Benchmarking a set of exam questions for introductory programming

Judy Sheard
Monash University
Australia
judy.sheard@monash.edu

Simon
University of Newcastle
Australia
simon@newcastle.edu.au

Julian Dermoudy
University of Tasmania
Australia
julian.dermoudy@utas.edu.au

Daryl D'Souza
RMIT University
Australia
daryl.dsouza@rmit.edu.au

Minjie Hu
University of Otago
New Zealand
minjie.hu@otago.ac.nz

Dale Parsons
Otago Polytechnic
New Zealand
dale.parsons@op.ac.nz

Abstract

This paper reports on the combining of two related but hitherto distinct themes in programming education research. The first is the recognition that students in programming courses tend to perform far more poorly than their teachers would like, and further, more poorly than their teachers would expect without a careful analysis of their results. The second is the proposal of a number of different styles of examination question, sometimes coupled with analysis of student performance on those questions, typically at single institutions. This work combines these themes by including a common set of short questions in the final examinations of introductory programming courses at six institutions in Australia and New Zealand, and analysing the student performance across all six institutions. The analysis results in a set of four simple questions that can be used to benchmark student performance in introductory programming courses at a wide range of institutions.

Keywords: Standards, quality, examination papers, CS1, introductory programming, assessment.

1 Introduction

While researchers have long observed that students have difficulty learning to program (e.g. Pea (1986); Soloway et al (1982)), the McCracken study (2001) brought broader attention to the issue by establishing that the problem is not constrained to individual courses in individual institutions but is widespread, and can come as a surprise to the academics teaching the courses in question.

An extreme interpretation of the work of the McCracken group is that many programming students cannot program. Yet it is clear that many students pass programming courses, so a number of researchers and projects have subsequently set out to examine more closely the assessment in programming courses, to try to establish how the assessment of programming aligns with the skills and knowledge that students are expected to acquire. Researchers working in this area include Simon et al (2010), Petersen et al (2011), and Sheard et al (2013).

In recent years, moreover, increasing attention has been paid to standards in higher education, with government bodies such as Australia's Tertiary Education Quality and Standards Agency (2012) charged with evaluating the performance of higher education providers against a new Higher Education Standards Framework. The combined discipline group for Information and Communication Technology (ICT) and Engineering has begun its quest for learning standards by drawing on existing learning outcomes developed from the relevant professional bodies (Cameron & Hadgraft, 2010).

The work reported here forms part of the BABELnot project (Lister et al, 2012), a principal goal of which is to explore a possible approach for the development and assessment of learning standards in programming courses. Formal written examinations are a common form of assessment in programming courses, and typically the form to which a large portion of marks are attached. In the work reported in this paper we have developed and tested programming assessment questions across multiple institutions with the aim of establishing a set of questions that can be used as a benchmark against which student performance can be measured.

2 Benchmarking

Benchmarking is the process of evaluating performance against an established standard. Benchmarking of student performance is often undertaken in universities and the results are used in a number of ways at the course, department, and institution level.

Benchmarking of student performance is an important tool for quality assurance. It may be used to establish minimum or acceptable levels of performance in a course or to compare course performance across cohorts or institutions (Woolf et al, 1999). The results of benchmarking may be used for marketing purposes or to make strategic decisions at a department or institution level.

Benchmarking may also be used to make improvements to a course or program of study. It can be used in evaluation of teaching approaches, resource provision and student selection. An example of the use of benchmarking in the computing education discipline is Oliver's (2000) work on developing benchmarks for IT literacy.

There are other benefits of benchmarking exercises. The development of a benchmark requires a community effort to examine and determine appropriate criteria and standards. The communication and collaboration required serves to strengthen the community. Sim, Easterbrook and Holt (2003) propose that "benchmarking, when embraced by a community, has a strong positive effect on the scientific maturity of a discipline" (p.74).

2.1 Benchmarking programming performance

There is a large corpus of research on the learning and teaching of introductory programming. Many studies have attempted to determine factors which influence learning outcomes. Although there are many positive findings, most studies are conducted in the context of a single course, making it infeasible to conduct comparisons across different institutions or different research studies. An analysis of 164 papers reporting research into the learning and teaching of programming found that 72% of the studies were conducted within one institution and most of these were within a single course (Sheard et al, 2009).

Several studies have investigated student performance in programming across multiple and international institutions. These include work by McCracken et al (2001), Lister et al (2004) and the BRACElet project (Clear et al, 2009/2010; Whalley et al, 2006). However, these studies have focused more on investigating performance of students to understand what and how they learn rather than the development of suitable questions which could be applied more widely for future benchmarking exercises.

The BABELnot project (Lister et al, 2012) has proposed a number of possible examination questions, along with a scheme for classifying questions according to a number of distinct criteria (Sheard et al, 2013). Harland et al (2013) examined the performance of students in two programming courses, with the aim of determining how the criterion of *degree of difficulty* might be measured for particular questions. The attempt was to establish 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. On the basis of BABELnot-designated exam questions in just two exams they concluded that the absolute scale of *low*, *medium* and *high* is not appropriate for classifying questions (or at least not for classifying questions based on students' results). By contrast, however, Simon et al (2012) established that researchers trained in the classification system can accurately assess the difficulty of examination questions on this same three-point scale.

The work in this area has been hampered by a scarcity of standard test questions that can be used to measure performance. One widely used example is Soloway's 'rainfall problem' (Soloway, 1986). Since the development of this question in the early 1980's, it has been used in a number of comparative studies of student performance. However, it has recently been suggested that this question is not suitable for current use (Simon, 2013).

The aim of our benchmarking study was to develop a set of questions that could be used by introductory programming educators at multiple institutions as a standard to measure the performance of their students.

3 Research approach

The work presented here was conducted as a follow-up to a workshop held in conjunction with the Australasian Computing Education conference in 2013 (ACE2013). The "Writing a good exam for a programming course" workshop was held as part of the BABELnot project (Lister et al, 2012). The aim of the workshop was to explore different styles of questions used for assessment of introductory programming students. A planned outcome was a set of introductory programming exam questions suitable for use in a benchmarking exercise across multiple institutions.

In preparation for the workshop, members of the BABELnot project team compiled a list of questions that were considered possible candidates for the benchmarking exercise. A total of 76 questions were sourced from exam papers from five institutions. Five project members then individually assessed the suitability of each question according to the criteria:

1. Is the question likely to be used by others?
2. Is there a clear marking guide? (to help achieve consistency in marking for benchmarking)

Based on the results of this assessment the list was trimmed to 34 questions.

Prior to the workshop the registrants were sent the set of 34 questions and asked to assess their suitability for use in an introductory programming exam. For each question they were asked to choose one of three options:

1. I would like to use this question
2. I would consider using this question
3. I would not use this question

Twelve people responded to the survey. Their responses were collated and the questions sorted according to their popularity.

Seventeen people participated in the workshop. All had taught or were currently teaching an introductory programming course. During the workshop the results of the survey were presented to participants. The idea was to provoke discussion about what makes a good or poor programming exam question and what would make a question unsuitable for use across multiple institutions. First, the least popular questions were presented and participants discussed potential issues with these

questions. Next, the most popular questions were presented and participants gave reasons why they would use these questions and any possible issues with their use. Finally, the remainder of the questions were discussed. This middle group were the most controversial as there were no clear decisions as to their suitability.

There were a variety of reasons why questions were deemed unsuitable for use in a multi-institutional benchmarking exercise. Issues identified:

- Question is too easy
- Question is too large
- Topic is too advanced or not usually covered in a typical introductory programming course.
- Student may not be familiar with the style of question.
- Style of question is not suitable for an exam situation, e.g. is it reasonable to ask students to identify syntax errors?
- Wording of the question is unclear or ambiguous.
- Question is idiosyncratic, e.g. referring to the coding style guide of a particular course.
- Question involves tricky code, which may obfuscate its purpose.

When all the questions had been discussed, participants were asked to reconsider the questions and vote for questions that they would be prepared to use in a benchmarking study. From the results of the voting a set of 11 questions were chosen. The final selection aimed for a spread across topics, question styles, skills required to answer the question and the type of response required from the student. A brief description of the questions is shown in Table 1.

Question	Style	Skill required	Open/closed response
Expressions	MCQ	Trace code	Closed
Assignment & Sequence	MCQ	Trace code	Closed
Selection A	Short response	Trace code	Closed
Selection B	MCQ	Trace code	Closed
Selection C	Short response	Explain code	Open
Iteration A	MCQ	Trace code	Closed
Iteration B	MCQ	Explain code	Closed
Iteration and Arrays A	Short response	Trace code	Closed
Iteration and Arrays B	MCQ	Explain code	Closed
Iteration and Arrays C	Short response	Write code	Open
Iteration and Arrays D	Code writing	Modify code	Open

Table 1: Questions for benchmarking study

At the end of the workshop a benchmarking study using the 11 questions was proposed, and six participants from four Australian and two New Zealand institutions agreed to continue with the study.

3.1 Benchmarking study

The benchmarking study was conducted during first semester in 2013. Each person obtained ethics approval from their institution and arranged for the questions agreed upon at the workshop to be placed in their final exam. The order of the questions and the marks allocated to each question were decided by each participant. However, to enable valid comparison of results the questions were kept in the same style. An exception to this was at one institution, where two questions were converted from short-answer to multiple-choice format. The responses for these questions from this institution have been omitted from the analysis.

The questions were originally designed in Java and were converted to C# or Python as required. This required minimal changes to the presentation of the questions. The greatest change required is that with one multiple-choice question concerning iteration, the obvious off-by-one error in the Java implementation was one less than the actual number of iterations; but when this was translated to Python, the obvious off-by-one error was one more than the actual number of iterations. This was accounted for by a change in one of the incorrect multiple-choice options, and a change in the order of the options.

At the end of semester the students' responses and marks gained for each question were collated for analysis.

3.1.1 Course profiles

A profile of the programming courses used in the study is shown in Table 2. All courses taught introductory programming, with five at the undergraduate level and one at the postgraduate level. The latter course is effectively the same as courses taught to first-year undergraduate students, but is taught to students who are taking a postgraduate computing qualification to supplement a degree in some unrelated area.

The language of instruction, IDE, and programing paradigm approach varied across the six courses in the study. However, all covered the basic programming constructs (expressions, assignment, sequence, selection, iteration and arrays) that are covered in the questions used in the study.

3.1.2 Exam profiles

A profile of the exams from which the data was collected for the benchmarking study is shown in Table 3. All exams were written and held at the end of the course. The exams comprised from 30% to 50% of the total assessment marks in their respective courses. The questions are worth 110% of the exam for course 5 because the exam consisted of just these questions, one of which was marked as a bonus question.

Course	Country	Level	Programming language	IDE	Teaching approach
1*	NZ	U/G	C#	Visual Studio 2010	Procedural
2*	Aus	U/G	Java	Eclipse	Objects later
3	Aus	P/G	Java	BlueJ	Objects first
4*	Aus	U/G	Java	JCreator	Objects early
5	NZ	U/G	C#	Visual Studio 2012	Objects later
6*	Aus	U/G	Python	JES	Objects later

Table 2: Course profiles, with asterisks indicating the four courses that were used in the final benchmarking

4 Results

This section presents the results of our benchmarking study. Data of student performance on the set of 11 questions was collected from six institutions. The number of exam papers from each institution varied from 13 to 297, as shown in Table 4.

The first step in the benchmarking exercise was to establish which institutions to include in the analysis. Our aim was to find questions upon which students over multiple institutions showed similar performance. For the benchmarks to be useful it is important that they can be used by other institutions and so we wanted to compare student results across the courses and exams with similar profiles.

To assist in this process we conducted a preliminary inspection of the student performance results. The graph in **Figure 1** of the percentage of correct responses to the closed questions (questions for which there was only one correct response and therefore no interpretation needed by the marker) shows that the results from two institutions appear to be outliers. The postgraduate group results (course 3) were higher than the other results for all but one question. Further inspection showed that the marks for each open question were also higher than for the other questions. Including the postgraduate cohort would make it less likely that we would find questions with similar results. It was therefore decided not to include this group.

At the lower end of the graph, the results for course 5 were lower for most questions and substantially lower for several. This course has a very small enrolment, and is usually assessed solely by practical assessments. The exam was included expressly for this research, and the students were not accustomed to written exams. In the circumstances, it seemed appropriate to omit this course, too, from the analysis.

The benchmarking analysis, conducted to find questions on which there was no significant difference across comparable course, was therefore conducted on the remaining four undergraduate courses.

The remainder of this section presents the questions themselves and our findings on the comparability of the students' performance of the students in courses 1, 2, 4, and 6.

4.1 Expressions

This question tests students' knowledge of a boolean expression with relational and boolean operators.

Expressions

A dependent child can be very loosely defined as a person under 18 years of age who does not earn $10,000 or more a year. An expression that would define a dependent child is

(a) `age < 18 && salary < 10000`
(b) `age < 18 || salary < 10000`
(c) `age <= 18 && salary <= 10000`
(d) `age <= 18 || salary <= 10000`

Overall, 85% of the students selected the correct response (a). Almost half of the remainder (7%) selected (c), suggesting an error with evaluating the relational operators. Fewer students (3%) selected (b), indicating less difficulty with the logical operator.

By course, the percentage of correct responses varied from 78-89%. The proportions of correct responses were compared using a chi-square test and no difference was found at $p < 0.05$. This marks this question as one that can be used for benchmarking across multiple institutions.

Id	Open/closed book	Questions used	% of exam	Exam % of course
1*	closed	11	47	35
2*	closed	11	28	40
3	closed	11	16	50
4*	open	10	36	50
5	open	11	110	30
6*	closed with 'cheat sheet'	11	35	50

Table 3: Exam profiles, with asterisks indicating the four courses that were used in the final benchmarking

Course	Number of papers
1*	36
2*	238
3	76
4*	297
5	13
6*	166

Table 4: Exam papers from each institution

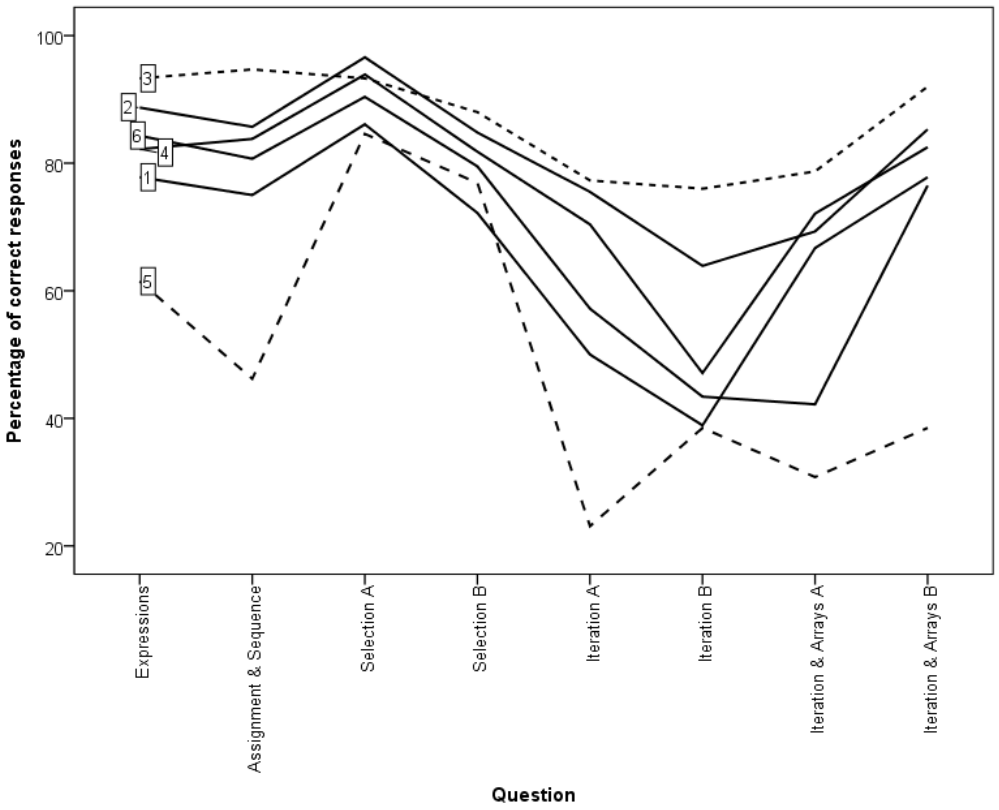


Figure 1: Percentage of correct responses for closed response questions

4.2 Assignment and sequence

This question tests students' knowledge of assignment and sequence and their ability to trace simple code.

Assignment & Sequence

What are the values of *girls*, *boys*, and *children* after the following code has been executed?

```
int girls = 0;
int boys = 0;
int children = 0;
children = girls + boys;
girls = 15;
boys = 12;
```

(a) 0, 0, 0
(b) 0, 0, 27
(c) 15, 12, 0
(d) 15, 12, 27

Most students (83%) selected the correct response (c). Most of the remainder (11%) selected (d), suggesting either that they evaluated the assignment operator but not the sequence of operations, or that they consider the assignment in the fourth statement to be some kind of statement of continuous state rather than a single operation in a sequence (Simon, 2011).

By course, the percentage of correct responses varied from 75-86%. The proportions of correct responses were compared using a chi-square test and no difference was found at $p < 0.05$.

4.3 Selection

This question requires students to trace a nested *if* statement.

Selection A

This question refers to the following code, where the variables *p*, *q*, and *result* all have integer values:

```
int p = 1;
int q = 2;
int result = 4;
if (p < q) {
    if (q > 4) {
        result = 5;
    } else {
        result = 6;
    }
}
```

What would be the value in the variable result after the code is executed?

Overall, 94% of the U/G students calculated the correct value of 6. The next most frequent answer was 5 (3%), indicating a problem evaluating the second *if* statement and then 4 (2%), indicating a problem evaluating the first *if* statement.

By course, the percentage of correct answers varied from 86-94%. A chi-square test showed this difference was significant, which means that the question cannot be reasonably used for multi-institutional benchmarking.

$$\chi^2 (3,737) = 10.035, p < 0.05$$

It has subsequently been suggested that not only are *p* and *q* uninformative names (which is intended), they are poor variable names for code in an exam question because they will be easily confused by students with dyslexia.

The next question also requires students to trace a nested if statement.

Selection B

Consider the following block of code, where the variables *a*, *b*, and *c* each store integer values:

```
if (a > b) {
    if (b > c) {
        answer = c;
    } else {
        answer = b;
    }
} else if (a < c) {
    answer = c;
} else {
    answer = a;
}
Console.WriteLine(answer);
```

In relation to the above block of code, which of the following values for the variables will cause the value in variable *b* to be printed?

(a) a=1; b=2; c=3;
(b) a=1; b=3; c=2;
(c) a=2; b=1; c=3;
(d) a=3; b=2; c=1;

Overall, 82% of the students selected the correct response (c). The next most frequent response was (d) (9%), indicating a problem evaluating the second *if* statement.

By course, the percentage of correct answers varied from 72-82%. The proportions of correct responses were compared using a chi-square test and no difference was found at $p < 0.05$.

The final selection question is a code-explaining question asking students to determine the purpose or outcome of the same piece of code that was used in the preceding question.

Selection C

In one sentence, describe the purpose of the above code (ie the if/else if/else block). Do NOT give a line-by-line description of what the code does. Instead, tell us the purpose of the code.

Overall, 36% of the students gave a fully correct answer and 43% a partially correct answer. The remaining 21% gave answers that were awarded no marks.

The mean mark varied from 52-70%. A Kruskal-Wallis test showed these differences were significant.

$$\chi^2 (3,704) = 41.213, p < 0.05$$

4.4 Iteration

The first iteration question checks whether students understand a simple *while* loop. The loop iterates too many times to be easily traced, although a student who is unsure would be able to substitute a smaller number, trace the code, and then deduce the answer for the larger number.

Iteration A

How many times will the body of the following loop be executed?

```
count = 0;
while (count < 357)
{
    balance = balance + deposit;
    count = count + 1;
}
```

(a) 1
(b) 356
(c) 357
(d) 358

Overall, 68% of the U/G students selected the correct response (c). The next most frequent response was (b) (21%), and then (d), (8%) indicating one-off errors in calculating the number of iterations. The low number of responses for (a) (4%) indicated that most students understood that the number of iterations was determined by the loop condition.

The percentage of correct answers for the U/G group varied from 50-76%. A chi-square test showed this difference was significant.

$$\chi^2 (3,736) = 21.172, p < 0.05$$

The next iteration question, a multiple-choice code-explaining question (Simon & Snowdon, 2011), asks students to determine the purpose of a simple *for* loop.

Iteration B

What is the purpose or outcome of the following piece of code?

```
int result = 0;
for (int i=1; i<=value; i++)
{
    result = result + i;
}
```

(a) to add a counter to a result
(b) to count the numbers from 1 to *value*
(c) to add all the numbers from 1 to *value* – 1
(d) to add all the numbers from 1 to *value*

Overall, 51% of the students selected the correct response (d). The next most frequent response was (b) (21%). While this is clearly a nonsensical explanation (the answer would be the same as *value*), this is potentially what we earlier called a tricky question, as some students might misread *i* as 1 in the assignment statement within the loop body. A different name should be used for the loop counter in future versions of this question. The combined response for (c) and (d) indicated that 63% of the students considered that the purpose was adding all the numbers.

By course, the percentage of correct answers varied from 39-64%. A chi-square test showed that this difference was significant.

$$\chi^2 (3,737) = 23.495, p < 0.05$$

4.5 Iteration and arrays

This code-tracing question requires students to trace a *while* loop operating on two arrays of integers.

Iteration & Arrays A

What will be the value of *result* after the following code statements are executed?

```
int[] val1 = {1,-5,2,0,4,2,-3};
int[] val1 = {1,-5,2,4,4,2,7};
int result = 0;
int i = 0;
while (i < val1.Length)
{
    if (val1[i] != val2[i])
    {
        result = result + 1;
    }
    i = i + 1;
}
```

Overall, 64% of the students calculated the correct answer (2). The next most frequent response was 1 (13%), and then 5 (9%).

The percentage of correct answers varied from 42-72%. A chi-square test showed this difference was significant.

$$\chi^2 (3,737) = 45.834, p < 0.05$$

Another multiple-choice code-explaining question (Simon & Snowdon, 2011) asks students to determine the purpose of a *for* loop containing a selection statement that processes the elements of an array.

Iteration & Arrays B

What is the purpose or outcome of the following piece of code?

```
int result = 0;
for (int i=0; i<=nums.Length; i++)
{
    if (nums[i] < 0)
    {
        result = result + 1;
    }
}
```

(a) to find the smallest element in the array of numbers
(b) to count the negative numbers in the array of numbers
(c) to count the numbers in the array of numbers
(d) to sort the array of numbers

Overall, 82% of the students selected the correct response (b). The next most frequent response was (c) (10%). The combined responses to (b) and (c) suggest that most students understood that the loop involved counting.

The percentage of correct answers varied from 77-85%. The proportions of correct responses were compared using a chi-square test and no difference was found at $p < 0.05$.

The first of the two code-writing questions in the set asks students to write some code (about three lines) comparing the elements of an array with their indexes.

Iteration & Arrays C

Suppose you had an array of integers called *myArray*. Write code that would print out every element of that array that had the same value as its index position. For example, given the array {0, 2, 1, 3}, the code would print the values 0 and 3.

Overall, 56% of the students gave a fully correct answer and 24% a partially correct answer. The remaining 20% were awarded no marks.

The mean mark varied from 62-75%. A Kruskal-Wallis test showed that these differences were significant.

$$\chi^2 (3,710) = 22.776, p < 0.05$$

The second code-writing question (shown in Figure 2) gives students a block of code that performs a certain operation on an array, and asks them to write code that reverses that operation. It is worth noting that a number of the participants at the workshop felt that this question was too hard; that one participant of the study declined to use the question because of its difficulty; and that one other participant included it as a bonus question, making it possible for students to score more than 100% on the exam. Overall, 32% of the students gave a fully correct answer and 48% a partially correct answer. The remaining 20% were awarded no marks.

The mean mark for the U/G group varied from 47-66%. A Kruskal-Wallis test showed that these differences were significant.

$$\chi^2 (2,679) = 39.213, p < 0.05$$

4.6 Summary of results

Four of the eleven questions showed no significant difference between performance results over the four institutions for our benchmarking exercise. These questions were all multiple-choice; three were code-tracing questions and one was a code-explaining question. The overall percentage of correct responses and the lower and upper range values across institutions are shown in Table 5.

The questions together cover six introductory programming topics. We propose that these questions could be used to benchmark undergraduate student performance in introductory programming courses that teach using Java, C# or Python; that introduce objects from an early to late stage; and that assess with open-book or closed-book exams.

Question	Number of papers	% correct responses	Lower and upper range across institutions
Expressions	737	85	78-89
Assignment & Sequence	737	83	75-86
Selection B	736	82	72-82
Iteration & Arrays B	737	82	77-85

Table 5: Suitable benchmarking questions

5 Discussion

The workshop at ACE provided an opportunity for 17 programming educators from 13 institutions in Australia and New Zealand to discuss and decide upon questions for the benchmarking exercise. The robust debate and ultimate consensus on the final selection of questions gave a firm foundation for this study. The involvement of educators from six different institutions in two countries should give wide acceptability to the findings of the study, with useful outcomes that may be more widely used by programming educators and applied in future research studies. Further benefits to this work include the illuminating discussion about exam questions, which was useful for reflection on assessment practices. An exercise of this type, with a practical outcome, also helps to strengthen the computing education community.

While we began this study with student performance data from six institutions, it quickly became clear that students at two of the institutions had performed quite differently from those at the other four. As we were seeking questions on which performance was comparable at a range of institutions, we excluded the two non-comparable institutions from our analysis. The four remaining institutions still represent a broad range: two of them are metropolitan universities in Australia, one is a large regional university in Australia, and one is a polytechnic in New Zealand.

Having found four questions on which there was no significant difference in performance across the four institutions, we proposed these questions as a benchmark. We were then effectively able to apply that benchmark to all six institutions, concluding with some circularity that the four institutions meet the benchmark, while one of the others fails to meet it and one exceeds it.

6 Conclusions and future work

The results of our study can serve as a benchmarking tool to compare learning outcomes of introductory programming students across courses, institutions and countries. Based on our findings we wish to extend this set to cover other core introductory programming topics.

It must be made very clear that we are not suggesting that the four questions identified here are all that needs to be assessed in an introductory programming course. What we are suggesting is that inclusion of these questions in the final exam of an introductory programming course will permit the assessor to form an idea of how the students in that course compare with the students at the three Australian and one New Zealand institutions analysed in this benchmarking exercise.

We are also not suggesting that these questions should set the standard of difficulty in the final exams of programming courses. All four of the exams in the benchmarking exercise included questions that were substantially longer and more difficult than those used here. Furthermore, our study did not consider possible differences in ability levels of students. All we are saying is that when eleven common questions were included in the final exams in those four courses, on four of those questions there was no significant difference in the performance of the students; and therefore that it should be possible to use those questions as a benchmark in other courses.

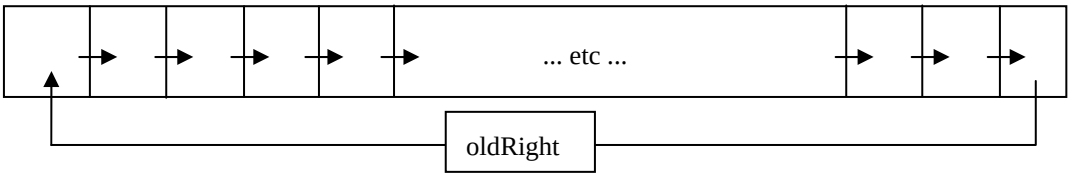
Future work will include an expansion of the question set to cover additional topics, and, if further participants are forthcoming, confirmation of these findings at a larger number of institutions.

7 Acknowledgements

The authors thank the Australian Government's Office for Learning and Teaching (OLT) for their support of the ACE 2013 workshop and of the BABELnot project.

Iteration & Arrays D

We can represent an array of integers as a sequence of elements arranged from left to right, with the first element at the left and the last element at the right. Using this representation, a programmer wishes to move all elements of an array one place to the right, with the rightmost element being 'wrapped around' to the leftmost position, as shown in this diagram.



Here is the code that performs that shift for an array called *values*:

```
int oldRight = values[values.Length - 1];
for (int i = values.Length - 1; i > 0; i--)
    values[i] = values[i - 1];
values[0] = oldRight;
```

For example, if *values* initially contains the integers [1, 2, 3, 4, 5], once the code has executed it would contain [5, 1, 2, 3, 4].

Write code that will undo the effect of the above code. That is, write code that will move all the elements of the array one place to the left, with the leftmost element being wrapped around to the rightmost position.

Figure 2: the final iteration and arrays question

8 References

Cameron, I. and Hadgraft, R. (2010): Engineering and ICT Learning and Teaching Academic Standards Statement. Strawberry Hills, NSW, Australia: Australian Learning and Teaching Council.

Clear, T., Philpott, A., Robbins, P. and Simon. (2009/2010): Report on the Eighth BRACElet Workshop: BRACElet Technical Report 01/08 AUT University, Auckland. *Bulletin of Applied Computing and Information Technology* **7**(1).

Harland, J., D'Souza, D. and Hamilton, M. (2013): *A comparative analysis of results on programming exams*. In proceedings of the 15th Australasian Computing Education Conference (ACE 2013), Adelaide, Australia.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, E., Sanders, K., Seppälä, O., Simon, B. and Thomas, L. (2004): A multi-national study of reading and tracing skills in novice programmers. *SIGCSE Bulletin*, **36**(4), 119-150.

Lister, R., Corney, M., Curran, J., D'Souza, D., Fidge, C., Gluga, R., Hamilton, M., Harland, J., Hogan, J., Kay, J., Murphy, T., Roggenkamp, M., Sheard, J., Simon and Teague, D. (2012): *Toward a shared understanding of competency in programming: An invitation to the BABELnot project*. In proceedings of the 14th Australasian Computing Education Conference (ACE 2012), Melbourne, Australia.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Ben-David Kolikant, Y., Laxer, C., Thomas, L., Utting, I. and Wilusz, T. (2001): A multi-national, multi-institutional study assessment of programming skills of first-year CS students. *SIGCSE Bulletin - Working Group reports: Making inroads to improve computing education*, **33**(4), 125-140.

Oliver, R. and Towers, S. (2000): *Benchmarking ICT literacy in tertiary learning settings*. In proceedings of the 17th Annual Australian Society for Computers in Learning in Tertiary Education (ASCILITE 2000), Coffs Harbour, Australia.

Pea, R. D. (1986): Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research*, **2**(1), 25-36.

Petersen, A., Craig, M. and Zingaro, D. (2011): *Reviewing CS1 exam question content.* In proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE'11), Dallas, Texas, USA.

Sheard, J., Simon, Carbone, A., Chinn, D., Clear, T., Corney, M., D'Souza, D., Fenwick, J., Harland, J., Laakso, M.-J. and Teague, D. (2013): *How difficult are exams? A framework for assessing the complexity of introductory programming exams.* In proceedings of the 15th Australasian Computing Education Conference (ACE 2013), Adelaide, Australia.

Sheard, J., Simon, Hamilton, M. and Lönnberg, J. (2009): *Analysis of research into the teaching and learning of programming*. In proceedings of the Fifth International Computing Education Research workshop (ICER 2009), Berkeley, San Francisco, USA.

Sim, S. E., Easterbrook, S. and Holt, R. (2003): *Using benchmarking to advance research: A challenge to software engineering*. In proceedings of the 25th International Conference on Software Engineering.

Simon. (2011): *Assignment and sequence: why some students can't read a simple swap*. In proceedings of the Eleventh International Conference on Computing Education Research – Koli Calling 2011, Joensuu, Finland.

Simon. (2013): *Soloway's rainfall problem has become harder*. In proceedings of the First International Conference on Learning and Teaching in Computing and Engineering (LaTiCE 2013), Macau.

Simon, Sheard, J., Carbone, A., D'Souza, D., Harland, J. and Laakso, M.-J. (2012): *Can computing academics assess the difficulty of programming examination questions?* In proceedings of the 11th Koli Calling International Conference on Computing Education Research, Finland.

Simon and Snowdon, S. (2011): *Explaining program code: giving students the answer helps – but only just.* In proceedings of the Seventh International Computing Education Research Workshop (ICER 2011).

Simon, B., Clancy, M., McCartney, R., Morrison, B., Richards, B. and Sanders, K. (2010): *Making sense of data structures exams*. In proceedings of the Sixth International Computing Education Research workshop (ICER 2010), Aarhus, Denmark.

Soloway, E. (1986): Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM*, **29**(9), 850-858.

Soloway, E., Ehrlich, K., Bonar, J. and Greenspan, J. (1982): *What do novices know about programming?* : Yale University, Department of Computer Science.

TEQSA (2012): *Tertiary Education Quality and Standards Agency*. Australian Government. Retrieved from http://www.teqsa.gov.au/higher-education-standards-framework

Whalley, J., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, P. K. A. and Prasad, C. (2006): *An Australasian study of reading and comprehension skills in novice programmers, using the Bloom and SOLO taxonomies*. In proceedings of the Eighth Australasian Computing Education conference (ACE 2006), Hobart, Australia.

Woolf, H., Cooper, A., Bourdillon, B., Bridges, P., Collymore, D., Haines, C., Turner, D. and Yorke, M. (1999): Benchmarking academic standards in history: An empirical exercise. *Quality in Higher Education*, **5**(2), 145-154.