More than the Rules of Precedence

Yawei Liang[*] *Royal Military College of Canada*

Abstract: In a fundamental computer-programming course, such as CSE101, questions about how to evaluate an arithmetic expression are frequently used to check if our students know the rules of precedence. The author uses two of our final examination questions to show that more knowledge of computer science is needed to answer them correctly. Furthermore, this part of knowledge isn't listed as any part of the Computing Curricula [1] for a course like CSE101, published by The Joint Task Force on Computing Curricula, IEEE Computer Society, and Association for Computing Machinery. The following presents a suggestion that will add some correlated knowledge of computer science to a course like CSE101.

Key words: computer science education issues computer science curriculum issues

1. Introduction

The rules of precedence refer to the order in which operations are carried out in a complicated expression. By investigating two of the examination questions from CSE101, a course regarding to the fundamentals of computer programming in Java, and then analyzing the answers from students, we realize that students may need to know more than just the basic rules of precedence that we taught in class. Moreover, to prepare them to answer questions like those, this paper suggests that some extra knowledge, such as context free grammar, might be the missing part in CSE101 curricula. In addition, the correlation of knowledge from the body of computer science must be clearly established to support the successful and thorough teaching of a course similar to CSE101.

2. An Examination Question

In CSE101, a fundamental computer programming course in Java, questions concerning how computers evaluate arithmetic expressions are frequently asked (Davies, Richard, 1999;Koffman, Elliot, B., and Wolz, Ursula, 2002;Eckel, Bruce, 2000;Chapman, Stephen, 2000). The following samples were taken from one of the previous final examination papers.

[Question 1] Given the following variable declarations in Java:

```
double x = 3.1416;
double y = -1.0;
double z = 2.0;
int p = 3;
int q = 4;
```

Give the exact value of the following expressions if they are valid, and identify those that are not, if any.

(a) (p / q * z)

(b) ((int)x * y + p)

A sample answer:

[*] Yawei Liang, Ph.D., professor at the Department of Mathematics and Computer Science, Royal Military College of Canada; Research field: artificial intelligence, modeling and simulation, parallel computing; Address: P. O. Box 17000, Station Forces, Kingston, On., K7K 7B4, Canada.

(a) 3/4*2.0 = 0*2.0 = 0.0

(b) ((int)3.1416*(-1.0)+3) = 3*(-1.0)+3 = -3.0+3 = 0.0

Initially, these questions did not appear to be very difficult, provided that the knowledge of the rules of precedence has been taught in the CSE101 course. However, after analyzing the answers from seventy-five students, we found a high rate of incorrectness, see Figure 1(a) and 1(b). Note that the students were in their first year from engineering and science departments. They were taking CSE101 in the second term of a two term academic year, and CSE101 is a one-term course.

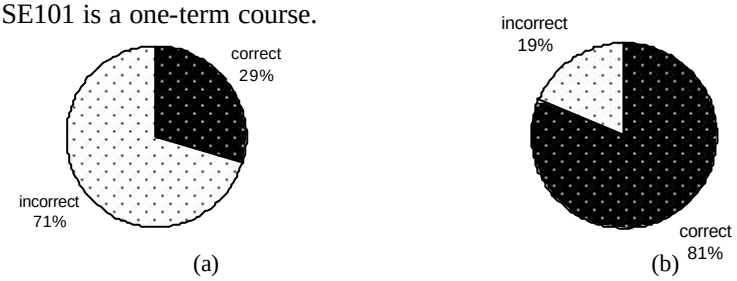

Figure 1 The Percentage of Correct Answers to Question 1, (a) and (b)

The high incorrectness of the answers for Question 1(a) must result from two possible causes: either we missed some part of knowledge in teaching, or students ignored some part of the required knowledge needed for answering these questions correctly. As a professor who taught this course, I would like to explore the former, taking into account that a part was missing in the teaching of CSE101. A detailed analysis will be given in the next section focusing on what extra knowledge, in addition to the rules of precedence, should be taught to our students in order to answer questions similar to Question 1 (a) and (b).

3. What We Taught to Answer the Questions

After further examining how students answered Question 1 (a) and (b), we are able to see why they answered (a) with a very low rate of correctness. In Question 1 (a), there were 31 out of the 53 incorrect answers (59%), which answered with the expression:(3/4)*2.0.

However, when they calculated this expression they concluded that it yields 1.5, a double value. It is believed that if the students know the rules of precedence, then they should be able to automatically figure out how the expression has to be evaluated. But most of them failed to do so. Then, what are the rules of precedence?

The Rules of Precedence refers to the order in which operations are carried out in complicated expressions. Table 1 gives the order from the highest precedence to the lowest.

Table 1 The Rules of Precedence (Liang, Yawei, 2005)

Precedence	Operator
Highest (evaluated first)	Method call
	Type cast
	!, +(unary), -(unary)
	*, /, %
	+(addition), -(subtraction)
	<. <=, >=, >
	= =, !=
	&&
	‖
Lowest (evaluated last)	=

We know that when there is more than one operator in an expression, the rules of precedence will help us determine which operation must be evaluated first. When there is more than one type of operands involved, computers have to convert them into one type which have higher accuracy. But how does a type conversion occur? Are they converted in one go for all the terms in an expression if there is a mixture of types in the expression? Obviously, most of the students thought so. Therefore, (3/4)*2 becomes (3.0/4.0)*2.0, which did yield 1.5 under this assumption.

This might also answer the question why the rate of correctness for Question 1 (b) was much higher, if the students change, in one go, the types of mixture data into one type which has the highest capacity to keep the precision:

(int)3.1416*(-1.0)+3 = 3*(-1.0)+3 = 3.0*(-1.0)+3.0 = 0.0

In this instance, they have the correct answers.

4. What is the Part of Missing Knowledge that Students Needed to Answer the Question?

One possible missing part of knowledge required to correctly answer Question 1 might be the part concerning context free grammar, which explains how a computer evaluates an arithmetic expression.

Context Free Grammars (CFGs) refer to a method of describing languages. CFGs are first used in the study of human languages. They have an important role in the specification and compilation of programming languages (Sipser, Michael, 1997; Savage, John, E., 1998). The following illustrates a simplified example showing how a CFG is used to evaluate an arithmetic expression.

[Example] Write a program, which evaluates an arithmetic expression typed by the user. The expressions can use positive real numbers and the binary operators +, -, *, and /. The unary minus operation is supported as well.

First of all, we define the context free grammar rules:

E ? [“-”] T [[“+” | “-”] T]...
T ? F [[“*” | “/”] F]...
F ? <number> | “(” E “)”

Where E represents an expression, T represents a term, and F represents a factor. A number <number> should start with a digit (i.e., not a decimal point, even though Java allows a number starting with a decimal point). A line of input expression from a user must contain exactly one such expression. If extra data is found from a line after an expression has been read, it is considered an error. A pair of square brackets includes a part of an expression that might be repeated 0 or 1 time. The algorithm listed below explains how a program works by following the context free grammar rules.

Step 1:

Enter an expression.

Step 2:

If an empty line is read, halt; else turn to Step 3.

Step 3:

Turn to Method 1 to evaluate the input expression. If there is not further characters after the expression, and then, return the value of the expression; else send an error message informing an extra data found at the end of the expression.

Step 4:

Halt.

Method 1: evaluate an expression.

Step 1:

If the first character of the input expression is a minus sign, turn to Method 2 and return value = - (term value). Else, turn to Method 2 and return value = term value.

Step 2:

If the next part of the expression is a "+" sign, get the next part (term) and turn to Method 2, and return value = value + term value; else get the next part (term) and turn to method 2, and return value = value – term value.

Method 2: evaluate a term.

Step 1:

Turn to Method 3, return value = factor value.

Step 2:

If the next sign is "*" or " /" , get the next part (factor) and turn to Method 3, and return value = value * (factor value) or value = value / (factor value).

Method 3: evaluate a factor.

Step 1:

If next part is a number, return it;

Else if the next part is a "(", turn to Method 1, and return expression value if there is a ")" following, but if next part is any signs other than ")", send an error message.

From this algorithm, one can clearly see that when computers evaluate an expression, it evaluates one operation at a time. This provides a piece of crucial information for our students: a computer does not evaluate an expression in one go, but it does evaluate an operation at a time. So any mixture of types in an expression will be evaluated operation by operation, i.e., type conversions will occur operation by operation, but not in one go.

For example, the expression from Question 1 (a) will be evaluated in this order: (3/4) will be evaluated first, and it will yields 0 since 3 and 4 are all int type in Java. Then, 0*2.0 will be evaluated; there, a common type will be used for the data involved, that is, 0.0*2.0, which yields 0.0.

5. Suggestions to Improve the Computing Curricula

Check the Syllabus for CSE101 from [1] and further exploring the units covered by various computer science body of knowledge are listed in [1]. Also, it is not hard to find where the piece of knowledge about the context free grammar will fit. The author suggests adding two core hours to explain how a computer expresses the grammar of an expression, and how it evaluates it. Without this part of knowledge, students will not be able to fully understand why they could not convert all mixed types of data to its highest precision term in one go, and some time, for example in case of Question 1 (a), they will not be correct.

6. Conclusions

This paper uses a previous examination question and its answers to show that when teaching a course like CSE101, certain parts of the computer science knowledge that must be specifically explained. This paper suggests an ad hoc way to tackle this problem. Systematically, we need a way to express and connect each piece of key knowledge of CSE101 to its correlated knowledge pertaining to computer science. Only then, will there be a comprehensible guideline for us to follow; furthermore, this allows us to teach the relevant knowledge before we

challenge our students with questions such as Question 1(a) and (b).

References:

1. The Joint Task Force on Computing Curricula IEEE Computer Society Association for Computing Machinery. (2001). *Computing Curricula 2001 Computer Science, STEELMAN DRAFT*, August 1

2. Davies, Richard. (1999). *Introductory Java for Science and Engineers*, Addison Wesley

3. Koffman, Elliot, B. & Wolz, Ursula. (2002). *Problem Solving with Java*, 2nd Edition, Addison Wesley

4. Eckel, Bruce. (2000). *Thinking in Java*, 2nd Edition, Prentice Hall

5. Chapman, Stephen. (2000). *Java for Engineers and Scientists*, Prentice Hall

6. Liang, Yawei. (2005).*CSE101, Introduction to Algorithms and Computing, Course Notes*, 2nd Edition, Royal Military College of Canada, April

7. Sipser, Michael. (1997). *Introduction to the Theory of Computation*, PWS Publishing Company

8. Savage, John, E. (1998). *Models of Computation, Exploring the Power of Computing*, Addison Wesley

(Edited by Saihu Xu, Dongling Zhang and Junqing Zhang)