Dossier, that students were not able to handle these basic questions. Teachers and students need to pay further attention to algorithm development.

Question 5 in section A was answered very poorly, possibly due to the change in the way the course addresses these concepts. The notion of an argument is the value that is passed to the variables declared in the method signature e.g. in the call average (12, 20) both 12 and 20 are arguments. Parameters are declared in the method signature e.g. double average (double n, double m), the variables n and m are parameters.

In future this type of question will use the term method signature and call to the method. Students can expect to be asked questions about a particular aspect of a method signature or about the components of a call to a method.

Levels of knowledge, understanding and skill demonstrated

Students in general demonstrated a reasonable understanding of the course, with the exception of the binary representation and algorithms.

Students in general tended not to answer the Explain or Discuss type questions with due emphasis on the requirements of the Action Verbs. Students need to be clear as to the meanings of these terms. Also, in general n marks requires n points, the application of this simple rule would also assist students to structure their answers.

A. The strengths and weaknesses of candidates in the treatment of individual questions

Some of these have been referred to in (A) above.

Strengths

Students showed good understanding of the basis theory. Questions such as 1, 2, 4, 6 etc were well answered.

Question 17 on Boolean logic was well answered, with many students demonstrating the ability to simplify Boolean expressions.

Question 18 was well answered by a range of students showing students basic understanding of sequential files and social and ethical issues associated with data collection.

Question 19 was in general well answered with many students being able to define direct access and to construct a binary tree to store the index.

Question 20 showed that many students understood the basic ideas behind polling and interrupts and were able to recommend the appropriate technique. Questions (c) and (d) were both handled well showing that students had a good appreciation of different data capture/input methods and were able to determine advantages and disadvantages.

Weaknesses

Algorithms: it is expected that students can represent basic problem solution logic in the form of an algorithm. Many students were not able to handle methods that receive data and then return a value. Java allows void methods that act like procedures and do not return a value. Java also allows methods to have a data type to indicate that a value is returned, such methods operate in the same ways as a function.

Students are expected to be able to write modular solutions and to reuse methods as is appropriate.

Binary and hex representation is poorly understood. Students can expect to be asked questions that require understanding of the representation of positive and negative integers, hex representation, minimum and maximum values of n bits, addition of binary integers and show an understanding of errors such as overflow and underflow.

File access: many students were not able to answer 19 (e). Direct access requires knowledge of the relative or physical address on disk. An index is often used to store the primary key and the relative record position. It is expected that students can explain how a data structure could be used e.g. array, linked list of binary tree to allow records to be accessed directly by use of a primary key. In this instance the primary key i.e. userID is used to access the binary tree nodes and then the relative record position is used to seek the record in the file.

Question 20 (a) was very poorly answered. Teachers and students need to be able to represent the relationships between components using a systems flowchart. The direction of the relationship is also important.

Recommendations and guidance for the teaching of future candidates

Candidates should make sure that their answers are related to computer science. Answers that show good “common sense” but no technical knowledge do not earn many marks.

Many candidates did demonstrate technical knowledge but were unable to apply it to the specific question. Two strategies to help this are to read the question carefully before attempting to answer it.

A little reflection would help appropriate application. Another is to practise questions in past papers. The more diverse applications that they meet, the more likely they are to be able to apply knowledge to new situations.

Higher level paper two

Component grade boundaries

Grade:	1	2	3	4	5	6	7
Mark range:	0-15	16-30	31-40	41-50	51-61	62-71	72-100

Range of achievement

The paper was generally accessible to most candidates and this was borne out by the fact that there were few very poor scripts and some exceptional ones. On the whole the level demonstrated was very satisfactory. Most candidates were able to make a good attempt at the algorithms. Only a handful left out parts of this question and there were some exceptionally good algorithms produced. Many candidates demonstrated a good knowledge of computer science and quite a few were able to show deeper understanding of the theory. There was no evidence that candidates were short of time.

The areas of the programme and examination which were successful or unsuccessful for candidates

Algorithm construction was particularly successful although calling a previously written function or procedure correctly, as in the case of HAMMING and SWAP in question 1 proved difficult for candidates.

Answers to the Case Study question often lacked computer science and relied on repetitive quotes from the Case Study – sometimes irrelevant ones.

Systems flowcharts appeared unfamiliar to many and some produced programming flowcharts instead.

Data capture methods and types of processing did not appear to have been studied to the level required to answer the questions adequately.

Areas of the programme that proved difficult for candidates

Candidates appeared to fall into two man areas – those that could interpret, trace and write algorithms and those that could not. Some candidates appeared to know very little Java and to be quite unprepared for this examination.

Many candidates showed little knowledge of packet switching and associated networking concepts. The knowledge of file structures appears to have improved relative to previous years but some candidates still did very poorly in this area.

The case study was generally well answered and the weaker candidates appeared to gain most of their marks in this area.

There were few candidates who showed good knowledge of both programming concepts and general subject knowledge but obviously these were the candidates that performed best.

Level of knowledge, understanding and skill demonstrated

There is a wide knowledge base among candidates entered for this examination. The performance of many candidates hovered around the average. A few candidates demonstrated very high levels of both knowledge and skill in answering all types of question.

Completing the recursive trace table appeared to be an issue for very many candidates. Many candidates seemed to struggle with the concepts around void functions as opposed to those returning a value. Many candidates find parameters difficult which is surprising at Higher Level.

Many candidates answer questions in a vague and non-specific manner and do not pay sufficient attention to the mark allocation or action verbs used in a question.

The strengths and weaknesses of candidates in the treatment of individual questions

Question 1

As noted above there were few candidates who scored in the mid-range on this question; they either knew how to deal with algorithms or they did not. In fact, some candidates appear to know very little about Java programming and one wonders how they manage to complete a Higher Level dossier program at all.

It is true that a recursive algorithm is tricky to trace but the remaining question parts were relatively straightforward. There were very few complete solutions to the trace table – many candidates seemingly lost completely.

The algorithm for part (b) was done well on the whole by those candidates who attempted it. Removing duplicates from the array seemed very difficult for candidates although most could mark the duplicates (eg with 0 or null). There were issues with making the transfer and correctly sizing the resultant array.

Most candidates who made some attempt at parts (b) and (c) could also complete part (d) successfully.

Question 2

This question was generally well answered by candidates who had learned the basic principles of packet switching and related networking concepts. Surprisingly, it appears that this is not the case in many schools. Of those who answered well, many struggled to gain all the marks for part (b) relating to transmission of packets across the Internet. It should be noted that there were also a number of excellent answers to this part.

Candidates sometimes lacked detailed knowledge of data communications checks as required by part (d) although answers relating to specific types of medium (such as fibre optic) were also accepted here.

Candidates answered part (e) fairly well and usually were able to identify encryption as a security measure. Some candidates failed to note the need for decryption at the receivers end.

Question 3

A very few candidates confused file structures with abstract data types which was disappointing. Most candidates could at least identify serial and sequential files and a good number could identify all four types stated in the subject guide.

Candidates who knew something about files were not always clear and specific in their answers which were often vague. The concept of indexing was sometimes confused as to the distinctions between fully-indexed (unordered) files and partially-indexed (ordered) files – both of which were acceptable answers to part (c) if properly explained. A few candidates also gained marks for explaining the process of direct access via hash tables for this part question.

Part (e) required the candidates to make a choice of medium and supply 4 or more points in support of their choice – not all candidates appeared to recognise that this was indicated by the mark allocation for the question part which suggests they might need additional practice in examination technique.

Question 4

The case study question continues to be difficult for many candidates who give the impression that their first sight of it is in the examination room on the day of the examination. Thus answers sometimes consist of random quotes from the case study which are largely unrelated to the question.

The case study is introduced to allow teachers to explore computer science concepts within a specific application – in this case MIDI.

The two parts (a) and (b) about the processes of recording music using MIDI and sampling were a case in point. Many candidates did not seem to grasp the fundamental differences between these two processes and gave quotes from the case study which did not demonstrate understanding of the

processes. Again, candidates did not appreciate the number of marking points required for each of these question parts.

Part (c) about recording methods was answered better which suggests that candidates knew more about the recording methods than they chose to reveal in parts (a) and (b).

Parts (e) and (f) were often given vague, general and ambiguous answers and again, candidates failed to supply the required number of points as indicated by the action verb used and the mark allocation given.

The final part questions on (g) were answered well by almost all candidates.

The type of assistance and guidance the teachers should provide for future candidates

There was a general weakness in examination techniques shown by many candidates who otherwise seemed to demonstrate ability in the subject. One useful technique is to give candidates sample scripts to mark –teachers can write these themselves with a view to demonstrating common errors such as:

- Failing to supply any kind of answer
- Answering a different question to that which was asked
- Writing too much for a simple "state" question
- Writing too little for a question with many marks allocated

Candidates should be exposed to writing algorithms under examination conditions and a good time to do this is in the lead up to the mock examinations and to the examination itself. There is no real substitute for practice at tracing, interpreting and writing algorithms under timed conditions.

Candidates should be supplied with a copy of the subject guide and given time to assess their knowledge of it and to fill any significant gaps before the examination. Many candidates appeared to know nothing at all about large areas of the Subject Guide .

Recommendations and guidance for the teaching of future candidates

Exposing students to a variety of systems flowcharts early on in the course and then encouraging them to represent systems in this way themselves when exploring systems would help them to understand the difference between programming and systems flowcharts.

Different types of processing and details of how and when they occur should be studied at frequent intervals. In particular, candidates should be encouraged to describe what happens from a computer science point of view and not only from a users view point.

Past papers carried out in timed conditions might bring home the fact that reading the question carefully is essential to giving an appropriate answer.

Standard level paper one

Component grade boundaries

Grade:	1	2	3	4	5	6	7

Mark range:	0-12	13-24	25-30	31-37	38-43	44-50	51-70

General comments

The paper seemed straightforward. Most candidates performed well in this component and only a few showed a poor standard. There were opportunities to pick up easy marks, but even so there were few high scores. There was a high correlation between the two sections. Most candidates who scored well in Section A also did so in Section B, and those who scored badly did so in both sections.

The areas of the programme and examination that appeared difficult for the candidates

The paper was well answered by most candidates. The weakest ones provided answers where there was an evident lack of detail or lack of computer science answers. Most of these weak candidates were unable to give definitions without using the word they were trying to define (e.g. "Logic errors are errors in the logic"). Therefore, some definitions were not correctly given by these weak candidates.

The weaknesses were especially found in the definition of errors, the difference between an index and the data in an array (P and X[P]), the difference between online and real time and in the software life cycle. Other answers which showed lack of detail were mainly found when defining pass-by-reference parameter, understanding how files can be linked and defining the type of processing involved while counting the new users (Q14).

The areas of the programme and examination in which candidates appeared well prepared

Very few candidates left questions unanswered. Most of them showed good knowledge of compilers, data compression, LANs and Networks. They were also good at tracing algorithms and completing the trace table.

The strengths and weaknesses of the candidates in the treatment of individual questions

Section A

Question 1

Even the weakest candidates were able to state a function of the compiler.

Question 2

Some good answers were found to this algorithmic question, though some candidates still get confused with p and X[P], not realizing the difference between the data in each position of the array and the index. There were some candidates who did not know what "initialisation" was.

Question 3

File access. Many answers lacking computer science were found to this question. Candidates mentioned the benefits to the users regarding DA. Many vague answers such as "it's very quick" were found.

Question 4

Many candidates did not include examples of real time and on line. They also gave answers using the same words they were trying to define.

Question 5

Most candidates could explain the difference between validation and verification.

Question 6

Data representation was successfully answered by all but the very weak candidates.

Question 7

Except for the candidates who used the word "compress" in their answers, all candidates showed no problems with data compression.

Question 8

Errors- straightforward with the exception of those who, again, could not find the words to explain "logic errors" or "syntax errors" without using the word "logic" or "syntax". Some were convinced that a division by zero was a logical error .

Question 9

Functions of primary and secondary memory was a well-answered question, though some did not realize that 2 functions were required for each type of memory (there were 4 available marks).

Question 10

Lots of different answers were given to the software life cycle question, and many candidates seemed to have read the question very quickly, as they did not take into account the words "clearly" and "before" in the question.

Section B

Question 11

Tracing an algorithm. Most candidates who answered this question did well. They successfully traced the algorithm and completed the table. They nevertheless were unable to produce answers to the level of detail required when explaining why the function needs to return a real result and when explaining what is meant by pass-by-reference. It seemed as if these candidates did not have the knowledge to present the answers to the level of detail required.

Question 12

This file-related question was a very popular one but not even the strongest candidates achieved high scores.

Almost all candidates were able to explain how to verify the input time (part a). In part b, many candidates did not relate the answer to computer science and were producing answers that involved the "privacy of the customer" and "security for the customer" so as to make sure no one else uses his account. In part c, candidates were unclear about linking files, and there was mainly a lack of detail in their answers. Very few candidates were able to correctly outline the type of file processing involved (part c ii). General answers like "sequential" or "direct" without further detail were found. Also, there were many candidates answering "batch processing" and the weakest ones were stating the file processing was "searching".

Question 13

LAN question. Quite a popular question with very good answers. Almost all candidates who answered this question included a correct diagram of the network and were able to state the topology (part a & b). Most candidates knew the role of the hub and correctly explained it (part c). While outlining the

advantages of a LAN (part d), some candidates did not relate their answers to the users, and just gave answers related to general features of Networks. No problems were found when outlining a security measure (part e) and when stating the hardware device needed to provide access to Internet (part f). Nevertheless, some candidates who answered "gateways" or "routers" did not know exactly what these did.

Question 14

This was probably the least popular question. No problems with stating types of software needed to access the service (part a), except for those candidates that just included commercial names. Almost all candidates were able to state batch processing (part b) and correctly elaborated their answer. Many candidates were able to outline the characteristics of the computer system needed (part c), stating servers or multi-user environment was needed. Nevertheless, the weakest candidates showed some confusion between multi-tasking and multi-user. No problems with the data restoring in case of failure (part d). Though most of the candidates understood what should be going on while counting the new users (part e), not very many were able to outline the "type of computer processing". Many answers lacking "computer science" were found.

Recommendations and guidance for the teaching of future candidates

- Candidates should be encouraged to read carefully the questions, to be precise in their answers and to make sure that they are including computer science in their answers. This can be attained through the use of past papers.
- Encourage candidates to give definitions without using the word that they are trying to define.

It is often found that candidates show a good understanding of basic concepts but when more technical knowledge is present, many of them are unable to make valid comparisons or find it difficult to produce answers to the level of detail required. Technical concepts mentioned in the Subject Guide should be studied in more depth.

Standard level paper two

Component grade boundaries

Grade:	1	2	3	4	5	6	7
Mark range:	0-11	12-22	23-29	30-35	36-41	42-47	48-70

General comments

The paper proved to be more difficult than in the previous year with few students achieving the higher grades. This can be put down, perhaps, to two factors: the lack of choice in the new paper format, and the increase in content of algorithm questions, a topic which prove difficult for many students at this level.

There are some schools entering students who clearly haven’t been taught the full Computer Science course, and are being entered as a result of studying an extended ICT course. The ability to construct and understand algorithms is fundamental to the course. Yet there are candidates who failed to score a mark on these questions. On the other hand, there are students who showed a great capacity for logical reasoning and produced some elegant solutions. The success or failure in this topic was school specific.

The strengths and weaknesses of the candidates in the treatment of individual questions

Question 1(Algorithm)

There was the complete range of marks. (a) and (b) dealt with basic programming theory (**signatures** and **scope**). More understood what information was contained in the signature than understood what was meant by scope. Scope to many incorrectly meant the range of values. Signature to some was the purpose of the algorithm. Scope is of particular importance when objects/classes are used, and the understanding should hopefully improve as students/schools get used to Java. It is clear that many candidates are constructing programs in their schools without a clear understanding of the theory underpinning their programs.

The trace of the given algorithm had mixed results. The two algorithms that the students had to construct were a fair test and provided quite a few good solutions. As pointed out above, there are certain schools whose students barely attempted the algorithms and were unable to formulate correctly even the most basic of structures. It remains a mystery how the same students presented complete dossiers for their internal assessment.

A common problem when using arrays was the confusion between the index of the last item in the array and the number of entries in the array.

Question 2

Prototyping (a) was answered quite well - there's an overlap here with the project where the students have to show examples of prototyping.

The use of sensors (b) was surprisingly poorly done, with many unable to describe a suitable sensor for the railway scenario. Many stated a particular sensor but then failed to describe how it would work. Repeating the question in the answer is a common fault at this level. Stating that 'the sensor would then capture the position of the train' is clearly not going to gain any credit.

Types of processing (c) was answered well by many, but the algorithm questions (d) and (e) again proved difficult. Many could not differentiate between data structures and data types (d), and consequently were lost when they came to constructing the algorithm (e). The actual algorithm was a good test for the top students with a few coming out with suitable solutions.

Question 3 (Case Study)

It was clear from the answers, which schools had actually studied the topic in depth and those who hadn't. Although questions are not going to be asked on aspects of the Case Study that are not in some way referenced in the Case Study itself, a prior understanding of the material provides students with an considerable advantage over others.

Candidates need to look closely at the action verbs used, and some schools need to prepare their candidates better in this aspect, as many candidates were answering in the same way (i.e. briefly 'stating') even when the action verb clearly indicted a deeper, more extended answer.

Several students did not understand why hexadecimal is used, believing that it in some way saves memory space.

Recommendations and guidance for the teaching of future candidates

The ability to understand and formulate algorithms is fundamental to this course, and the new paper format makes the avoidance of certain topics impossible. Therefore algorithms must be treated as an ongoing topic to be taught throughout the course, and not simply viewed as a by-product of the students' programming. With the algorithm content being increased, it is not possible to score high marks on this paper without being competent in this skill.

The way in which to answer action verbs need to be looked at carefully, as too many students are failing to gain full marks on certain types of questions ('explain', 'discuss') owing to the brevity of their answers.

