A Comparative Analysis of Results on Programming Exams

James Harland, Daryl D’Souza and Margaret Hamilton

School of Computer Science and IT, RMIT University
GPO Box 2476, Melbourne, 3001, Australia
{james.harland,daryl.dsouza,margaret.hamilton}@rmit.edu.au

Abstract

Measuring student performance on assessments is increasingly important, especially when mapping outcomes to particular topics in a university subject. In this paper we investigate the classification of exam questions. In particular, we examine the performance of students in two programming subjects, as a means of determining how we can measure the difficulty of a particular question. This can not only serve as a calibration for the expectations of instructors about the difficulty levels, but also as a means of examining what it means for a question to be considered difficult.

Keywords: BABELnot, Exam classification, Assessment measurement

1 Introduction

In Australia, there is a push to measure and compare educational institutions. School data from 10,000 schools around Australia has been published in the MySchool website and universities are being asked to be more accountable for their funding and to document academic standards clearly. Universities are being assessed both by our students and by our funding bodies.

The work reported in this paper contributes to the overall aims of the BABELnot project (Lister et al. 2012), which commenced in 2011. It is funded via a grant from the Office of Learning and Teaching (OLT), and, in a nutshell, aims to develop a common language in which educators of programming in ICT degrees may better communicate about assessments and standards within their subjects.

Programming has long been regarded as a learning bottleneck for novices, typically students entering their first semester of their ICT degrees. High failure and attrition rates are commonplace, and a lot of energy has been spent on research to understand the reasons but yet often the causes of such outcomes have been explained away on the basis of opinion and folklore (Sheard et al. 2009). The BABELnot project seeks to develop an epistemology or common understanding of programming concepts and ways of assessing competency across programming subjects.

In the School of Computer Science & IT at RMIT University, we have several programming subjects, varying from introductory ones to advanced ones (requiring two or three previous semesters of programming experience). All the subjects involve the learning of programming, but use Python, Java, PHP and C as the primary coding languages, and offer different outcomes for undergraduate or postgraduate students, as well as for various other students taking a programming stream. For instance, a student may study multimedia, or games, or engineering, all of which have degree programs which contain a programming stream consisting of three programming subjects. Also, a student may be undertaking an undergraduate program of three years, or a postgraduate program of one year and still require a knowledge of programming for their ICT qualification. Hence the outcomes required by students when undertaking the programming stream differ immensely.

The context of each subject makes it very unclear where to ask to insert various exam questions. Another factor is the readiness of the subject manager to allow exam questions from a different origin. The learning outcomes of the various subjects are sufficiently different to introduce confusion about whether or not a particular question will be testing that outcome. Finally the mixture of concepts required to answer the question makes it difficult to be able to place it within the context of any particular subject.

However, despite all of the above difficulties, we approached all the subject managers and asked if any would be interested in putting any of our BABELnot exam questions into their exam papers for that semester. Two lecturers agreed, and their answers provide the results which we discuss here in this paper.

Several issues arise. The stated outcomes and capabilities in subject guides are typically vague, a problem being addressed elsewhere in the BABELnot project, and often lead to dichotomous teacher-student perceptions about the questions in exams. Teachers will of course believe that they are using the summative assessment instrument appropriately, and more often than not their attempts are genuine and sincere in attempting to meet the broadly expressed outcomes.

On the other hand, and particularly at the introductory programming stage, students do not necessarily handle summative assessments as their lecturers expect them to (Shuhidan et al. 2010, Tew and Guzdial 2010). Typically high failure rates demonstrate that a significant number of students find exams hard (Tew and Guzdial 2011).

In this paper we investigate how we may use student performance on exam questions as an indicator of the difficulty of the question. In particular, we wish to be able to relate the Degree of difficulty measure used in the BABELnot classifications (Sheard
et al. 2011) to student performance, and hence improve the accuracy of our classification. There are a number of measures used to classify exam questions, many of which come with some associated criteria for their application (Simon et al. 2012). The Degree of difficulty is intended as a holistic measure, and hence does not come with any specific criteria. This is also considered the measure most likely to correlate with student performance. Hence our focus is on how we may measure the difficulty of a question by the students’ performances on it, and how this relates to the BABELnot classifications.

Another aim is for a standard approach to the reporting of exam performance, and in particular the statistical information that will best provide for the classification of questions. There are many issues to be considered in the development of such a standard (such as those discussed by Anfoff (Angoff 1971)), and the approach reported in this paper is not necessarily going to become a standard one, but it should be seen as a starting point for discussion.

This paper is organised as follows. In Section 2 we discuss the background to our work, and the BABELnot project. In Section 3, we describe the two subjects used in our research, and our approach to analysing the exam results. In Section 4 we present our data, and in Section 5 we discuss the implications of it. Finally in Section 6 we present our conclusions and some areas of further work.

2 Background

As indicated earlier this work contributes to the BABELnot project, which was funded in 2011 by the the Office of Learning and Teaching (OLT) (http://www.olt.gov.au). The BABELnot project held its inaugural major meeting in October of 2011 in Melbourne and is funded through to August 2013.

The project was prompted by a need to document academic standards associated with a sequence of up to three programming subjects in six participating universities (UTS, QUT, Monash, RMIT and the universities of Sydney and Newcastle) and optionally other universities as well. In order to meet this objective, two broad subgoals were presented. One was to “develop a rich framework for describing the learning goals associated with programming” and the other was to “benchmark exam questions that are mapped onto this framework”. Further details about the rationale for the project are documented in Lister et al (Lister et al. 2012). The project unified several projects in existence at the time of commencement: Exam Question Classification, Syllabus Specification, Exam Question Generation and Benchmarking, and Neo-Piagetian Theory (and its application to the learning of programming).

This paper contributes to the Exam Question Generation and Benchmarking component of the BABELnot project. A significant precursor to this component of BABELnot was the BRACElet project (Clear et al. 2010). The aim of BRACElet was to collect and analyse the exam results, and hence inform our classification of exam questions, and some details of the two chosen subjects.

3 Exam Questions

In this section we describe our approach to the analysis of exam questions, and some details of the two chosen subjects.

3.1 Methodology

There are a number of ways in which exam results could be analysed (Angoff 1971, de Klerk 2008). Our intention is to inform our classification of exam questions by student performance results, and in particular to see how student perceptions of difficulty align with the marks obtained by the students on particular questions or groups of questions.

In order to do this, we first look at the overall grade distribution. This gives us some insight into the overall difficulty of the exam, as well as the distribution of ‘student types’, i.e. how many high-achieving students there are compared to those who barely pass. The most obvious categorisation of student types is by the grades they achieve, and so we will classify students according to their grades, rather than a finer-grained scheme (e.g. the percentage decile in which their mark falls) or a coarser-grained one (e.g. whether they passed the exam or not).

The relevant grades are given in the table below.

<table>
<thead>
<tr>
<th>Grade</th>
<th>Mark</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD</td>
<td>≥80%</td>
<td>High Distinction</td>
</tr>
<tr>
<td>DI</td>
<td>&lt;80% and ≥70%</td>
<td>Distinction</td>
</tr>
<tr>
<td>CR</td>
<td>&lt;70% and ≥60%</td>
<td>Credit</td>
</tr>
<tr>
<td>PA</td>
<td>&lt;60% and ≥50%</td>
<td>Pass</td>
</tr>
<tr>
<td>NN</td>
<td>&lt;50% and ≥25%</td>
<td>Fail</td>
</tr>
<tr>
<td>FF</td>
<td>&lt;25%</td>
<td>Fail</td>
</tr>
</tbody>
</table>

FF is not actually a separate grade from NN. However, we separate the data here for analysis purposes (and specifically to try to isolate ‘genuine’ failures
from those who made no serious attempt). As it turns out, there were comparatively few of these students (4 out of 236 in P1, and 6 out of 160 in PT), and hence their effect on our results is minimal.

We then investigate the students’ performance on each question or question type. Firstly, we look at the mean and median marks for each question (and the mode, if applicable). This gives us some indication of the performance of the students on each question, as well as the range of marks. For example, there are some questions in which the mean mark is well below the median, which indicates that there is a wide spread of marks (as if at least 50% of the students scored better than the average, then the lower scoring students must have scored generally very low marks). In cases where the median mark is 100%, this indicates that at least half the students got full marks.

Our next two analyses are similar in spirit, but differ on some detail. One analysis looks at performance on each question by the different classes of students, as classified by their grades. Hence we look at how the HD students performed on each question or question type, as measured by the average mark obtained on the question by the HD students. We then perform the same analysis for the DI students, the CR students and so on. This effectively gives us a profile of the performance of a ‘typical’ HD, DI, CR, PA, NN or FF student, and in particular how this question can be rated by the students’ performance on it (e.g. “The best students generally struggled with this question, while the weaker students found it very difficult”).

The other analysis is to consider the number and range of marks obtained for each question. We do this by looking at the number of students who scored 80 or more on this question, and we label these students as HD, those who obtained between 70% and 80% as DI, and so forth, giving us an idea of the distribution of the performance of the overall student population on each question. This will help inform us about the rating of questions by allowing us to draw conclusions such as “Very few students got full marks for this question”, or that “Most students got at least 50% on this question”.

Hence the difference between these two latter analyses is that the former divides the student population into classes, and then looks at the performance of each class on specific questions, whereas the latter one looks at the spread of student marks for each question.

### 3.2 Subjects

Both of the subjects whose results are presented here were taught in semester 1, 2012 at RMIT University.

**Programming 1** is a first programming subject in Java, with no previous knowledge of programming necessary. There were 236 students in semester 1, 2012 who sat for the exam in this subject. In BABELnot terms, this is a level 1 subject.

The Programming 1 exam consisted of three main parts:

1. 20 multiple choice questions (30 marks)
2. 6 short answer questions (35 marks)
3. 3 interrelated programming problems (35 marks)

We will refer to each of these three parts as **MCQ**, **Short** and **Classes** respectively.

**Programming Techniques** assumes some significant programming experience, and specifically two previous semesters of programming in Java. This subject is taught in C, and 160 students sat for the exam in semester 1, 2012. In BABELnot terms, this is a level 3 subject.

The Programming Techniques exam consisted of

1. 7 short answer questions (35 marks)
2. 9 programming problems (145 marks)

### 4 Exam Results

In this section we discuss the results for Programming 1 and Programming Techniques.

#### 4.1 Programming 1

The results for Programming 1 are presented in Figures 1, 2, 3 and 4.

The graph in Figure 1 (`P1 Grade Distribution`) shows the overall distribution of grades. This conforms to a typical pattern for introductory programming subjects, i.e. showing some bipolar tendencies. 70 students out of the total of 236 (or almost 30%) got a grade of HD, whilst around 16% failed. Of the passing grades, PA had the lowest proportion of students at 10%.

![Figure 1: Programming 1: Overall grades](image1.png)

![Figure 2: Programming 1: Mean & Median](image2.png)
The graph in Figure 2 (‘P1 Mean & Median’) shows the mean and median mark on each of the three sets of questions (the mode mark for all three was zero). One point to note is that the average mark on all three question sets was above 60%. Given that the median is higher than the mean on all three question sets, this is some evidence of a wide spread of marks, as if at least 50% of the students scored better than the average, then the lower scoring students must have scored generally very low marks.

It is perhaps a little surprising that the highest average mark is for the Classes questions, which are the ones that would be expected to be the most difficult. This is probably explained by the fact that the Programming 1 students were informed in advance of the scenario on which the questions would be based, and the students were given code skeletons which they had to complete. This was a new approach compared to previous years, in which students were not informed in this way, and they had to write complete pieces of code.

The graph in Figure 3 (‘P1 Student type performance’) shows the performance of the students, classified by their grade, against the three question types and their overall results. In other words, the students who obtained an HD grade scored an average of just over 80% on the MCQ questions, just under 90% on the Short questions and just over 90% on the Classes questions, whilst averaging 88% overall. Those who obtained a DI grade had a similar pattern of also performing best on the Classes questions. The CR and PA students were at their worst on the Short questions, unlike the HD and DI students, for whom the MCQ questions were their worst performance. This is a little counter-intuitive, in that one may expect the better students to perform best on all questions; further analysis of this data and comparison with similar exams are some items of future work. Less surprisingly, the NN and FF students performed best on the MCQ questions.

An interesting property of the Classes questions is that they appear to have an “Eden-Monaro” property, i.e. that performance on this question mirrors accurately the students’ performance on the exam overall. This property is that the students who obtained HD overall scored at least 80% on this question, those who obtained DI scored between 70% and 79%, those who obtained CR scored between 60% and 69%, those who obtained PA scored between 50% and 59% and those who failed the exam scored less than 50%. This is perhaps not altogether unsurprising given that the Classes questions are worth 35 marks out of a total of 100, but it is also worth noting that neither of the other two question sets, worth a total of 65 marks overall, had this property.

The graph in Figure 4 (‘P1 Student performance’) shows the performance of the students on each question. Hence nearly 100 students got 80% or more on the Classes questions, with just under 60 students getting 80% or more on the MCQ questions. This shows perhaps most starkly that the students performed best on the Classes questions.

In order to compare the effectiveness of the three question sets (MCQ, Short, Classes) as means of classifying students, we performed a χ² test comparing performance on the individual question sets compared with the students overall performance. This returned Pr values of 0.027, 0.072 and 0.032 for the sets MCQ, Short and Classes respectively. This is in some ways a biased test, in that each class component forms a part of the overall performance, but interestingly only the Short class returned a Pr value over the traditional threshold of 0.05 for statistical significance. From this we conclude that the Short class of questions was a more accurate prediction of the students’ overall performance than either of the other two question sets.

We also performed similar tests comparing each pair of question sets. The only one of these three pairs to return a non-zero Pr value was for Short and Classes, for which the value was 0.067. This suggests that there is some correlation between performance on these sets of questions, but that the performance on MCQ does not reflect performance on the other sets.

4.2 Programming Techniques

The results for Programming Techniques are presented in Figures 5, 6, 7, 8 and 9. The graph in Figure 5 (‘PT Grade Distribution’) shows the overall distribution of grades. This conforms to a typical pattern for advanced programming subjects, with a large number of students getting a grade of HD. This is at least partly due to students
having to have completed two prior semesters of programming before entering this subject, and so a relatively advanced level of programming ability is a pre-requisite.

The graph in Figure 6 (‘PT Mean, Median & Mode’) shows the mean, median and mode for each question. Note the scale of the graph, in which the lowest value shown is 50%, reflecting that the mean, median and mode for each question was at least 50% (and in fact the lowest mean value was 57%). From this it seems reasonable to conclude that Question 4 was the one the students found easiest, as it had the highest mean (of 90%) and with the median and mode both being 100%. Questions 1, 3, 4, 7 and 14, being those for which the median and mode are 100% could also be argued to be ones on which performance showed some variability, with Question 1 (the one with the lowest mean) the most varied of all. Questions 8 and 6, in contrast, had the most uniform performance, with the mean, median and mode marks all virtually identical. It is also notable that Questions 2 and 10 are arguably the most difficult, with the two lowest median marks. The high modes for each of these questions suggests that the distribution is bi-polar rather than spread, i.e. that most student either got most of the marks, or very few marks.

The graph in Figure 7 (‘PT Student type performance’) shows the performance of the students, classified by their grade, against all questions. The HD students, as expected, were the best performed students on all questions. However, the questions on which the average marks were lowest were Questions 1 and 2. It is clear from this graph that Question 4 can be considered the easiest question, as every class of student scored 80% or more for this question. The fact that the CR students performed slightly worse than the PA, NN and FF students is presumably due to noise, rather than indicative of anything more significant. It is also noticeable that the only questions on which NN or FF students scored an average of 50% or more were Questions 4, 5, 6 and 9, which also suggests that these were easier questions than the others. Question 14 is also arguably an easier question, given that the HD students got an average mark of 90% on it, and it was also the question on which the PA students obtained their second-highest average mark (after Question 4). However, the NN and FF students didn’t score particularly well on this question. As the PA students had an average mark of 75%, this question seems to have a ‘polarising’ quality, in that students who passed the exam generally did well on it, but those who failed tended to score poorly on it. Similar comments apply to Questions 3, 8, 11, 12 and 13. Question 7 is almost in the same class, except for the rather puzzling ‘reversal’ of performance of the DI and CR students on this question (the DI students averaged 49%, the CR students 75%).

Question 15 is arguably a moderately challenging question, in that the only classes of students with an average mark of 50% or more were the HD and DI students. This has a similar polarising effect, but this time in separating the better students from those who barely pass or fail. Question 10 is similar. Questions 1 and 2 are similar, although the division between the HD students and the DI students is starker, and the performance of the CR and PA students is more variable on Questions 1 and 2, making them seemingly less reliable polarisers. Question 16 could also be considered a polarising question, in that the HD, DI and CR students all performed relatively well, but with the PA students scoring an average of 47%.

Question 11 has the Eden-Monaro property mentioned above (i.e. a students performance on this
question accurately reflects his or her overall mark). Questions 3 is similar, although not quite perfect given that the CR students have an average mark of 70%.

Question 1 clearly had a polarising effect, in that almost all students got either over 80% or under 25%. Questions 3 and 7 are similar, without being quite as extreme, and if one considers the HD and DI population together, Questions 5 and 6 also had this effect. One could also argue that Questions 1, 7, 3, 15 and 2 were the most difficult, on the grounds that these were the questions with the greatest number of students who scored in the FF range.

We also performed $\chi^2$ tests for each of Questions 1 to 16 compared to the overall result. The only values that were not 0 or 0.001 were those for Questions 8, 11 and 13 with values of 0.046, 0.046 and 0.527 respectively, of which only the value of 0.527 for Question 13 is above the threshold value of 0.05 for significance. Hence student performance on Question 13 seems to be a very good predictor of overall performance, with Questions 8 and 11 not at the same level, but perhaps having some indication of ability, especially when compared to all of the other questions.

5 Discussion

As mentioned above, we are interested in investigating the Degree of difficulty measure. This is intended as a holistic measure, and one that should correlate the most with student performance (although this measure could and should correlate to some degree with other measures, such as Intellectual complexity and Conceptual complexity). We may see the results discussed above as a measure of the Degree of difficulty more than anything else. Given that the rating is one of the three values low, medium and high, it would seem reasonable to use the above data to determine a way of classifying questions on this scale according to student performance (such as an average mark of 80% indicating that the question is of low difficulty).

Turning to the Programming 1 result discussed above, it seems that measuring Degree of difficulty by looking at the average mark on a question is too simplistic; it is important to look at the spread of results and whether the question is ‘polarising’ students or not. In relative terms, Classes was the one the students found easiest, with MCQ next and then Short. The better students (HD, DI) tended to score better marks on Short than on MCQ (a situation that is reversed for every other type of student), but it is arguable that this is a property of more difficult questions. Certainly a question on which the top students struggle to get more than 50% would be considered difficult; is a question difficult if the top students do well, but other students struggle? In the case of Short, it would seem that this would be considered medium, as it doesn’t seem to be low, and to class it as high when nearly half the students score more than 70% doesn’t seem right. This seems to leave the only option for MCQ to be also medium, as it is hard to argue that it is more difficult than Short, but calling it low seems to contradict that this was the question on which the HD and DI students scored least well. This would make Classes either low or medium, depending on how significant one feels the Eden-Monaro property is (with greater significance indicating a stronger likelihood that it is medium).

Turning to Programming Techniques, it seems clear that Questions 4 and 6 should be classified as low (if for no other reason that the FF students scored nearly 90% on Question 4 on average, and over 50% on Question 6). One could also argue that Question 5 should be rated as low, as on this question, the
NN students scored more than 50% on average (and Questions 4, 5, 6 and 9 were the only ones with this property). On the strong overall performance of the students, it would seem a long bow to draw to classify any of the question as high. The hardest questions, according to the mean mark, were Questions 1, 10, 2 and 15 respectively, and on these questions the HD and DI students generally did well, whilst the others struggled. This is analogous to the Short question on Programming 1. Note also that there is a polarising effect here, strongest in Question 1, but present to some degree in Questions 10 and 15. Hence it is tempting to classify Questions 4, 5,6 and 9 as the easiest group, Questions 1, 2, 10 and 15 as the hardest, and the remaining 8 questions somewhere in the middle. It seems hard to argue that Questions 1, 2, 10 and 15 are of high difficulty, and at least as hard to argue that they are low. It seems reasonable to argue that Questions 4, 5 and 6 (and possibly 9) are low, but this would make all others (in this case at least 12 out of 16 questions) medium.

Questions 7 and 12 on the Programming Techniques exam were drawn from the BABELnot repository, as were Questions 5, 11 and 12 from the MCQ questions on the Programming 1 exam. In fact, Questions 7, 11 and 12 on the Programming Techniques exam were basically the same, only that the Programming 1 version was scaffolded to a much greater degree, and the students only had to select which lines of code had to be changed. The Programming Techniques students had to write complete (C) code for the specified function.

We have not presented the data for the individual parts of the MCQ questions, and so it is hard to make a direct comparison between performance on the Programming 1 version of the question compared to the one for Programming Techniques. However, we note that Question 12 on the Programming Techniques exam had a mode of 100%, an average mark of 74% and a median of 86%. Moreover, the passing students (i.e. the HD, DI, CR, and PA students) all did well on this question, whilst the NN and FF students scored an average of 42% and 7% respectively. Hence, whilst it may not be the easiest question on the paper, it was in the easier half, at least. It is also worth noting that 49% of the Programming 1 students got the correct answer for MCQ Question 11. This lower level of performance together with the simpler nature of the question being asked tends to indicate that the Programming 1 students found this question considerably more difficult than their Programming Techniques counterparts, as would be expected.

6 Conclusions and Further Work

We have discussed the exam results of two programming subjects, one of which assumes no programming background, and the other requiring two semesters of programming experience. We have seen how attempting to align student performance on these exams with the BABELnot system of difficulty levels is not appropriate for classifying questions (or at least not for classifying questions based on students’ results). The underlying issue is that this measure is attempting to summarise student performance in an overly simplistic way; it seems that the distribution of students’ marks contains some features which are not readily apparent in a three-point scale.

Perhaps the most intuitive way to characterise the difficulty of questions is to look at student types, and to determine the difficulty of the question relative to the performance of each student type. For example, if the weakest students (as measured by overall performance on the exam) get an average of 90% or more, it seems inescapable that the question is of low difficulty. However, when the performance of each student type diverges, it can be more problematic. For example, if half of the student get full marks and the other half get 0, how difficult is the question? It would seem that it is better to classify such a question as a ‘perfect polariser’ rather than being of a particular level of difficulty.

It certainly seems that the only way to make progress on issues like these is to continue analyses of this sort on an increasingly large set of data. This would allow more detailed and specific criteria to emerge, and possibly also a more refined understanding of the notion of difficulty.

One item of future work is to perform a further analysis of the data from these two exams, such as measuring the internal consistency of the questions in each exam, as suggested by de Klerk (de Klerk 2008), using inter-correlations as measured by Cronbach’s Alpha. Another dimension discussed by de Klerk is the validity of testing, which in our case corresponds to determining the appropriate level of content for level 1 and level 3 programming exams. Another possibility is to determine the difficulty of a given question in comparison to a normative sample, or standard group for comparison purposes. Both of these aspects will be informed and enhanced by the work of the BABELnot project.

References


Lister, R., (2011), Concrete and other neo-Piagetian forms of reasoning in the novice programmer, Thir-


Appendix: Programming Techniques Exam Questions

QUESTION 7
In one sentence, explain the purpose of the following piece of code.

```c
int fn(int *array, size_t n) {
    size_t i;
    for (i = 0; i < n - 1; i++) {
        if (array[i] > array[i + 1])
            return -1;
    }
    return 1;
}
```

QUESTION 12
The purpose of the block of code below is to take an array of integers and move all elements of the array one place to the right, with the rightmost element moving around to the leftmost position.

```c
void shift_right(int *array, int n) {
    int temp = array[n - 1];
    int i;
    for (i = n - 2; i > = 0; i--)
        array[i + 1] = array[i];
    array[0] = temp;
}
```

Write a function `shift_left` that will move all elements of the array one place to the left, with the leftmost element moving around to the rightmost position.

```c
void shift_left(int *array, int n) {
    /* . . . */
}
```
QUESTION 5  (1.5 Marks) Consider the following method:

```java
boolean testArray(int [] x, int arrayLength) {
    for (int i = 0; i < arrayLength-1; i++)
    {
        if (x[i] > x[i+1]) return false;
    }
    return true;
}
```

If `testArray` returns `true` after this code is executed, which of the following is the strongest statement we can make about the contents of the array `x`? Assume the subscripts `p`, `p+1`, and `q` are legal indexes of `x`.

A) `x[p] <= x[q]` for all `p < q`.
B) `x[p] <= x[p+1]`
C) `x[p] <= x[p+1]` where `p` is an even number (i.e. 0, 2, 4, etc).
D) No statement can be made, as the last iteration of the loop will attempt to index an element of the array that does not exist.
E) `x[0] < x[arrayLength-1]`

QUESTION 11 (1.5 Marks)

The code below in the right of the table moves all elements of the array `x` one place to the right, with the `rightmost` element being moved to the `leftmost` position. The variable `length` contains the number of elements in the array `x`:

<table>
<thead>
<tr>
<th>Line</th>
<th>left</th>
<th>right</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>int temp = ???</td>
<td>int temp = x[length-1];</td>
</tr>
<tr>
<td>2</td>
<td>for (int i ???)</td>
<td>for (int i=length-2; i&gt;=0; i--)</td>
</tr>
<tr>
<td>3</td>
<td><code>x[i-1] = x[i];</code></td>
<td><code>x[i+1] = x[i];</code></td>
</tr>
<tr>
<td>4</td>
<td>?? = temp;</td>
<td><code>x[0] = temp;</code></td>
</tr>
</tbody>
</table>

Consider the partial code provided in the above table, on the left. When the occurrences of `??` are replaced with appropriate code, that code can undo the effect of the code on the right. That is, when the `??` are replaced appropriately, the code can move all elements of the array `x` one place to the left, with the `leftmost` element being moved to the `rightmost` position.

The table already shows that line 3 is different in the code on left to the code on the right. When the `??` on lines 1, 2 and 4 are replaced with appropriate code, which of these lines of code must be different between the left and the right?

A) None of the lines 1, 2 and 4 must be different.
B) Only lines 1 and 4 must be different.
C) Only line 2 must be different.
D) All of the lines 1, 2 and 4 must be different.
QUESTION 12

(1.5 Marks)

Below is incomplete code for a function which returns the minimum value in the array \( x \). When an appropriate line of code is selected from each box, the completed code will scan across the array, using the variable \( \text{minsofar} \) to remember the best candidate for minimum so far.

```c
int min(int x[], int arrayLength) {
    int minsofar = \[A\] 0 \[B\] x[0];
    for (int i = 1 ; i < arrayLength; i++){
        if ( x[i] < \[C\] minsofar \[D\] x[minsofar] )
            minsofar = \[E\] i \[F\] x[i];
    }
    return \[G\] minsofar \[H\] x[minsofar];
}
```

Which of the following choices of lines from the boxes will produce a correct version of the function \( \text{min} \)?

A) Lines ACFG only.
B) Lines ADEH only.
C) Lines BCFG only.
D) Both lines ADEH and lines BCFG.