A basic recursion concept inventory

Sally Hamouda, Stephen H. Edwards, Hicham G. Elmongui, Jeremy V. Ernst & Clifford A. Shaffer

To cite this article: Sally Hamouda, Stephen H. Edwards, Hicham G. Elmongui, Jeremy V. Ernst & Clifford A. Shaffer (2017) A basic recursion concept inventory, Computer Science Education, 27:2, 121-148, DOI: 10.1080/08993408.2017.1414728

To link to this article: https://doi.org/10.1080/08993408.2017.1414728

Published online: 15 Dec 2017.

COMPUTER SCIENCE EDUCATION, 2017
VOL. 27, NO. 2, 121–148
https://doi.org/10.1080/08993408.2017.1414728

A basic recursion concept inventory

Sally Hamouda[a], Stephen H. Edwards[b], Hicham G. Elmongui[c], Jeremy V. Ernst[d] and Clifford A. Shaffer[b]

[a]Department of Computer Engineering, Cairo University, Cairo, Egypt; [b]Department of Computer Science, Virginia Tech, Blacksburg, VA, USA; [c]Department of Computer and Systems Engineering, Alexandria University, Alexandria, Egypt; [d]School of Education, Virginia Tech, Blacksburg, VA, USA

ABSTRACT
Recursion is both an important and a difficult topic for introductory Computer Science students. Students often develop misconceptions about the topic that need to be diagnosed and corrected. In this paper, we report on our initial attempts to develop a concept inventory that measures student misconceptions on basic recursion topics. We present a collection of misconceptions and difficulties encountered by students when learning introductory recursion as presented in a typical CS2 course. Based on this collection, a draft concept inventory in the form of a series of questions was developed and evaluated, with the question rubric tagged to the list of misconceptions and difficulties.

ARTICLE HISTORY
Received 08 June 2017
Accepted 21 November 2017

KEYWORDS
Recursion; concept inventory; misconceptions

1. Introduction

Recursion is both an important and a difficult topic for introductory Computer Science students. Recursion is one the most important and hardest topics in lower division Computer Science courses (Dale, 2006; Goldman et al., 2010; Hertz & Ford, 2013; Tew & Guzdial, 2011). Efforts can be made to enhance the learning of recursion through interventions such as allowing student to practice exercises that address their misconceptions (Hamouda, Edwards, Elmongui, Ernst, & Shaffer, 2018). But evaluating any such interventions depend on being able to effectively measure a given student's understanding both before and after the intervention. This need for a reliable measurement tool is the main motivation to this work.

Students often develop misconceptions about the topic that need to be diagnosed and corrected. In order to assess student progress, it is helpful to have a test that can recognize whether a given student has the known misconceptions. A Concept Inventory (CI) is a test that can classify an examinee as either someone who thinks in accordance with accepted conceptions on a body of knowledge or in accordance with common misconceptions (Adams & Wieman, 2011; Rowe & Smaill, 2007).

CONTACT Clifford A. Shaffer shaffer@vt.edu

To be considered a successful and valid instrument, a CI must be approved by content experts. A CI is not a comprehensive test of everything a student should know about a topic after instruction (Herman, 2011). Rather, CIs selectively test only critical concepts of a topic (Rowe & Smaill, 2007), since these are required to be considered to have mastered the topic.

CIs have been successfully developed and used in STEM disciplines like Physics (Savinainen & Scott, 2002), Chemistry (Krause, Birk, Bauer, Jenkins, & Pavelich, 2004), and Biology (D'Avanzo, 2008) to drive discipline-specific education research and pedagogical reforms (Almstrum et al., 2006; Taylor et al., 2014). For example, in Physics, the Force Concept Inventory (FCI) showed gaps between how students and instructors think about concepts related to mechanics (Savinainen & Scott, 2002). In Computer Science, the development of concept inventories is growing. The related work section presents efforts in Computer Science concept inventory development. This paper describes our process used to develop a concept inventory that measures students understanding of basic recursion. We began by developing a collection of misconceptions and difficulties encountered by students when learning introductory recursion as presented in a typical CS1 or CS2 course. We then developed a series of questions, with the question rubric tagged to the list of misconceptions and difficulties. Care was taken to ensure that as many of the items on the misconceptions and difficulties list as possible are covered by multiple concept inventory questions.

2. Related work

In Computer Science, the development and use of CIs is rapidly growing. We are aware of efforts to develop CIs for the topics of discrete math (Almstrum et al., 2006), digital logic (Herman, Loui, & Zilles, 2010), operating systems (Andrus & Nieh, 2012; Webb & Taylor, 2014), introductory programming courses (Kaczmarczyk, Petrick, East, & Herman, 2010), algorithms and data structures (Danielsiek, Paul, & Vahrenhold, 2012; Paul & Vahrenhold, 2013), Binary Search Trees (Karpierz & Wolfman, 2014), and Object-Oriented Programming (Ragonis & Ben-Ari, 2005). Our research group has also recently developed a concept inventory for introductory Algorithm Analysis (Farghally, Koh, Ernst, & Shaffer, 2017). In the remainder of this section, we present an overview for many of these efforts.

Kaczmarczyk et al. (2010) worked to find student misconceptions in a CS1-level programming course. Using a Delphi process (Dalkey & Helmer, 1963), the authors gathered 30 concepts from a pool of experts that they think are the most difficult in CS1 programming. From these, the authors selected ten concepts as their initial focus of interest. They are memory model, references and pointers, primitive and reference type variables, control flow, iteration and loops, types, conditionals, assignment statements, arrays, and operator precedence. The authors designed a test of 18 questions covering the concepts of interest. In order to make sure that the results are not problem dependent, each concept was covered in questions with at least two different variations. The authors conducted student interviews to help them understand student misconceptions regarding the targeted concepts. Eleven undergraduate students participated in the interviews. These students were either currently or recently enrolled in the CS1 course. Each interview lasted about an hour and was audio and video recorded. In the interviews, each student was asked to solve questions for all ten concepts. The purpose of the interview was to reveal the misconceptions of the students and validate the Delphi expert's conclusions about the difficult concepts. The authors analyzed the student interviews and described in detail the misconceptions found in memory model representation and default value assignment of primitive values.

Danielsiek et al. (2012) described their first steps toward building a concept inventory for Algorithms and Data Structures. Their results were based on expert interviews and the analysis of 400 exams to identify the core concepts that are considered to be associated with misconceptions. They reported a pilot study to verify misconceptions previously reported in the literature and to identify additional misconceptions. They have then wrote an initial instrument to detect misconceptions related to algorithms and data structures (Paul & Vahrenhold, 2013). They presented the results from a second study that aimed at assessing first-year student misconceptions. Their second study confirmed findings from the previous small-scale studies, but additionally broadened the scope of the topics.

Karpierz and Wolfman (2014) report an initial effort to determine misconceptions and design a CI for Binary Search Trees and Hash Tables. They focused on iterative methods rather than recursion. The authors found student misconceptions by showing exam responses to nine instructors, showing them sample exam responses with the goal to understand how an expert reorganizes something important that the audience does not. The authors also reviewed more than 200 exam problems along with project code to determine the most difficult problems. They interviewed 25 students who each solved two questions while thinking aloud. The authors found three main topics where students hold misconceptions: the possibility of duplicates in BSTs, conflation of Heaps and BSTs, and Hash table resizing. The authors designed three multiple choice questions to address those misconceptions.

Ragonis and Ben-Ari (2005) presented an initial effort to identify misconceptions and difficulties in Object-Oriented Programming (OOP). The authors gathered data during two academic years from students studying OOP in tenth grade CS. The data gathered included home works, lab exercises, tests, and projects. They used these data to identify a comprehensive categorized list of misconceptions and difficulties in OOP understanding. One novel aspect of this work is the reporting of difficulties in addition to misconceptions.

Taylor et al. (2014) presented a recent survey paper on Computer Science CIs. It includes a recommendation to build CIs for topics that should evaluate student's ability to engage in processes such as code analysis, program design, program modification, and testing, as these aspects of learning are difficult to assess. Similarly, Zingaro, Petersen, and Craig (2012) notes that it is hard to evaluate traditional code writing exercises.

3. Building a CI

This section presents the steps that have come to be considered best practice when building a CI, and then for measuring a CI's reliability and validity (Goldman et al., 2008; Herman, 2011; Herman et al., 2010; Krone, Hollingsworth, Sitaraman, & Hallstrom, 2010; Nelson, Geist, Miller, Streveler & Olds, 2007).

(1) *Choose concepts* (set the scope): First a set of concepts should be chosen by the CI developers to define the CI's scope. To assure that the CI is a valid assessment tool, many domain experts must acknowledge that the tool assesses the right content, and that it does in fact assess what it claims to assess. By involving expert opinion from the beginning of the CI development process, we can trust that the designed CI assesses core concepts, and that it has appropriate content validity (Allen & Yen, 2001).

(2) *Identify misconceptions*: Instructors and students can be interviewed to identify the specific sub-topics that students struggle to understand. Instructors can identify students' misconceptions from their teaching and exam-marking experience. Students can also be helpful in identifying their confusion about a certain topic (Allen & Yen, 2001).

(3) *Write CI items and draft the CI* (write the questions): The CI developers should use the misconceptions identified from the previous step to formulate the CI questions. The questions could be multiple choice (MCQ), or any other type of question where incorrect answers can be used to identify the associated misconception. For the sake of reliability, the CI would ideally test every concept multiple times (Buck, Wage, Hjalmarson, & Nelson, 2007).
After writing questions for the initial CI, refinement and validation are done through two feedback cycles: the student feedback cycle and the expert feedback cycle.

(4) *Student feedback cycle*: CI developers should give the CI to students and analyze the quality of the CI through interviews and statistical analysis. The interviews should ask students about the clarity of the questions and the answer choices (for MCQs) and find out if the students are truly solving the questions wrongly when they have the targeted misconception. In this step, the reliability of the CI is to be measured to assess the prevalence of various misconceptions and explore the data for differences in performance between sample populations. The CI should be revised and improved based on these analyses before repeating this cycle.

(5) *Expert feedback cycle*: The CI content and individual items are evaluated by experts. The opinions from a diverse group of experts can reach consensus

using a Delphi process (Dalkey & Helmer, 1963), an approach that has been used to develop previous CIs (Goldman et al., 2008; Gray, Evans, Cornwell, Costanzo, & Self, 2003; Streveler, Olds, Miller, & Nelson, 2003).

(6) *Iterate*: The above sequence of steps could be repeated many times, until a reliable and valid CI is achieved. After each iteration, the CI is revised and modified to do a better job of evaluating student misconceptions, and the reliability and validity are measured.

3.1. Measuring a CI's reliability and validity

Reliability of a CI is usually estimated by three methods: test–retest reliability, split-half reliability, and the Cronbach alpha.

In the test–retest method, the reliability of the CI is measured by giving students the CI multiple times in close succession (Allen & Yen, 2001). Test–retest is not usually done because it is a time consuming process and the students can learn little by taking the instrument multiple times, so its results may not be accurate.

Split-half reliability splits the test into two halves and treats each sub-test as a separate instance of the instrument. An estimate of the total reliability is made by building a correlation between the observed scores on the two sub-tests.

The most commonly used method is Cronbach alpha, which finds the average split-half reliability of every possible set of sub-tests. The Cronbach alpha value ranges from -1 to 1 like a correlation coefficient. A cut-off value is selected for alpha above which the CI is considered to be reliable. For example, Herman (2011) suggests a cut-off of .70 for the alpha value, because a high level of reliability was required. CIs need a high level of reliability to be used as a research instrument. However, some inconsistency can be acceptable, since students are inconsistent when they apply their conceptual knowledge.

3.2. Validity

The validity of an instrument can be estimated by correlating the observed scores of a newly created instrument with the observed scores of an accepted instrument (Allen & Yen, 2001). If there is no currently accepted instrument to measure the true score of a topic, statistical methods cannot be used to estimate the validity. Statistical estimates for the reliability for the instrument can potentially invalidate an instrument. As the reliability of an instrument decreases, the validity of the instrument also decreases. If the CI has a Cronbach alpha value below the selected cut-off, then it should not be considered as valid.

Validity can also be established in some cases through face validity and content validity (Allen & Yen, 2001). Face validity exists if the typical person who is familiar with the material believes at first glance that the instrument measures the true score. Face validity must be done along with content validity to ensure the instrument's validity. Content validity is done by systematically polling the opinions of experts to see if they believe that the instrument measures the true score (Allen & Yen, 2001). To test the validity of an instrument, its developers must clearly define what the instrument measures.

The next sections document how we followed the typical steps of building a concept inventory to build a draft basic recursion concept inventory.

4. Building the recursion concept inventory

4.1. Choose concepts

The first step in building a CI is to identify the concepts (topics) based on experts' rating for its difficulty and importance. Previous research has determined the most common problematic topics that lead to students' misunderstanding of recursion. For example, Sanders and Scholtz (2012) claimed that a key factor in mastering recursion is understanding how the program moves from active control, to the base case, and then to passive control in recursive functions. The complexity of the flow-of-control mechanism makes it a difficult concept for students to comprehend. It was found also that in most cases, students that have some difficulty with active flow are also confused about passive flow and have misconceptions about the base case (Scholtz & Sanders, 2010). In addition, students are confused with the comparison to loop structures (Benander & Benander, 2008) and the lack of everyday analogies.

The following is a list of previously identified common problematic topics found in the literature for teaching recursion, ranked based on the frequency of appearance in the literature:

- Passive/backward control flow after reaching the base case (George, 2000; Sanders & Scholtz, 2012; Scholtz & Sanders, 2010).
- The limiting case (George, 2000; Sanders & Scholtz, 2012).
- Active flow (George, 2000; Sanders & Scholtz, 2012).
- Comparison to loop structures (Benander & Benander, 2008).
- Variable updating either due to difficulty in evaluating a conditional statement or difficulty in understanding an explicit update statement (George, 2000).

We began with the topics list that we gathered from the literature. We this extended this list and broke some of the topics into multiple parts to be more descriptive and understandable. We also changed the wording of some topics to be more clear. We then provided the resulting list of topics along with brief description of each topic to 22 instructors to determine their opinions. The extended topics list presented to the instructors is as follows:

- Backward flow (BF): Passive control flow after reaching the base case.
- Active flow (AF): Active control flow until reaching the base case.
- Recursive calls (RC): How to formulate the recursive call.
- Limiting case (LC): How to formulate the stopping condition and when it will be triggered.

- Infinite recursion (INF): Wrongly write or call the recursive function so that the limiting case is never reached.
- Confusion with loop structure (LP): Implementing recursive functions (especially tail recursion) as a loop.
- Variable updating (VU): Unawareness of how variables are updated on every recursive call.

We asked the instructors to order the list with respect to how confusing they think that the topics are to students. We encouraged the instructors to add, delete, merge, or re-word the topics if needed. Two instructors were interviewed face to face, after which we emailed the list to 20 other instructors, along with instructions on how we wished the list to be evaluated and modified. We received replies from 10 out of the 20 instructors. The instructors who replied were from five different institutions in three different countries. Overall, the twelve instructors provided minor modifications on the names or the order of the concepts, and all agreed on the fundamental presentation.

4.2. *Identify misconceptions*

The next step in building a CI is to identify the misconceptions that students have related to the identified topics. To find out student misconceptions, typically, instructors and student interviews are conducted. We sent invitations by email to ten students taking CS2114: Data Structures and Software Design (a traditional CS2 course) during Spring 2014 at Virginia Tech. We asked them to come for interviews. We received a positive reply and interviewed two students. Participation was voluntary and records were stripped of identification after the interviews were completed. The interview was audio recorded and the students were made aware of that. The students solved 8 recursion tracing exercises and one code writing exercise. Since student participation was low, it did not help us in finding student misconceptions. The common misconception found in the answers of both students interviewed was related to backward flow, where the students did not understand what happens to information after the recursive call.

Our primary sources of information for deducing student misconceptions were test answers and the research literature. We analyzed approximately 8000 responses to recursion questions given to students over three semesters in pre-test, post-test, mid-term, or final exams of CS2114: Data Structures and Software Design to find the most common misconceptions. Table 1 shows the number of students and recursion questions on each test.

We have chosen to present our findings from the interviews and the analysis of student answers as a list of misconceptions and difficulties, inspired by Ragonis and Ben Ari's work on object-oriented programming (Ragonis & Ben-Ari, 2005). A misconception is a mistaken idea or view resulting from a misunderstanding of something. Difficulty here means the empirically observed inability to do something. It is possible that a student exhibits a difficulty due to an underlying

Table 1. Number of students and number of recursion questions per each exams.

Exam	No. of students	No. of questions
Pre-test spring 2014	152	10
Mid-term spring 2014	160	5
Pre-test fall 2014	178	8
Mid-term fall 2014	216	4
Post-test fall 2014	203	8
Pre-test spring 2015	166	5
Mid-term spring 2015	43	5
Final spring 2015	167	4

misconception (possibly one already listed here or one so far unidentified). A difficulty might also result because the student lacks some skill or knowledge.

The following is the list of common misconceptions and difficulties that we found, categorized by the topic related. We give each an identification tag for use in our analysis presented in later sections of this paper.

4.2.1. *Backward flow*

(1) Misconception: No statements after the recursive call will execute. [BFneverExecute]

(2) Misconception: Statements that come after the recursive call will execute before the recursive call is executed. [BFexecuteBefore]

4.2.2. *Infinite recursion*

(3) Misconception: If there is a base case then it will always execute. If the recursive call does not reduce the problem to the base case, then the base case will return, and that will terminate the recursive method. [InfiniteExecution]

4.2.3. *Recursive call*

(4) Difficulty: Cannot formulate a recursive call that eventually reaches the base case. [RCwrite]

(5) Misconception: A value will be returned from a recursive call even if the `return` keyword is omitted. [RCnoReturnRequired]

(6) Misconception: All recursive calls require the `return` keyword even if the recursive function does not return a value. [RCreturnIsRequired]

4.2.4. *Base case*

(7) Misconception: The base case must appear before the recursive call. The base case must be in the `if` condition, while the recursive call must be in the `else` condition or an `if else` condition. So the student has difficulty recognizing whether the recursive call or the base case is executed when tracing code. [BCbeforeRecursiveCase]

(8) Misconception: The base case action must always return a constant, not a variable. [BCactionReturnConstant]

(9) Misconception: The base case condition must always check a variable against a constant, not against another variable. [BCcheckAganistConstant]
(10) Difficulty: Cannot write a correct base case. The student is given a description for what a function should do, and an incomplete implementation for the function with a missing or incorrect base case. The student has difficulty coming up with a correct base case to complete the implementation. [BCwrite]
(11) Difficulty: Cannot properly evaluate the value for the base case. In nearly all such cases, the student believes that the recursive method executes one more or one less time than it actually does. [BCevaluation]

4.2.5. Updating variables

(12) Misconception: Prior to the recursive call, we can (within the recursive function) define a "global" variable that is initialized once and updates when each recursive call is executed. [GlobalVariable]

4.3. Write CI items and draft the CI

The third step in creating a CI is to write initial CI items (questions) based on the misconceptions generated from the previous step. We initially attempted to create a Concept Inventory as a series of multiple choice questions, with each question targeted to identify whether a student has a particular misconception or not.

We quickly realized that a given multiple choice question with multiple distractors naturally relates to several misconceptions or difficulties, where each distractor ideally relates to a specific misconception or difficulty. We also realized that the nature of our topic lends itself to exercises where the student needs to determine the result of executing a piece of code. It did not seem productive to limit the student to a specific list of distractors, as this would both "lead the witness" and also preclude discovering that students had previously unrecognized misconceptions or difficulties. Thus, all the questions are cast as "fill in the blank" (or free answer) questions, with a rubric that identifies the misconception or difficulty that would lead to a specific answer. If in the process of evaluating the answers to the concept inventory, it is found that some answers not in the rubric are frequently given by students; this would suggest the need for further analysis to discover the cause. The initial rubrics for most of the questions are created from the answers that we have seen from approximately 8000 test responses. The first and the second iterations of the draft CI questions and the misconceptions measured by each question can be found in Appendix 1.

4.4. Recursion CI administration

The first administration was done using the first draft concept inventory. The CI was given to 23 students as a part of the mid-term exam in CS2114 during Summer II at Virginia Tech. The mid-term exam had a total of 21 questions, of which 10 questions were on recursion (the first iteration concept inventory questions in Appendix A.1).

After refining the first draft concept inventory, we came up with the second draft CI. The second administration was done using the second draft concept inventory. The CI was given to 111 students as a part of the mid-term exam in another course in another institution, CSE017 during Fall 2015 at Lehigh University. The mid-term exam had a total of 15 questions, of which 6 questions were on recursion. The second iteration concept inventory questions are shown in Appendix 1).

5. Reliability and validity

5.1. *CI reliability*

We measure CI internal consistency as a measure of reliability. The CI was given as a test to CS2114 students during Summer II. To measure reliability, we used Cronbach-α. The CS2114 exam had a Cronbach-α reliability rating of .8. This preliminary finding indicates that the inventory has acceptable (above .5 is considered acceptable) internal consistency reliability after the first administration.

5.2. *CI validity*

5.2.1. *First administration*

On our initial design of the rubric for each question, we tried to base the rubrics on patterns that we have seen in previous student responses to recursion questions. We checked student answers on the test administered in Summer II to see if, for each question, all candidate answers that we had in the rubrics have been given by the students. We found that all candidate answers in the question rubrics were indeed given by the students. We found one answer that was not already covered by the rubric for each of Questions 1, 8 and 9, and so we have updated the rubrics to include those answers. We also found that Question 3 was answered correctly more than any other question. 95% of the students solved the question correctly. We conclude that the misconception covered by Question 3 is not widely held by the students. We looked at each of the ten CI questions for all the 23 students. We believe that the design of the question and rubric items make a reasonable grader agree that, given certain answers, the student holds the matching misconception as listed in our rubrics. For each question, we have determined the corresponding misconceptions from our rubric for that question. We counted the number of student answers that express each misconception. Table 2 shows the percentage of Summer II students who appear to hold each misconception.

We checked to see if better performance on the entire CI correlated with better performance on individual questions, a process referred to as item response analysis (Crocker & Algina, 1986). So in addition to the classical test theory (CTT), we also used Item response theory (IRT) to evaluate the CI. CTT has the following three problems that IRT solved for us: (An & Yung, 2014):

Table 2. The percentage of students holding each misconception based on the first CI administration.

Misconception	Percentage (%)
BFneverExecute	17.4
BFexecuteBefore	13.04
InfiniteExecution	4.35
RCwrite	0
RCnoReturnRequired	8.7
RCreturnIsRequired	8.7
BCbeforeRecursiveCase	8.7
BCactionReturnConstant	4.35
BCcheckAganistConstant	0
BCwrite	0
BCevaluation	21.75
GlobalVariable	30.43

- CTT has a limitation that the item and student characteristics, such as item difficulty parameters and student scores, are not discernible. Depending on the sub-population in question, item characteristics might change. If a high-ability sub-population is considered, all test items would appear to be easy. But when a low-ability sub-population is considered, the same set of items would be difficult. This limitation makes it difficult to assess individuals' abilities using different test forms. However, using IRT, the item characteristics and the personal abilities are formulated by distinctive parameters. After the items are calibrated for a population, the scores for subjects from that population can be compared directly even if they answer different sub-sets of the items.
- The definition of reliability in CTT is based on parallel tests, which are difficult to achieve in practice. The precision of measurement is the same for all scores for a particular sample. In CTT, longer tests are usually more reliable than shorter tests. However, reliability in IRT is defined as a function that is conditional on the scores of the measured latent construct. Precision of measurement differs across the latent construct continuum and can be generalized to the whole target population. With IRT, measurement precision is often depicted by the information curves. These curves can be treated as a function of the latent factor conditional on the item parameters. They can be calculated for an individual item or for the whole test.
- Missing values in CTT are difficult to handle during both test development and subject scoring. Subjects who have one or more missing responses cannot be scored unless these missing values are imputed. In contrast, the estimation framework of the IRT models makes it straightforward to analyze items that have random missing data. IRT can still calibrate items and score subjects using all the available information based on the likelihood; the likelihood-based methods are implemented in the IRT procedure.

Table 3. Item analysis for the first draft CI.

Question	Difficulty index	Discrimination index
Item 1	56.52	31.06
Item 2	59.48	23.51
Item 3	95	15.79
Item 4	91.3	−4.8
Item 5	91.3	47.19
Item 6	62.32	42.05
Item 7	91.23	58.28
Item 8	92.75	54.39
Item 9	60.87	22.67
Item 10	90.23	63.6

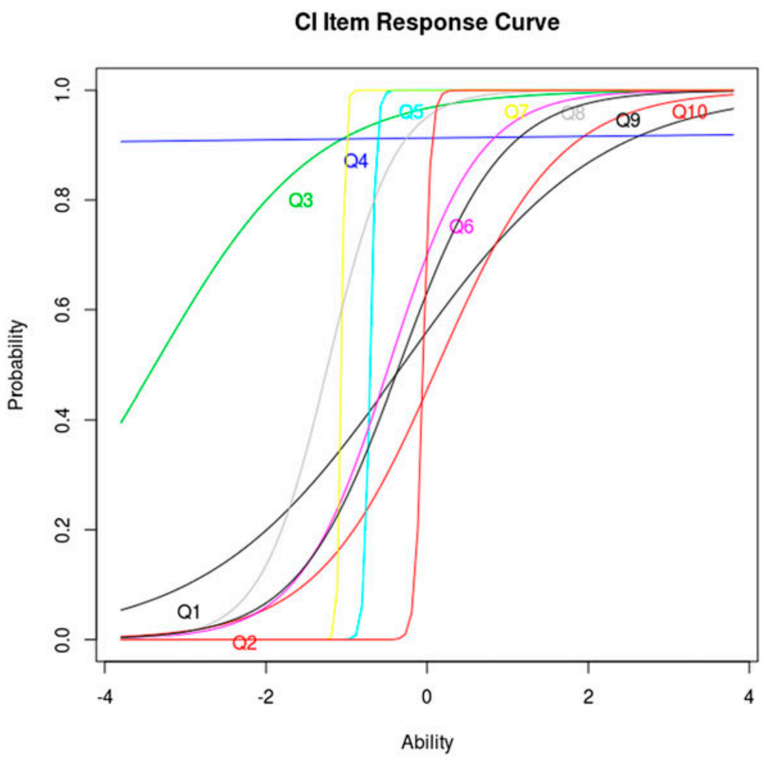

Figure 1. Item response curve for all the items in the first iteration of the CI. The ability on the *x*-axis of the IRCs refers to student performance.

For these reasons, we used IRT to evaluate item quality by performing item analysis as shown in Table 3, and by constructing item response curves (IRCs) as shown in Figure 1.

For most of the questions, the IRC demonstrates the desired correlation between conceptual knowledge and item performance. Student ability is measured by the sum of the scores of the ten concept inventory questions. For most of the questions, as student ability increases, the probability to solve the question correctly increases as well. We found that Question 4 did not show the desired

Table 4. Misconceptions and questions matrix for the second iteration of the CI.

Misconception	Q1	Q2	Q3	Q4	Q5	Q6
BFneverExecute	X					
BFexecuteBefore	X					
RCwrite		X	X			X
RCnoReturnRequired	X					X
RCreturnIsRequired		X	X			
BCbeforeRecursiveCase	X			X		
BCactionReturnConstant					X	
BCcheckAganistConstant					X	
BCwrite		X	X			X
BCevaluation		X	X	X		X
GlobalVariable						X

behavior since student performance on the question did not vary according to student performance on the exam. This is why this question also has a low discrimination index.

Based on the findings from the first iteration, in the second CI version, we dropped Question 3. Based on our investigation of previous recursion pre-tests, we determined that students can easily spot infinite recursion using their prior knowledge. Depending on expert feedback, we may also drop the items whose discrimination index is less than 30% (Questions 4 and 9). Since Question 10 is a writing question, we changed it to make it a little harder (e.g. ask the students to implement a recursive function to find the largest element in an array, or to search for a given number in an array). The second iteration CI questions can be found in Appendix 1, along with the detailed rubrics and associated misconception or difficulty indicated by each possible answer. Table 4 shows a summary of misconceptions and difficulties associated with each question.

5.2.2. Second administration

In Fall 2015, we administered the second iteration of the CI. We found that all the candidate answers in the questions rubric were given by the students. We did not find any answers for any of the questions that were not covered by the rubrics. We looked at each of the six CI questions for all of the 111 students. We then counted the number of student answers that expresses each misconception. Table 5 shows the percentage of Fall 2015 students who appear to hold each misconception.

We also evaluated the CI by performing item analysis in Table 6 and constructing item response curves (IRCs) in Figure 2. We can see from the IRC that most of the questions except Question 5 demonstrated the desired correlation between the conceptual knowledge and item performance. For all the questions, as the student ability increases, the probability to solve the question correctly increases as well. We found Question 5 did not show the a good behavior as other questions, it did not show as sharp difference in the probability depending on the student ability. We will think about how Question 5 can be modified and will ask the experts about their opinion on how to modify this question. It could be that the misconceptions tested by this question is not widely held by students.

Table 5. The percentage of students holding each misconception based on the second CI administration.

Misconception	Percentage (%)
BFneverExecute	27.93
BFexecuteBefore	9.5
RCwrite	30
RCnoReturnRequired	20
RCreturnIsRequired	23
BCbeforeRecursiveCase	15
BCactionReturnConstant	6
BCcheckAganistConstant	23
BCwrite	42
BCevaluation	18
GlobalVariable	15

Table 6. Item analysis for the second draft CI.

Question	Difficulty index (%)	Discrimination index (%)
Item 1	34.45	49.16
Item 2	69.13	49.80
Item 3	63.35	78.23
Item 4	80.21	43.45
Item 5	58.64	25.18
Item 6	56.24	60.23

5.2.3. *Can one misconception hide another?*

As part of validating our CI, we are interested in answering the question: Can one misconception hide another? In other words, if a student solved a question with a certain answer such that we believe that this student holds the misconception corresponding to this answer on the rubric, can he also have other misconception that we cannot detect because of the first one? Since not all of the misconceptions are covered by more than one item, if a student did not show a certain misconception on a certain question, then we need to be sure that this is not because another misconception is hiding the first. We have designed our rubrics so that each possible answer covered by the rubric is mapped to a misconception(s). For the most part, the possible misconceptions that can relate to a given problem are also associated with separate answers in the rubric, which minimizes the chances that one misconception is hiding another covered by the same question. The rubrics are presented in Appendix 1. However, to answer this question more accurately, the concept inventory could be expanded to have more questions so that a misconception is covered by more than one question. This will require a test that needs more time. That in turn makes administering the concept inventory harder because instructors will only devote limited time to exams.

5.2.4. *Expert content validity*

In Fall 2015, in order to check content validity of the second draft of the CI, we collected feedback from experts. We choose experts who had at least one-year experience in teaching recursion. On individual items, the experts were asked to:

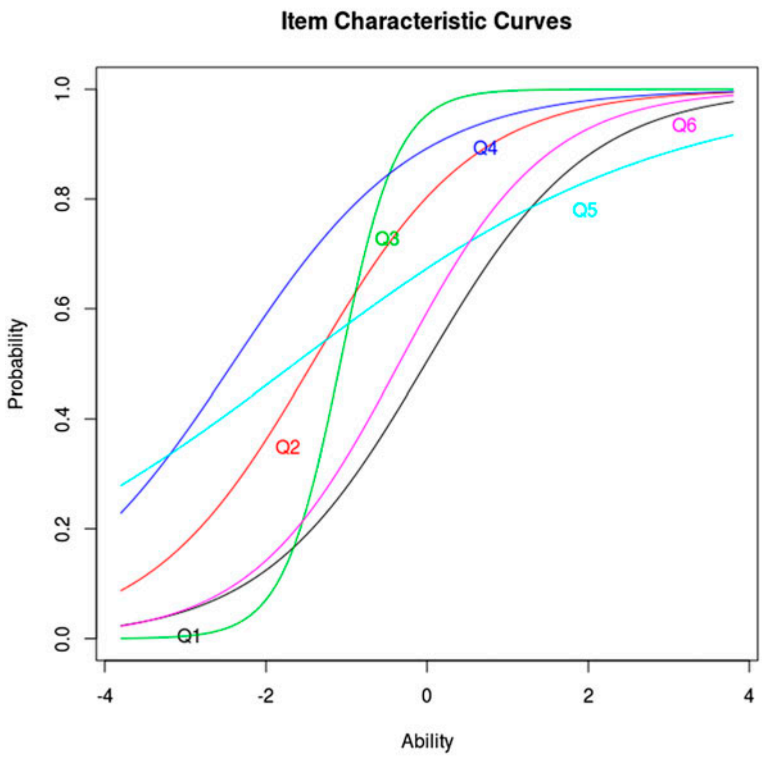

Figure 2. Item response curve for all the items in the second iteration of the CI.

(1) Answer the CI.
(2) Decide whether the item reflects basic recursion concepts that students should know after completing a CS2 level course.
(3) Rate the quality of the question.

Finally, the experts were asked to provide their opinions about the CI as a whole. The experts were asked to:

(1) Decide if the CI as a whole reflects basic recursion knowledge after a CS2 level course
(2) Comment on the topic coverage, and
(3) Indicate how confident they would be that a student who performed well on the CI will perform well in basic recursion in a CS2 level course.

We received 6 responses from the experts contacted. All the experts agreed that the concept inventory does a good job to identify the student misconceptions on basic recursion. Three experts suggested that Question 6, the writing question, should be clarified more. They suggested to provide the function signature at least. One expert noted that the programming language that is used to teach recursion may have an effect on the rubrics that we designed. He noted that for Scala, some of the candidate answers in the rubrics may not fit the targeted misconception. Another expert suggested to have more questions covering the same misconceptions. As we have discussed earlier, adding more questions to the CI increases resistance from instructors to use it. One expert asked why we do not include misconceptions or questions related to multiple recursive functions. Our response is that we consider multiple recursive calls a topic for an advanced recursion concept inventory, not a basic recursion concept inventory. Two experts suggested that we look at student answers to the CI questions to see if it is actually measuring what it is supposed to measure (validating the approach that we have in fact taken). In conclusion, the second draft CI was accepted by the experts.

5.2.5. Validity according to APA standards

We used the validity guidelines in the APA standards for educational and psychological testing (American Educational Research Association, American Psychological Association & National Council on Measurement in Education and Joint Committee on Standards for Educational and Psychological Testing (U.S.), 2014) to evaluate how well our Recursion CI adheres to those standards. The following guidelines are applicable, and we provide a justification of adherence for each one.

(1) A rationale should be presented for each recommended interpretation and use of test scores, together with a comprehensive summary of the evidence and theory bearing on the intended use or interpretation.
    The recursion CI was developed using up-to-date analysis of previous test questions on recursion, comprehensive analysis of the literature on recursion misconceptions, expert interviews and student interviews, and an analysis of 8000 student responses for recursion questions.
(2) The test developer should set forth clearly how test scores are intended to be interpreted and used. The population(s) for which a test is appropriate should be clearly delimited, and the construct that the test is intended to assess should be clearly described.
    We designed a rubric for all possible answers to each question. We used binary (correct/incorrect) grading to grade the CI.
(3) If validity for some common or likely interpretation has not been investigated, or if the interpretation is inconsistent with available evidence, that fact should be made clear, and potential users should be cautioned about making unsupported interpretations.
    It is made clear that the test can be generalized only to be used for C-like languages (such as Java) at the CS2 course level.
(4) If a test is used in a way that has not been validated, it is incumbent on the user to justify the new use, collecting new evidence if necessary.
    We are still working on extending the test to other programming languages.

(5) The composition of any sample of examinees from which validity evidence is obtained should be described in as much detail as is practical, including major relevant sociodemographic and developmental characteristics.
This information is described in detail in this paper.

(6) When a validation rests in part on the opinion or decisions of expert judges, observers, or raters, procedures for selecting such experts and for eliciting judgments or ratings should be fully described. The qualifications, and experience, of the judges should be presented. The description of procedures should include any training and instructions provided, should indicate whether participants reached their decisions independently, and should report the level of agreement reached. If participants interacted with one another or exchanged information, the procedures through which they may have influenced one another should be set forth.
As detailed in this paper, we use expert judges to establish the validity of the CI. This feedback is taken into account for revisions of the CI.

(7) When validity evidence includes statistical analyses of test results, either alone or together with data on other variables, the conditions under which the data were collected should be described in enough detail that users can judge the relevance of the statistical findings to local conditions. Attention should be drawn to any features of a validation data collection that are likely to differ from typical operational testing conditions and that could plausibly influence test performance.
These details are described in this paper.

(8) When a test use or score interpretation is recommended on the grounds that testing or the testing program per se will result in some indirect benefit in addition to the utility of information from the test scores themselves, the rationale for anticipating the indirect benefit should be made explicit. Logical or theoretical arguments and empirical evidence for the indirect benefit should be provided. Due weight should be given to any contradictory findings suggesting important indirect outcomes other than those predicted.
We do not generally claim that there are indirect benefits of using our CI aside from the purposes stated in this paper: As a test of the students, and as data for evaluating the merits of various way to teach recursion.

6. Evidence-centered assessment design for the recursion CI

Evidence-centered design (ECD) is a framework that provides a more formal understanding for procedures that we have followed (Mislevy & Haertel, 2006). ECD organizes the work of assessment design and implementation in terms of layers. In this section, we discuss how the process of creating the recursion CI can be couched in terms of ECD layers.

(1) Domain Analysis: Gather substantive information about the domain of interest that has direct implications for assessment; how knowledge is constructed, acquired, used, and communicated. We performed the following steps for our domain analysis:

  (a) Review the literature for previous research done to find misconceptions about recursion.
  (b) Interview students to learn more about their misconceptions. The interviews were audio recorded. The student was given an exam on recursion and was asked to think aloud while solving the exam questions. Many misconceptions were made clear in this way.
  (c) Interviews with instructors who have been teaching recursion to understand common misconceptions that students are observed to hold on recursion.
  (d) Instructor surveys for the instructors that couldn't participate in the interviews. The surveys had the same questions that were asked of instructors during the interview, in addition to questions about the amount of time dedicated for recursion in class and outside of class.
  (e) Analysis of 8000 student answers on recursion exam questions.

(2) Domain Modeling: Express assessment in narrative form based on domain analysis. For the recursion CI, we can express it as: We need to evaluate student understanding of recursion and test their knowledge with regards to recursion misconceptions.
(3) Conceptual Assessment Framework: Express assessment argument in structures and specifications for tasks and tests, evaluation procedures, measurement models. The recursion CI was given as a part of a midterm exam. It is composed of few (6 to 8) questions. All of the questions are tracing questions, except for one which asks students to write a complete recursive function to accomplish a certain task. Our target students should have studied basic recursion in a CS2-level course.
(4) Assessment implementation: Implement assessment, including presentation-ready tasks and calibrated measurement models. Recursion concept inventory questions are shown in Appendix 1, along with the rubric for each question. For each question, the rubric shows all plausible answers for that questions, based on extensive experience with student responses. The rubrics were built through the manual analysis of the 8000 answers to recursion questions. The few answers given on the first administration that were not covered in rubric were added prior to the second administration. We expect that instructors will use a binary grading system for each question.
(5) Assessment Delivery: Coordinate interactions of students and tasks: task-and test-level scoring; reporting. Renderings of materials; numerical and graphical summaries for individual and groups; Graphical sum-

maries. We present tables laying out how each question, response, and misconception relate to each other. In Section 5, for each recursion CI administration, we show the difficulty index (which is equivalent to the mean grade of the question). We also show a graphical representation of the item response analysis of each question. For each misconception, We show the percentage of the answers which had a this misconception.

7. Conclusion

This paper presents our initial efforts to define a collection of misconceptions and difficulties encountered by students when learning introductory recursion, as presented in a typical CS2 course. We have presented first and second iterations of a draft concept inventory in the form of a series of questions, with the question rubric tagged to a specific list of misconceptions and difficulties.

This initial effort should be continued by giving the CI to more students in different institutions and asking more experts to evaluate the CI. The reliability and the validity of the CI should be measured each time the CI is administrated, to confirm that the developed recursion CI measures students' misconceptions on basic recursion. This recursion concept inventory is meant to measure student understanding of basic recursion, regardless of the instruction method used. We have in fact continued administering the CI in more courses and at more Universities. The basic recursion CI is now available in two programming languages, C and Java. Efforts will be made in the future to extend it to other programming languages in order for the recursion CI to be more generalized.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work received support from the VT-MENA program of Egypt, and the National Science Foundation [grant number DUE-0836940], [grant number DUE-0937863], [grant number DUE-0840719].

References

Adams, W. K. & Wieman, C. E. (2011). Development and validation of instruments to measure learning of expert-like thinking. *International Journal of Science Education, 33*(9), 1289–1312.

Allen, M. J., & Yen, W. M. (2001). *Introduction to measurement theory*. Long Grove, IL: Waveland Press.

Almstrum, V. L., Henderson, P. B., Harvey, V., Heeren, C., Marion, W., Riedesel, C., ... Tew, A. E. (2006). Concept inventories in computer science for the topic discrete mathematics. *SIGCSE Bulletin, 38*(4), 132–145.

American Educational Research Association, American Psychological Association, & National Council on Measurement in Education and Joint Committee on Standards for Educational and Psychological Testing (U.S.). (2014). *Standards for educational and psychological testing*. American Educational Research Association.

An, X., & Yung, Y. F. (2014). Item response theory: What it is and how you can use the IRT procedure to apply it. *SAS364-2014*. SAS Institute Inc. Retrieved from https://support.sas.com/resources/papers/proceedings14/SAS364-2014.pdf

Andrus, J., & Nieh, J. (2012). Teaching operating systems using Android. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE 2012)* (pp. 613–618). Raleigh, NC, USA.

Benander, A. C., & Benander, B. A. (2008). Student monks – Teaching recursion in an IS or CS programming course using the Towers of Hanoi. *Journal of Information Systems Education, 19*(4), 455–467.

Buck, J. R., Wage, K. E., Hjalmarson, M. A., & Nelson, J. K. (2007). Comparing student understanding of signals and systems using a concept inventory, a traditional exam and interviews. In *37th Annual Frontiers in Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports* (pp. S1G–1), Milwaukee, WI, USA.

Crocker, L., & Algina, J. (1986). *Introduction to classical and modern test theory*. New York, NY: Holt Rinehart & Winston.

Dale, N. B. (2006). Most difficult topics in CS1: Results of an online survey of educators. *SIGCSE Bulletin, 38*(2), 49–53.

Dalkey, N., & Helmer, O. (1963). An experimental application of the Delphi method to the use of experts. *Management Science, 9*(3), 458–467.

Danielsiek, H., Paul, W., & Vahrenhold, J. (2012). Detecting and understanding students' misconceptions related to algorithms and data structures. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE 2012)* (pp. 21-26). Raleigh, NC, USA.

D'Avanzo, C. (2008). Biology concept inventories: Overview, status, and next steps. *BioScience, 58*(11), 1079–1085.

Farghally, M. F., Koh, K. H., Ernst, J. V. E., & Shaffer, C. A. (2017). Towards a concept inventory for algorithm analysis topics. In *Proceedings of the 48th ACM Technical Symposium on Computer Science Education (SIGCSE 2017)* (pp. 207–212). Seattle, WA, USA.

George, C. E. (2000). Erosi-visualising recursion and discovering new errors. In Proceedings of the 31st ACM Technical Symposium on Computer Science Education (SIGCSE 2000), SIGCSE '00 (pp. 305–309). Austin, TX, USA.

Goldman, K., Gross, P., Heeren, C., Herman, G., Kaczmarczyk, L., Loui, M. C., & Zilles, C. (2008). Identifying important and difficult concepts in introductory computing courses using a Delphi process. *SIGCSE Bulletin, 40*(1), 256–260.

Goldman, K., Gross, P., Heeren, C., Herman, G. L., Kaczmarczyk, L., Loui, M.C., & Zilles, C.(2010). Setting the scope of concept inventories for introductory computing subjects. *ACM Transactions on Computing Education*, *10*(2), 5:1–5:29.

Gray, G. L., Evans, D., Cornwell, P., Costanzo, F., & Self, B. (2003). Toward a nationwide dynamics concept inventory assessment test. *American Society for Engineering Education Annual Conference & Exposition*. Westminster, CO: IEEE.

Hamouda, S., Edwards, S., Elmongui, H., Ernst, J., & Shaffer, C. (2018). RecurTutor: An interactive tutorial for learning recursion. *ACM Transactions on Computing Education* (To Appear).

Herman, G. L. (2011). *The development of a digital logic concept inventory* (PhD thesis). Champaign, IL: University of Illinois at Urbana-Champaign.

Herman, G. L., Loui, M. C., & Zilles, C. (2010). Creating the digital logic concept inventory. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE 2012)* (pp. 102–106). Milwaulkee, WI, USA.

Hertz, M., & Ford, S. M. (2013). Investigating factors of student learning in introductory courses. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education, SIGCSE '13* (pp. 195–200). Denver, CO, USA.

Kaczmarczyk, L. C., Petrick, E. R., East, J. P., & Herman, G. L. (2010). Identifying student misconceptions of programming. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 107-111). Milwaulkee, WI, USA.

Karpierz, K., & Wolfman, S. A. (2014). Misconceptions and concept inventory questions for binary search trees and hash tables. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE 2014)* (pp. 109-114). Atlanta, GA, USA.

Krause, S., Birk, J., Bauer, R., Jenkins, B., & Pavelich, M. J. (2004). Development, testing, and application of a chemistry concept inventory. In *34th Annual Frontiers in Education Conference (FIE 2004)* (pp. T1G–1). Savannah, GA, USA.

Krone, J., Hollingsworth, J. E., Sitaraman, M., & Hallstrom, J. O. (2010). *A reasoning concept inventory for computer science* (Report No. RSRG-09-01). Clemson, SC: Clemson University.

Mislevy, R. J., & Haertel, G. D. (2006). Implications of evidence-centered design for educational testing. *Educational Measurement: Issues and Practice, 25*(4), 6–20.

Nelson, M. A., Geist, M. R., Miller, R. L., Streveler, R. A., & Olds, B. M. (2007). How to create a concept inventory: The thermal and transport concept inventory. In *Annual Conference of the American Educational Research Association*. Chicago, IL, USA.

Paul, W., & Vahrenhold, J. (2013). Hunting high and low: Instruments to detect misconceptions related to algorithms and data structures. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education, SIGCSE '13* (pp. 29-34). Denver, CO, USA.

Ragonis, N., & Ben-Ari, M. (2005). A long-term investigation of the comprehension of OOP concepts by novices. *Computer Science Education, 15*(3), 203–221.

Rowe, G., & Smaill, C. (2007). Development of an electromagnetic course-concept inventory – A work in progress. *Proceedings of the 18th Conference of the Australian Association for Engineering*. Melbourne, Australia.

Sanders, I., & Scholtz, T. (2012). First year students' understanding of the flow of control in recursive algorithms. *African Journal of Research in Mathematics, Science and Technology Education, 16*(3), 348–362.

Savinainen, A., & Scott, P. (2002). The force concept inventory: A tool for monitoring student learning. *Physics Education, 37*(1), 45–52.

Scholtz, T. L., & Sanders, I. (2010). Mental models of recursion: Investigating students' understanding of recursion. In *Proceedings of the 15th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2010)* (pp. 103–107). Bilkent, Ankara, Turkey.

Streveler, R. A., Olds, B. M., Miller, R. L., & Nelson, M. A. (2003). Using a Delphi study to identify the most difficult concepts for students to master in thermal and transport science. *Proceedings of the Annual Conference of the American Society for Engineering Education*. Nashville, TN, USA.

Taylor, C., Zingaro, D., Porter, L., Webb, K., Lee, C., & Clancy, M. (2014). Computer science concept inventories: Past and future. *Computer Science Education, 24*(4), 253–276.

Tew, A. E., & Guzdial, M. (2011). The FCS1: A language independent assessment of CS1 knowledge. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE 2011)* (pp. 111–116). Dallas, TX, USA.

Webb, K. C., & Taylor, C. (2014). Developing a pre-and post-course concept inventory to gauge operating systems learning. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE 2014)* (pp. 103–108). Atlanta, GA, USA.

Zingaro, D., Petersen, A., & Craig, M. (2012). Stepping up to integrative questions on CS1 exams. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE 2012)* (pp. 253-258). Raleigh, NC, USA.

Appendix 1. CI questions and rubrics

A.1. First iteration CI questions

A.1.1. Backward flow

(1) Given the following code:

```
int function(int y) {
  if (y == 1)
    return 5;
  else {
    function(y - 1);
    y = y + 1;
    return 83;
  }
}
```

What will be returned when `function(2)` is executed? Write a number, or write "infinite recursion" if you think that this call will lead to infinite recursion.

Table A1. Question item 1 rubric.

Answer	Misconception
83	Correct
5	BFneverExecute
6	RCnoReturnRequired
Infinite recursion	BFexecuteBefore
Other	?

(2) Consider the following function.

```
void PrintArray(int[] A, int n) {
  if (n > 0) {
    PrintArray(A, n - 1);
    System.out.print(A[n]);
  }
}
```

What will be printed when `PrintArray(A, 5)` is executed, with array `A` initialized so that position `A[i]` stores value `i`? Write a sequence of numbers that will be printed, or write "nothing" if you think that it will print nothing. Write "infinite recursion" if you think that the call will lead to infinite recursion.

A.1.2. Infinite recursion

(3) Consider the following function.

```
int mystery(int x) {
  if (x > 0)
    return 8;
  else
    return 2 + mystery(x - 1);
}
```

Table A2. Question item 2 rubric.

Answer	Misconception
12345	Correct
1234	BCevaluation
01234	BCevaluation
012345	BCevaluation
54321	BFexecuteBefore
543210	BFexecuteBefore and BCevaluation
4321	BFexecuteBefore and BCevaluation
Nothing	BFneverExecute
Infinite recursion	BCevaluation or ?
Other	?

Table A3. Question item 3 rubric.

Answer	Misconception
Infinite recursion	Correct
8	InfiniteExecution
2 or 10, 12, 14, etc.	BCevaluation
Other	?

Table A4. Question item 4 rubric.

Answer	Misconception
Line 5: return function(x - 1 , y)	Correct
Line 2: x ! = y	Correct
Line 5: return function(x , y+1)	Correct
Line 5: function(x - 1 , y)	RCnoReturnRequired
Line 2: x > any positive number or x == any number >= 3	Correct but may be BCcheckAganistConstant
Line 2: x < any positive number	BCwrite and BCcheckAganistConstant
Line 5: function(x + any positive number)	RCwrite
Line 5: return y- any positive number	BCevaluation
Line 5: return any constant value	BCcheckAganistConstant
Other	?

What value will be returned when `mystery(0)` is executed? Write a number, or write "infinite recursion" if you think that the call will lead to infinite recursion.

A.1.3. Recursive call

(4) The following code leads to infinite recursion when called as `function(3, 2)`:

```
1  int function(int x, int y) {
2    if (x == y)
3      return y;
4    else
5      return function(x + 1, y);
6  }
```

Pick ONE line that you think is the cause of the infinite recursion and write a replacement, so that this replacement will fix the infinite recursion.

(5) Given the following incomplete code:

Table A5. Question item 5 rubric.

Answer	Misconception
return k + SumTo(k - 1)	Correct
return SumTo(k - 1)	RCwrite
return k + SumTo(k + 1)	RCwrite
k + SumTo(k - 1)	RCnoReturnRequired
k + SumTo(k + 1)	RCnoReturnRequired and RCwrite
Any answer that has no recursive call	BCbeforeRecursiveCase
Other	?

Table A6. Question item 6 rubric.

Answer	Misconception
CountDown(x , y-1)	Correct
CountDown(x , y+1)	RCwrite and BCevaluation
return CountDown(x , y-1)	RCreturnIsRequired
return CountDown(x , y+1)	RCreturnIsRequired and RCwrite and BCevaluation
Any answer that has no recursive call	BCbeforeRecursiveCase
Other	?

```
int SumTo(int k)
{
  if (k > 0)
    // missing line;
  else
    return 0;
}
```

Write something to replace the line `// missing line` so that when given a number k, `SumTo` will return a cumulative sum of the values from 1 to k. For example, 15 will be returned when `SumTo(5)` is called, 21 when `SumTo(6)` is called, and so on.

(6) The following incomplete code is meant to print the numbers going from `y` down to `x`, where `x` < `y`. For example, if `CountDown(3, 7)` is called then the following should be printed: 76543

```
void CountDown(int x, int y) {
  if (x <= y) {
    System.out.print(y);
    // missing recursive call
  }
}
```

Write a recursive call that should replace `// missing recursive call`.

A.1.4. Base case

(7) Given the following two methods:

```
int function1(int x, int y) {
  if (x == 1)
    return y;
  else
    return function1(x-1, y) + y;
}
```

Table A7. Question 7 rubric.

Answer	Misconception
6 and 6	Correct
Two different values	BCbeforeRecursiveCase
The same value, but not 6	BCevaluation or ?
Infinite Recursion for both	BCevaluation or ?
Other	?

Table A8. Question item 8 rubric.

Answer	Misconception
a==b and return a	Correct
a==b and return b	Correct
a==b and return constant	BCcheckAganistConstant
A condition like a==constant or b==constant and return a	BCwrite and BCcheckAganistConstant
Other	?

```
int function2(int x, int y) {
  if (x > 1)
    return function2(x-1, y) + y;
  else
    return y;
}
```

What values are returned by the calls `function1(2,3)` and `function2(2,3)`? Write a number for each return value, or write "infinite recursion" if you think either will eventually lead to infinite recursion.

(8) Given the following incomplete recursive method:

```
int Sum(int a, int b) {
 if ( //Missing Case// )
   //Missing Action//
 else
   return Sum(a, b-1)+ b;
}
```

Write something to replace `//Missing Case//` and `//Missing Action//` so that when this recursive function is passed 2 numbers, it will return the sum of all the integers between them. For example, given 2 and 5, add 2 + 3 + 4 + 5 and return 14. If the two numbers are equal, then return that value.

A.1.5. Variables updating

(9) The following function is intended to find the minimum value in an array.

```
int recursiveMin(int[] array, int index) {
  int min = array[0];
  if (index == 0)
    return min;
  else {
    if (array[index] < min)
```

Table A9. Question item 9 rubric.

Answer	Misconception
10	Correct
2	GlobalVariable
8	?
Infinite Recursion	?
Other	?

Table A10. Question item 1 rubric.

Answer	Misconception
83	Correct
5	BFneverExecute
6	RCnoReturnRequired
Infinite recursion	BFexecuteBefore
583	BCbeforeRecursiveCase
Other	?

```
      min = array[index];
    return recursiveMin(array, index-1);
  }
}
```

What will be returned by `recursiveMin` when the following lines are executed?

```
int [] array = {10, 20, 2, 30, 8};
int var= recursiveMin(array, array.length);
```

Write a number, or write “infinite recursion” if you think that the call will lead to infinite recursion.

A.1.6. Writing question

(10) Write a recursive function to compute x to the power y. Assumes that y is positive or zero and both x any y are integers.

A.2. Second iteration CI questions

(1) Given the following code:

```
int function(int y) {
  if (y == 1)
    return 5;
  else {
    function(y - 1);
    y = y + 1;
    return 83;
  }
}
```

What will be returned when `function(2)` is executed? Write a number, or write “infinite recursion” if you think that this call will lead to infinite recursion.

Table A11. Question item 2 rubric.

Answer	Misconception
return k + SumTo(k - 1)	Correct
return SumTo(k - 1)	RCwrite
return k + SumTo(k + 1)	RCwrite
k + SumTo(k - 1)	RCnoReturnRequired
k + SumTo(k + 1)	RCnoReturnRequired and RCwrite
Any answer that has no recursive call	BCbeforeRecursiveCase
Other	?

Table A12. Question item 3 rubric.

Answer	Misconception
CountDown(x , y-1)	Correct
CountDown(x , y+1)	RCwrite and BCevaluation
return CountDown(x , y-1)	RCreturnIsRequired
return CountDown(x , y+1)	RCreturnIsRequired and RCwrite and BCevaluation
Other	?

(2) Given the following incomplete code:

```
int SumTo(int k)
{
  if (k > 0)
    // missing line;
  else
    return 0;
}
```

Write something to replace the line `// missing line` so that when given a number k, `SumTo` will return a cumulative sum of the values from 1 to k. For example, 15 will be returned when `SumTo(5)` is called, 21 when `SumTo(6)` is called, and so on.

(3) The following incomplete code is meant to print the numbers going from `y` down to `x`, where `x` < `y`. For example, if `CountDown(3, 7)` is called then the following should be printed: 76543

```
void CountDown(int x, int y) {
  if (x <= y) {
    System.out.print(y);
    // missing recursive call
  }
}
```

Write a recursive call that should replace `// missing recursive call`.

(4) Given the following two methods:

```
int function1(int x, int y) {
  if (x == 1)
    return y;
  else
    return function1(x-1, y) + y;
```

Table A13. Question item 4 rubric.

Answer	Misconception
6 and 6	Correct
Two different values	BCbeforeRecursiveCase
The same value, but not 6	BCevaluation or ?
Infinite Recursion for both	BCevaluation or ?
Other	?

Table A14. Question item 5 rubric.

Answer	Misconception
a==b and return a	Correct
a==b and return b	Correct
a==b and return constant	BCcheckAganistConstant
A condition like a==constant or b==constant and return a	BCwrite and BCcheckAganistConstant
a==b and return a+b	?
Other	?

```
}

int function2(int x, int y) {
  if (x > 1)
    return function2(x-1, y) + y;
  else
    return y;
}
```

What values are returned by the calls `function1(2,3)` and `function2(2,3)`? Write a number for each return value, or write “infinite recursion” if you think either will eventually lead to infinite recursion.

(5) Given the following incomplete recursive method:

```
int Sum(int a, int b) {
 if ( //Missing Case// )
   //Missing Action//
 else
   return Sum(a, b-1)+ b;
}
```

Write something to replace `//Missing Case//` and `//Missing Action//` so that when this recursive function is passed 2 numbers, it will return the sum of all the integers between them. For example, given 2 and 5, add 2 + 3 + 4 + 5 and return 14. If the two numbers are equal, then return that value.

(6) Write a recursive function to search for a given value in a given array.