Developing a Concept Inventory and Tutorial to Enhance Teaching Recursion

Sally Hamouda, N. Dwight Barnette, Eric Fouh, Mohamed F. Farghally, Hicham G. Elmongui and Clifford A. Shaffer

Department of Computer Science, Virginia Tech

Introduction

Recursion is one of the **most important and most difficult topics** in lower division computer science courses. We describe our development process to create a concept inventory (CI) that can be used as a pre- and post-test to evaluate the effectiveness of recursion teaching. We have developed a tutorial on recursion as part of the OpenDSA framework. The tutorial provides additional value over and above typical instruction as it currently occurs. As a side benefit, the act of doing practice exercises helps to insure that students spend adequate time learning recursion. The recursion tutorial will be used in CS2 courses, replacing or augmenting current instruction on recursion.

Concept Inventory

- Addresses Recursion misconceptions.
- Can be used as pre- and post-tests.
- Measures the learning gain as pre-post- test difference.

Write: Recursive Call

Consider the following incomplete function:

```
int Question1(int Q1_var1)
{
  if (Q1_var1 <= 0)
    return 0;
  else
    return ("Missing code");
}
```

What should replace "missing code" so that the value of the invocation `Question1(6)` returns the value 21 ?

Consider the following function:

```
void Question4(int Q4_var1)
{
  if(Q4_var1 > 0)
  {
    Question4((Q4_var1)-1);
    System.out.print(Q4_var1);
  }
}
```

Trace: Backward Flow

What is the `Printed` value of the invocation `Question4(7)` ? Either give a number, or if it will eventually lead to an infinite recursion just write "infinite recursion", (without the quotes).

Write: Base Case

For the following recursive method, give the missing code so that when passed 2 numbers, it will find the sum of all the integers between them inclusive. Example: Given 1 and 4, add 1 + 2 + 3 + 4 and return 10. If the two numbers are equal, then return only one of them and not their summation. Example: Given 2 and 2, return 2.

```
int Question5(int Q5_var1, int Q5_var2)
{
  if ("Missing Case")
    "Missing action"
  else
    return Question5(Q5_var1, (Q5_var2)-1)+ Q5_var2;
}
```

Consider the following function:

```
int Question3(int Q3_var1, int Q3_var2)
{
  if (Q3_var2 == 1)
    return Q3_var1;
  else
    return Q3_var1 + Question3(Q3_var1, (Q3_var2)+1);
}
```

Trace: Infinite Recursion

What is the returned value of the invocation `Question3(2,3)`? Either give a number, or if it will eventually lead to an infinite recursion just write "infinite recursion", (without the quotes).

Write: A Complete Function

Write a recursive method `int BinaryToInt(String binary)` that takes a string with a binary number (it only has 1 and 0s) and returns the integer (numeric) representation of the number. Your routine must be recursive.

```
int BinaryToInt(String binary)
{
  // implementation, remember recursive and base cases
}
```

For example, the following code `System.out.println(BinaryToInt("1010"))` prints 10.

How to convert from binary to decimal: Each position in a binary number is valued at $2^{position}$. That is, the right most value is 2^0, the next right most is 2^1, etc.

Binary digit	1	0	1	0
Power of 2	2^3	2^2	2^1	2^0
Decimal value	8	4	2	1

Results

We used draft concept inventory as a pre-test Fall 2014 to measure how much students struggle with recursion.

Percentage of students that struggle with each Misconception

Misconception	Percentage
Backward Flow	80%
Limiting Case	63.4%
Recursive Case	47.6%
Forward Flow	44.1%
Confusion with loop structure	35%
Infinite Recursion	16.6%

Surveys

We gave surveys to CS instructors and students regarding their views on how well students are learning recursion using present instructional methods. The surveys identify differences between required versus actual number of hours spent by students on recursion.

Question	Count/10	Average
Course Level Taught	CS2: 9 CS3: 1	N/A
Required Recursion Background	Nothing Required: 8 Basic Knowledge: 2	2 hours
Time Spent on Recursion in Class	5 to 10 hours: 8 Can not be estimated: 2	2.5 hours
Time Required by the student to spend on Recursion	4 to 7 hours: 2 8 to 27 hours: 8	10 hours
Does students need more time on recursion?	Yes: 10 No: 0	N/A

Instructors Survey Responses

Students Survey Responses

Question	Percentage	Average
Time spent on Recursion	≤ 7 hours: 71.2% > 7 hours: 28.8	4 hours
Time spent on Coding Bat Exercises	≤ 3 hours: 71% > 3 hours: 29%	2 hours
Confidence Level (on a scale of 5)	≤ 3: 65.3% > 3: 34.7%	2.5

Recursion Tutorial

The tutorial uses greater interaction with automated feedback. It is an online tutorial designed to address the common misconceptions that the students have in recursion. It differentiates between code writing and code reading/tracing skills.

This tutorial is part of OpenDSA, a collection of open-source tutorials that make extensive use of algorithm visualizations and automatically assessed interactive exercises.

Recursion Tutorial

The tutorials presents a large number of interactive exercises, allowing students to practice with automatically assessed small-scale programming and non-programming exercises.

Programming Exercises

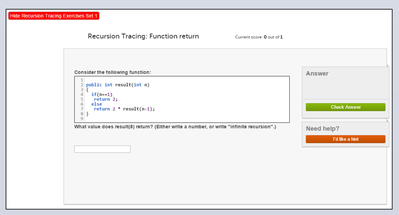
- Automatic programming assignment assessment framework
- Randomized tracing exercise instances
- Static code analysis to detect common misconceptions
- Feedback
  - Compilation errors
  - Run time errors
  - Incorrect vs Correct Answer

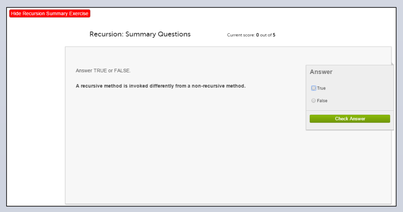
Code Completion Exercises

Harder Completion Exercises

Code Writing Exercises

Code Tracing Exercises

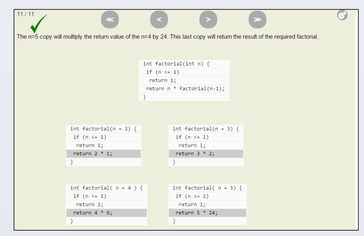




Summary Exercises

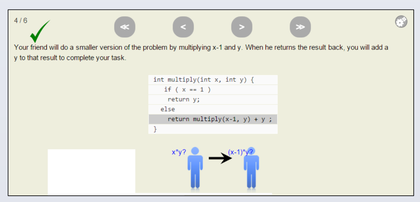
Visualizations

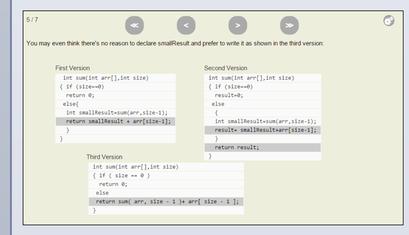
- Uses successful mental models from the literature.
- Illustrates the steps of code tracing through the copies model.
- Illustrates the steps of code writing through the delegation model.
- Addresses common recursion misconceptions.



A tracing of a factorial through copies model

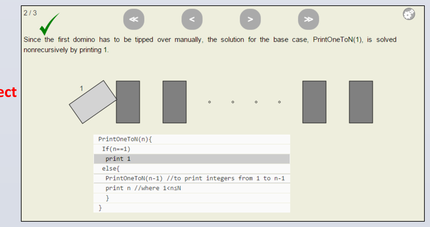
Multiply x and y through delegation model

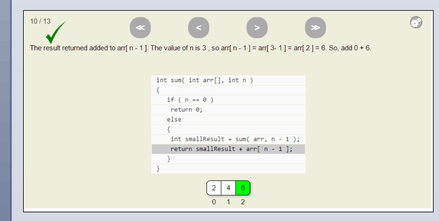




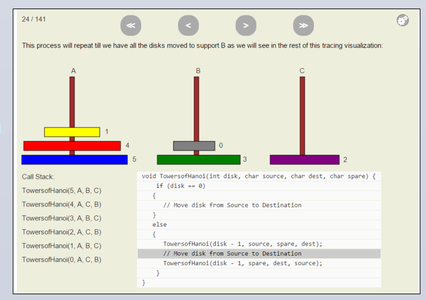
Different Versions of a Recursive Function

Recursive Domino Effect to print 1 to N





Tracing of a Sum Function



Full Tracing of Towers of Hanoi