Operational Expectations for CS1

Prepared by Dan-Adrian German
for Dec. 16, 2004

CSCI/INFO Retreat on A201 and A202

Abstract

Teaching introductory programming is[1] sometimes a challenge. One of the more objective but also acute of the difficulties encountered is the mismatch in expectations. At least one set of expectations can and should be managed effectively: those of the students with respect to the contents of the course. But how do you give a complete description of computer programming to students who register to study it for the first time? We propose a set of problems that any beginner student[2] should take as a test during the first day or week of classes. The problems are presented along with instructor's notes, and should act as a predictor test for students and instructors alike. For this exercise to be effective the students should work out the solutions mostly on their own—otherwise[3] almost all effect is lost. Relevance to the course curriculum is discussed for each problem together with a summary of potential and/or desired topics for A201.

1 Does Canada have a 4th of July?

Consider the following excerpt from Grosswirth and Salny and then try to solve the problems listed at the end of this section (on your own):

"I admit that mine is a personal and untested theory, but I believe that over the years, we have been culturally conditioned to abandon thinking as unnecessary. Everything—news, information, entertainment, medical care, food, merchandise—is provided in neat packages, predigested, preconditioned. Just add water, batteries not included. We've lost the habit of thinking for ourselves. [...]

Intelligence implies the ability to think, and an IQ score is an index of a potential. If your scores are lower than you would have liked, you are making a mistake if you believe there is nothing you can do about [it]. It has been said before, but it bears repeating: Use it or lose it.

You don't need any teachers, counselors, psychologists, or gurus to help you expand your intelligence. All you need is the desire to do so and the initiative to practice. Ask questions. Understand everything, even if

[1] ... always a pleasure but...
[2] And in fact only those labelled as absolute beginners are the group of interest here.
[3] This is also an opportunity to establish clear guidelines with respect to working in groups.

it has to be explained several times or in a new way. Give free rein to your curiosity. Most important of all, free yourself of the mental boundaries I mentioned earlier."

Consider the problems listed below. You may find them tricky, and they are. What makes them tricky is that they rely on the likelihood that your thinking is conditioned to move in only one direction with respect to the information given. Two problems are described in detail, the rest are listed at the end for your exercise.

1. The Scottish Name Gambit

   This should be done orally; it's too easy to solve when you see it in print. Ask your victim to pronounce M-A-C-T-A-V-I-S-H. After he or she has done so, try M-A-C-C-A-R-T-H-Y; then M-A-C-D-O-U-G-A-L. You can use any other "Mac" names, as long as they are legitimate. After each name has been pronounced in turn, ask the sucker to pronounce M-A-C-H-I-N-E-S. As you can see, that spells *machines*, but if your victim doesn't answer with "MacHines," it's safe to assume that he or she has fallen into the trap before. By spelling out the "Mac" names first, you will have effectively preconditioned the listener's thinking, leading it straight down a predetermined—and totally incorect—path.

2. The Time Gambit

   One more supposedly trick question, this one to demonstrate how it isn't even necessary for you to set the person up; you can simply take advantage of the preconditioning that already exists. Punctuate the following: TIME FLIES I CANT THEYRE TOO FAST. Most people will stick in the missing apostrophes immediately, but then they're stymied. Look how simple it is, once you free up your thinking (the quotation marks are a little added refinement to aid comprehension): "Time flies." "I can't. They're too fast." Still doesn't make sense? Look again. The problem is that you're already familiar with the cliche, "Time flies". In that expression, *time* is a noun and *flies* is a verb, and your conditioning leads you to assume that it has to be that way. It doesn't. Read the punctuated sentence again, but this time consider *time* as the verb and *flies* as the noun. Tricky? Only if you insist on retaining the strictures placed on your mind, on your native intelligence.

3. Practice Questions (with Answers)

   a) Under international law, if a plane crashes in the middle of the Atlantic, where would the survivors be buried?

   b) How many months have twenty-eight days?

   c) What is the next letter in this series? O T T F F S S E N.

   d) I have two coins that add up to 55 cents. One of them is not a nickel. What are the two coins?

   e) Divide 100 by 1/2 and add 10. What do you have?

   f) A snail is climbing out of a well. The well is twenty feet deep. Every day the snail climbs up three feet and every night he slips back two feet. How many days will it take to get out of the well?

g) A poor but honest knight wants to marry a beautiful princess, and she wants to marry him. The king offers the knight a choice, as follows: He can draw one of two slips of paper from a golden box. One will say "Marriage," the other "Death". The princess manages to whisper to her suitor that both slips say "Death". But the knight and the beautiful princess are wed. How did the knight accomplish this?

g) A very fast train runs from A to B in an hour and a quarter, but on the return trip it takes 75 minutes under identical conditions. Why? (Answers are provided here for the group as a whole[4].)

2 What is Programming?

2.1 A First Example

You are given two different length strings that have the characteristic that they both take exactly one hour to burn. However, neither string burns at a constant rate. Some sections of the strings burn very fast; other sections burn very slowly. All you have to work with is a box of matches and the two strings.

a) *Describe an algorithm that uses the strings and the matches to calculate when exactly 45 minutes have elapsed.*

b) *Same task when each string takes 30 minutes to burn completely.*

Instructor's notes: this problem serves a dual role. Its solution clearly marks it as an exercise in threads scheduling (pun intended) and/or parallel programming. But there is a more important role it plays: it gives the student that solves the problem (on her own) tremendous confidence.

The satisfaction that comes out of finding the solution to this problem (on one's own) is so strong it can become addictive. Because despite severe limitations and restrictions (how much is really given in the problem?) an elegant solution is immediate. It is also instructive sometimes to compare and contrast the problem above with this one:

Using six identical matches create four equilateral triangles.

Students will always find more than one solution to this last problem. This would also be a perfect place to differentiate between this type of problems and those of the kind shown before of which we list some more:

a) What five-letter English word becomes shorter when you add two letters to it?

[4] a) Nowhere. You don't bury survivors. b) Every month has twenty-eight days; most of them also have more. c) T, for Ten. The sequence is One, Two, Three, etc. d) One of them *is* a nickel: a 50 cents piece and a 5 cents piece. e) 210. If you divide a number by 1/2 you double it. f) Eighteen days. On the eighteenth day he reaches the twenty-foot level and climbs out; he doesn't have to fall back. g) The poor but honest and clever knight tears up the paper he picks, and offers the other one to the king. Since the untorn one says "Death," obviously, says the knight, the one he tore up said "Marriage" h) One hour and a quarter is 75 minutes; there is no difference. A few more examples will be given in the next section.

b) What four-letter English word becomes shorter when you add three letters to it?

c) How many bricks do you need to complete a brick house in England?

d) Take away my first letter: I remain the same. Take away my fourth letter: I remain the same. Take away my last letter and I remain the same. What am I?

e) What is the number of letters in the correct answer to this question?

f) What English word is pronounced incorrectly by more than half of the Harvard and Yale graduates?

These questions don't offer any deep insight into what programming is (which is why we list their answers right away[5]).

And perhaps these last few questions are somewhat humorous. Yet students need to know that the two types of problems discussed in this section could be equally entertaining and, under the circumstances, quite possibly *just as hard*. We need to remind the students constantly that programming is a contact sport.

2.2 The Rest of the Examples

2.2.1 The Captive Queen

> *A captive queen weighing 195 pounds, her son weighing 90 pounds, and her daughter weighing 165 pounds, were trapped in a very high tower. Outside their window was a pulley and rope with a basket fastened on each end. They managed to escape by using the baskets and a 75-pound weight they found in the tower. How did they escape? The problem is that anytime the difference in weight between the two baskets is more than 15 pounds, someone might get killed. Describe a sequence of steps that gets them down safely.*

Instructor's notes: at first this is not as clean a problem as the previous one. Issues about the pulley and the basket and many operational details always arise. But that should be a reason to encourage discussion of the problem with the clear goal of abstracting away all the irrelevant details. In any event after a certain amount of clarification the problem becomes just as unambiguous as the preceding one.

This is not a difficult problem (the previous one is). What makes this problem easy is that you can work at it and perhaps accidentally discover a solution. In that respects it's like the Tower of Hanoi problem, only not as hard. The real benefit of this problem, however, lies in the fact that the steps comprising the first two fifths of the solution need to be repeated at the end. So it would be a perfect case of defining a named procedure: a sequence of steps that you want to refer to later, by name.

[5] a) short, b) shoe (SHOrtEr), c) just one (in England and anywhere else the last one completes the house), d) a mailman, e) four (in English, it can be trickier in other languages), and f) "incorrectly" is the word.

2.2.2 A Farmer's Predicament

> *A farmer lent the mechanic next door a 40-pound weight. Unfortunately, the mechanic dropped the weight and it broke into four pieces. The good news is that, according to the mechanic, it is still possible to use the four pieces to weigh any quantity between one and 40 pounds on a balance scale. How much did each of the four pieces weigh? (Note: You can weigh a 4-pound object on a balance by putting a 5-pound weight on one side and a 1-pound weight on the other).*

Instructor's notes: both issues of *representation* and of *logic programming* can be brought up. Students are always very creative with this problem. I use it to introduce the balanced ternary system of notation (which is a non-redundant and symmetric positional number system). Once this topic is brought up it's easy to discuss the binary representation of numbers or representation in any other base. So, what are the four numbers?

2.2.3 Calculating a Test

> *Given two numbers (identified by the names x and y) can you calculate the largest of the two numbers?*

Instructor's notes: there will always be students that will try to solve this problem by *comparing* the two numbers[6]. (Although the class is for absolute beginners some students will be retaking the class, or may have tried to learn programming on their own before). And indeed one could use this problem to describe the nature (and the convenience) of the `if` statement, but for the purpose of this problem using an `if` statement to *calculate* is strictly prohibited. So, how `else` can we do it?

2.2.4 Symbolic Reasoning

For this problem the students should work in pairs. Ask students to:

a) write down detailed rules for multiplication of integers

b) do the same[7] for the calculation of square root of integers

The students should then be handing each other the rules together with some data and asked to apply the rules *exactly* to multiply the integers or calculate the square root.

Instructor notes: Most likely they will find it *very* boring to provide all the details for part (a), but having part (b) around helps balance their perspective. Hardly any one of them will remember immediately the rules for calculating the square root of an integer. Still, part (a) is instructive, especially if students work in pairs, and try to perform multiplication *exactly* as indicated in the instructions put together by their partners.

[6] But you can't. You don't know the *values* of the two numbers, you only know their *names*.

[7] And lest we forget, let it be said that a similar request should be made of the students with respect to the calculation of the greatest common divisor of two integers.

The second part of this problem is a test of their mathematical ability to understand an argument (in a more formal way). Provide students with the following instructions and ask them to:

a) use them to calculate some square root by hand, and

b) to explain if they think the rules are correct, or complete, or not?

Rules: Start by making an initial guess $x_0 > 0$ for $\sqrt{5}$. If we guessed right already (how do we know?) then we are done. But if x_0 is not the square root of 5 then x_0 is either smaller or bigger than the square root we're looking for. When it's smaller we have

$$x_0 < \sqrt{5} \tag{1}$$

and when it's bigger we have

$$\sqrt{5} < x_0 \tag{2}$$

In both cases we multiply on both sides by $\sqrt{5}$ and divide by x_0 to obtain equivalent expressions:

$$\sqrt{5} < \frac{5}{x_0} \tag{3}$$

$$\frac{5}{x_0} < \sqrt{5} \tag{4}$$

Combining that, regardless of the original guess x_0 we have either:

$$x_0 < \sqrt{5} < \frac{5}{x_0}$$

or

$$\frac{5}{x_0} < \sqrt{5} < x_0$$

Therefore, if we take the average of x_0 and $\frac{5}{x_0}$, namely

$$x_1 = \frac{1}{2}\big(x_0 + \frac{5}{x_0}\big)$$

the result must be closer to $\sqrt{5}$, which is what we're after. Now what?

2.2.5 Recursion and Analytical Geometry

Given two distinct points in the plane $A(x_0, y_0)$ and $B(x_1, y_1)$

a) determine their midpoint[8]

b) determine a way to draw point by point the *entire* segment AB

Instructor's notes: this is first an exercise in (minimal) analytical geometry. Intersections between two straight lines expressed parametrically could also be discussed eventually, but this is not the real aim of this problem. I use it to introduce both recursion and notions of 2D graphics, at the same time.

[8]The point $M(x, y) \in AB$ that's at the same distance from A as it is from B

2.2.6 Repetitive Behavior

Given a sequence of numbers $\{x_1, \ldots, x_n\}$ use the formula below

$$S = \sqrt{\frac{\sum {x_i}^2 - \frac{1}{n}\left(\sum x_i\right)^2}{n-1}}$$

to calculate their standard deviation.

Instructor's notes: the main goal of this problem is to check if students can read simple math. The truth is that some cannot—and giving them a heads up is very much appreciated by the students. This problem also discusses iteration and loops. Note however that the final emphasis should be on how we describe what we do, rather than doing it. From here on the problem can branch out as follows:

a) use formula above to calculate standard deviation in one pass

b) convert the formula above to the definition and use "arrays" (sequences) of empty squares on paper to calculate in pencil: first the average, and then the sum of squared deviations to the mean

Simple unidimensional "arrays" are being introduced and used.

2.2.7 Magic Squares

Implement the following procedure to construct magic n-by-n squares; it works only if n is odd. Place a 1 *in the middle of the bottom row. After k has been placed in the* (i, j) *square, place* $k + 1$ *into the square to the right and down, wrapping around the borders. However, if the square to the right and down has already been filled, or if you are in the lower right corner, then you must move to the square straight up instead. Here's the* 5×5 *square that you get if you follow this method:*

11	18	25	2	9
10	12	19	21	3
4	6	13	20	22
23	5	7	14	16
17	24	1	8	15

Check that the square above is magic.

Instructor's notes: this serves as a brief introduction to two-dimensional "arrays". Emphasis, this time, is on being able to follow the steps—being able to understand the data structure, and use it effectively. Everything should be done in paper and pencil. Of course, the true benefit of these problems is that no programming is required to solve any of them, but skills akin to those used later in programming are being warmed up and worked out. So here, just like in the previous problems, the emphasis is on calculations (usually done by hand) with a verifiable result at the end, but also on writing clear instructions and determining the vocabulary of actions allowed.

Thus programming is really the luxury of delegating responsibility (in some sense). But one can only do it if one is meticulous and very precise.

3 What is CS1?

A CS1 course should start from values, simple expressions and variables. The notion of procedure should be introduced next. Conditional statements add to the expressiveness of procedures (little recipes with ingredients, a name, and expected outputs) and recursion is an immediate candidate for discussion. Syntactic constructs for iteration (loops) are tools of increased efficiency. Variables could be grouped in one and two dimensional arrays to provide vast and more uniform access to a collection of values. When an object-oriented language (such as Java) is used the situation is not complicated in any way. In the case of Java IDEs like DrJava or BlueJ can be used (with a plus for DrJava in the beginning).

There isn't anything else in programming but variables and procedures. A class is a container, that holds a blueprint. The blueprint is used to create objects (like a cookie cutter). In Java procedures and variables can be placed only in classes or objects and this is indeed a most fortunate situation. Because if we (as instructors) come up with the following taxonomy of variables (and explain it): local, parameter, instance and class (static), and if we encourage a naming scheme that takes into account the kind of variable (class and instance variables should always be named using the class or object name to which they belong, followed by a dot, followed by the name of variable) then at least a naming convention will be in place that will completely eliminate ambiguities and object-oriented programming will appear to the CS1 student as what it really is: simply a style[9] of programming.

This naming convention has the advantage of being immediately transferrable to methods. But, being a convention only, it can't be enforced, it can only be encouraged. However one of the more interesting consequences of this convention is that it eliminates any mystery with respect to the meaning of the `this` reference. (How else could we refer (from an instance method point of view) to an instance variable in the same object if the naming convention that we are promoting here is respected?)

Other topics are not too difficult for CS1 (inheritance is as easy to understand as the set union of features, modulo the inevitable details) but it is our belief that CS1 should be used for a thorough grounding of the student in the core concepts. There is plenty of time in CS2 to commit one's energy to learning new concepts (interfaces, abstract classes, event handling, basic graphics) and CS1 could and should offer a glimpse of those. But the emphasis and testing should be on the main ideas behind programming[10].

[9] Object-orientation pays off in code reuse (useful in large projects although it can be illustrated in small code samples as well) and in parallel and distributed computation. But these are not very good topics in CS1 (they can't be studied in detail in a CS1 context, that is, and they can only be *introduced* meaningfully). There is also a tendency to exaggerate somewhat by trying to focus more than necessary on Java. Java is secondary here, no matter how we look at it, and no matter how much we like it—and we do like the language.

[10] See, for example the 1972 "Collection of Programming Problems and Techniques" by Maurer and Williams published by Prentice Hall. A truly refreshing book for our times.