Design and Use of a Visualization for Teaching Integer Coercion

Steven Carr  
steve.carr@wmich.edu  
Western Michigan University  
Kalamazoo, MI, USA

Yu Chin Cheng  
yucheng@ntut.edu.tw  
National Taipei University of Technology  
Taipei, Taiwan, ROC

Yu-Hsiang Hu  
t109599006@ntut.edu.tw  
National Taipei University of Technology  
Taipei, Taiwan, ROC

Jean Mayo  
jmayo@ntu.edu  
Michigan Technological University  
Houghton, MI, USA

Ahmed Radwan  
ahrenabdelwaha.radwan@wmich.edu  
Western Michigan University  
Kalamazoo, MI, USA

Ching-Kuang Shene  
shene@mtu.edu  
Michigan Technological University  
Houghton, MI, USA

James Walker  
jwwalker@mtu.edu  
Michigan Technological University  
Houghton, MI, USA

ABSTRACT
The C language is used to develop software that implements fundamental mechanisms used by higher level software to protect data. Yet C continues to be difficult for students to understand and use securely, and integer errors continue to create vulnerabilities. In fact, Integer Overflow or Wraparound is listed at position 11 in the 2020 CWE Top 25 Most Dangerous Software Weaknesses.

This paper presents the Expression Evaluation (EE) visualization tool that helps students understand the type conversions that take place implicitly within a C program. This tool depicts step-wise the coercions that take place within the compilation of an expression with mixed integer type operands. This enables students to create unlimited examples to test their understanding.

We present the results of our evaluation of EE in both a lower-level class and an upper-level class. We also present the results of an expanded evaluation of a complementary integer security education tool Integer Representation (IR) in these same classes. This represents evaluating IR across a wider student audience; prior evaluations of the IR tool were within classes focused on low-level programming. Our evaluation results showed that students in an upper-level course improved their understanding in both IR and EE more significantly than students in a lower-level course. As shown by the data collected from both classes, our tools were easy to use and very effective.

CCS CONCEPTS
- Social and professional topics → Computing education.

KEYWORDS
Integer security; visualization; cybersecurity education

ACM Reference Format:

1 INTRODUCTION
The C language is widely used for software that implements basic mechanisms used to keep data safe. Examples include operating systems, databases, and embedded systems. At the same time, most undergraduate programs have moved away from using C widely, in part because of its complexity.

In order to help address a need for graduates proficient in secure C programming, to facilitate deeper coverage of C security issues in the time available within a crowded undergraduate curriculum, and to encourage self-study, we have created the Visualization and Analysis for C Code Security (VACCS) system to help students learn secure coding in C. This system contains several perspectives on a program execution to illustrate coding errors that can create vulnerabilities. The system connects the code that students write to the effect it has on memory values during execution. VACCS currently focuses on integer errors and memory errors. A section on kernel file structures is under development that will connect file operations to common errors like improper use of symbolic links and leaking of open file descriptors.

Integer errors continue to create vulnerabilities. In fact, Integer Overflow or Wraparound is listed at position 11 in the 2020 CWE Top 25 Most Dangerous Software Weaknesses. Hence we have created two tools within VACCS that focus on helping students learn to avoid integer errors. We have previously described the Integer Representation (IR) feature of VACCS [15]. In this paper, we present a complementary representation that shows the effect of operations among mixed integer types. We call this the Expression
Evaluation (EE) representation. The EE visualization depicts the resulting type of each operation in an expression. It also allows students to construct an unlimited number of examples to further their understanding of integer promotion and the usual arithmetic conversions. The tool is very convenient for lecture.

There are many pedagogical tools for visualization of security concepts, including network protocols and defenses [4, 10, 18, 20], formal models [5, 9, 17] and cryptography [2, 6, 8, 11–13]. Benham proposed a method to teach data representation using a modular architecture [1]. There are visualization systems aimed to teach computer architecture [14, 19]. The EE representation focuses only on integer coercion.

In this paper we describe the EE tool and present the results of our classroom evaluation. Unlike our earlier work [15, 16], the evaluation was performed in a lower-level course and in an upper-level course with a much broader focus than low-level programming or security. We also report the results from an evaluation of the IR feature within this broader audience.

2 MOTIVATION

While one of the authors was visiting National Taipei University of Technology (NTUT) teaching a Concurrent Computing course, he tried to assess students' knowledge in interrupts (i.e., overflow, underflow, etc.). A very simple programming exercise was assigned asking students to compute the factorials from 1! to 25! using long and report their findings in a programming assignment. The same problem was repeated in an exam. It was a surprise that not many students were able to answer this problem accurately. Most students just indicated that the numbers were too large to be stored in a long and some provided very vague statements about some large factorials being negative. Because not many students had a computer organization course before this course, it is not a surprise. As a result, we used our integer representation [15] and expression evaluation materials in this course, visualization tools included. NTUT people joined force with us and also used the same materials in a lower end object-oriented programming course. In this way, we are able to evaluate our materials across students from freshmen to graduate students, a wide range of samples.

3 VISUALIZATION TOOLS: IR AND EE

The EE tool helps student to determine when a coercion will take place and identify the resulting type. A complementary tool, the integer representation visualization tool IR, depicts the effect of a coercion on the underlying binary representation and the resulting decimal value. The integer representation visualization tool IR was discussed in detail in [15]. Due to space limitations, it is not described in detail here. Together, these two tools are the primary contribution of VACCS to help students avoid integer errors.

The Expression Evaluator EE tool includes a set of slides and a visualization tool eevis explaining how type conversions are performed in evaluating a mixed integer type expression. Because all type conversions are carried out implicitly by the compiler and are hidden from users, without a solid understanding how these type conversions are performed a beginner may be surprised and commonly fail to find errors. To overcome this problem, a set of slides carefully explain how integer type conversions are done based on the C standard, and eevis is used to "reconstruct" the steps a compiler would perform to carry out these conversions step-by-step.

The material we cover closely matches the description from the INT-02 recommendation of the SEI Cert C coding standard [7]. Type conversions occur explicitly in C and C++ as the result of a cast or implicitly as required by an operation. Implicit conversions are a consequence of the C language ability to perform operations on mixed types. The C integer conversion rules define how C compilers handle conversions: integer promotions are performed automatically, integer conversion rank is used by usual arithmetic conversions, and usual arithmetic conversions are rules for conversion after promotion.

Promotions. Integer types smaller than int are promoted when an operation is performed on them. If all values of the original type can be represented as an int, a value of smaller type is converted to int. Otherwise, it is converted to unsigned int.

Rank. Each integer type has a conversion rank used to determine how usual arithmetic conversions are performed. Table 1 has the ranks of the integer types from high to low, and the types on each row have equal rank.

<table>
<thead>
<tr>
<th>operand type</th>
<th>long long int, unsigned long long int</th>
<th>long int, unsigned long int</th>
<th>int, unsigned int</th>
<th>short int, unsigned short int</th>
<th>char, signed char, unsigned char</th>
</tr>
</thead>
</table>

Conversions. eevis is a simple visualization tool designed to make type promotions and conversions explicit. The left side of the eevis window (Figure 1) allows the user to input a variable, its type and its initial value, and an expression built from these variables. The user may also save the expression and reload it later. The right side of the window shows a flow chart of the promotion and conversion rules with rule numbers clearly shown (Figure 2).

Figure 1: The eevis Window

Users click the Add Variable button to enter the name of a variable (var_name), use the pull-down menu int to select an integer type (Figure 3), and enter a value for that variable (Enter Value).

Then a user can enter an assignment statement in the Create Custom Assignment Statement (e.g., A = B + C) field. Only the four binary operators +, -, *, and / and the unary operator - are supported, and explicit type casting is not currently supported.¹

¹Note that a user can easily change the type of a variable to explore the effect of any type for a variable.
An expression can be saved with the Save button or loaded from disk using the Load Equation button (Figure 1). After clicking the Evaluate button, eevis compiles the expression and shows the results with two tables (Figure 4). The top one is a summary of the variables used and their types and values, and the bottom one has all the promotion and conversion details.

Figure 4 shows the assignment of a signed short value -32768 to an unsigned int. It is clear that the signed short is promoted to unsigned int, converting the original value -32768 to 4294934528 by extending the sign bit. This intermediate value is stored in a temporary variable %t0 which is saved to the unsigned int variable x. This is simple enough, but it is a shocking example to many students who are not aware of the rules for integer representation, type promotion and conversion.

Consider a little more complex example. The following is a simple programming segment. What is the result in x?

```c
int x;
unsigned char uc = 100;
unsigned int ui = 123;
long l = 256;
x = uc + (ui + l);
```

Figure 5 has the output from eevis. It shows that the type of uc is converted to int and then to long because l is a long type variable. Variable ui is converted to long and is added to l to get a long type result, which is used to compute the sum of the converted to long type of uc. This long type is converted back to int, the type of variable x, by removing all higher order bits.

Therefore, the user should be able to construct a type conversion tree (Figure 6). Currently, this construction must be performed by the user; however, we intend to add this feature in a future version of eevis. The rules used for constructing this conversion tree are available from eevis's output (Figure 5).

4 EVALUATION

4.1 Testing Environment

IR and EE materials and visualization tools were used in two classes at NTUT, namely: Object-Oriented Programming (OOP) which is the second programming course taught to freshman of the Electrical Engineering and Computer Science Department, and Concurrent Programming (CP) which is the second programming course taught to freshman of the Electrical Engineering and Computer Science Department.
Computing (CC) which is an upper-level elective one offered to junior students and up, MS and PhD students included. Both classes had the same TA who helped set up the environment of the software, collect the test data and participate in analysis.

4.2 Testing Procedure and Design
A complete testing cycle took more than six weeks. It started with a Pre-Test 1 that measures how good the students were at a particular topic (i.e., integer representation and expression evaluation). A one-hour lecture was delivered, and one week later the students received a Pre-Test 2. Then, the visualization tool was discussed, and one week later the students received the Post-Test. Along with the Post-Test there was an exit survey to evaluate the effectiveness, usability and quality of the visualization tool and the materials used for that component. In this way, Pre-Test 2 measures the effect of the lecture of a topic, and the Post-Test measures the cumulative impact of the lecture and the visualization tool.

Due to COVID-19, everything was significantly slowed down. As a consequence, the discussion of EE visualization had to be scheduled at the beginning of the last week of the Spring 2020 semester, and its Post-Test and the survey were conducted after the final exam. This had a negative impact as students spent their time for the finals and less time on this extra work; the participation rate dropped and performance was not as good as anticipated.²

4.3 Samples and Background
Most students had not taken any computer organization-related courses. Many students knew something about integer representation and expression evaluation from their programming experience and courses in C/C++ and Java. For each component, all tests used the same set of questions and no solutions were provided to students, because we hoped to gauge the progress of each student in a paired way. This fits the repeated measure ANOVA; but paired non-parametric hypothesis tests can also be used.

Even when bonus points were offered, not all students finished all three tests, and only those who finished all tests were included. Each test had 15 problems, from simple to a bit difficult. Each problem was assigned one point and graded in an all-or-nothing way. The maximum score for each test is 15 points.

4.4 Assessment: The IR Component
Table 2 is a summary of the statistics collected from the IR component, where μ, σ, η, and p are the mean, standard deviation, median and the p-value obtained from the Anderson-Darling normality test. OOP and CC are the datasets from the Object-Oriented Programming and Concurrent Computing classes, and -1, -2 and -Post indicate the first pre-test, the second pre-test and the post-test. It is clear that normality cannot be rejected for the datasets OOP-1, OOP-2, and CC-1. It is interesting to note that once students learned the concept and after using the visualization tool, the score distributions became not normally distributed and skewed to the left with a long tail, meaning a significant improvement.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size</th>
<th>μ</th>
<th>σ</th>
<th>η</th>
<th>p-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOP-1</td>
<td>45</td>
<td>6.78</td>
<td>3.75</td>
<td>7.0</td>
<td>0.186</td>
</tr>
<tr>
<td>OOP-2</td>
<td>45</td>
<td>8.38</td>
<td>3.96</td>
<td>9.0</td>
<td>0.105</td>
</tr>
<tr>
<td>OOP-Post</td>
<td>45</td>
<td>9.00</td>
<td>4.35</td>
<td>10.0</td>
<td>&lt;0.005</td>
</tr>
<tr>
<td>CC-1</td>
<td>28</td>
<td>7.79</td>
<td>4.82</td>
<td>6.5</td>
<td>0.063</td>
</tr>
<tr>
<td>CC-2</td>
<td>28</td>
<td>11.39</td>
<td>3.91</td>
<td>13.0</td>
<td>&lt;0.005</td>
</tr>
<tr>
<td>CC-Post</td>
<td>28</td>
<td>12.75</td>
<td>2.98</td>
<td>14.0</td>
<td>&lt;0.005</td>
</tr>
</tbody>
</table>

After introducing the concept of integer representation, students in both classes improved their understanding significantly. The means of OOP-1 and CC-1 increased to 8.38 and 11.39 from 6.78 and 7.79, and the medians increased to 9 and 13 from 7 and 6.5. Adding visualization did not improve much. This is because students had learned all the basics of integer representation and adding visualization only further reinforced their understanding. The improvements in the CC class were higher than those in the OOP class, which is understandable because upper-level students are more mature and have more experience.

Because all test scores were paired and because half of the datasets were not normally distributed, non-parametric methods were used for hypothesis testing, namely: the Kruskal-Wallis (KW) test, Mood’s Median (MM) test and Mann-Whitney (MW) test. The p-values calculated by KW and MM for OOP-1 vs. OOP-2 and OOP-2 vs. OOP-Post were all less than 0.005, indicating that the null hypothesis should be rejected. Hence, the improvements from OOP-1 to OOP-2 and from OOP-2 to OOP-Post were significant. Even though the medians of CC-1 and CC-2 looked different, KW (p = 0.083) and MM (p = 0.067) both suggested weak evidence against the null hypothesis. However, MW provided p = 0.006, clearly suggesting a significant difference in medians. In the case of CC-2 vs. CC-Post, MM’s calculated p = 0.005 showed a strong evidence against the null hypothesis. Hence, the improvement after using visualization was significant. Finally, the Friedman repeated measure ANOVA tests reported p < 0.005 for both the OOP and CC datasets, suggesting the overall improvements were significant.

Figure 7: Boxplots of the IR Datasets

²Some many suggest that we should use a control vs. experimental approach. However, there are two major reasons for not using it. First, it is almost impossible to divide the class into two groups under the impact of COVID-19 restrictions so that one group received the treatment (i.e., IR and EE training) and the other did not. Second, this approach does not fit our goal. As we all know, teaching something new to a class always yields some effect. However, this does not tell us how significant the effect is. This is the main reason for us to use the pre-test/post-test approach. Moreover, visualization alone without a discussion of what is being visualized makes no sense. In our design, the second pre-test has the effect of the lecture, and the post-test has the total effect of the lecture plus visualization, which is incremental. The visualization component should not be taken out of this context.
Figure 7 shows boxplots of all datasets, where black circles are medians, circles with a cross are means, and circles are data points. The median of CC-1 was the lowest and its mean was slightly higher than that of the OOP-1, and CC-1 was the only dataset with its median less than the mean. After the introduction, the median and mean of CC-2 went up sharply, and they increased further after using visualization. The 50% box of CC-Post was the smallest with the third quartile at the maximum, suggesting that the performance of students in CC at the end of this component was very strong. Because the sample sizes were not equal, the Mann-Whitney test was used to compare OOP and CC. The $p$-values for OOP-1 vs. CC-1, OOP-2 vs. CC-2 and OOP-Post vs. CC-Post were 0.460, 0.001 and 0.000. Hence, class CC out performed class OOP, which is expected because CC was an upper level course.

### 4.5 Assessment: The EE Component

Table 3 and Figure 8 have a summary of the statistics collected from the EE component. Because the datasets were not normally distributed and non-parametric tests must be used. It is also clear that the OOP class improved from OOP-1 to OOP-2 and from OOP-2 to OOP-Post. The CC class also improved from CC-1 to CC-2; however, the mean and median of CC-2 were down a little from 11.20 and 12 (CC-2) to 10.60 and 10 (CC-Post).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size</th>
<th>$\mu$</th>
<th>$\sigma$</th>
<th>$\eta$</th>
<th>$p$-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOP-1</td>
<td>38</td>
<td>3.66</td>
<td>2.11</td>
<td>3.0</td>
<td>0.008</td>
</tr>
<tr>
<td>OOP-2</td>
<td>38</td>
<td>6.79</td>
<td>2.57</td>
<td>7.0</td>
<td>0.026</td>
</tr>
<tr>
<td>OOP-Post</td>
<td>38</td>
<td>9.08</td>
<td>3.44</td>
<td>9.0</td>
<td>0.030</td>
</tr>
<tr>
<td>CC-1</td>
<td>25</td>
<td>5.04</td>
<td>2.30</td>
<td>5.0</td>
<td>0.010</td>
</tr>
<tr>
<td>CC-2</td>
<td>25</td>
<td>11.20</td>
<td>3.38</td>
<td>12.0</td>
<td>&lt;0.005</td>
</tr>
<tr>
<td>CC-Post</td>
<td>25</td>
<td>10.60</td>
<td>3.29</td>
<td>10.0</td>
<td>0.027</td>
</tr>
</tbody>
</table>

Table 4: Cohen’s $d$ of the IR Component

<table>
<thead>
<tr>
<th>Class</th>
<th>Test 1 $\rightarrow$ Test 2</th>
<th>Test 2 $\rightarrow$ Post</th>
<th>Test 1 $\rightarrow$ Post</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOP</td>
<td>0.415</td>
<td>0.150</td>
<td>0.547</td>
</tr>
<tr>
<td>CC</td>
<td>0.822</td>
<td>0.391</td>
<td>1.240</td>
</tr>
</tbody>
</table>

Table 5 has the $d$ values for the EE component. The three effect sizes of the OOP class were all high. As discussed in Section 4.5, the medians of Pre-Test 2 and Post-Test of the CC classes were statistically indifferent, and the effect size was small. It is interesting to note that both classes had rather high and similar overall effect sizes. This is unlike the IR component, where the CC class showed a higher effect size than that of the OOP class.

<table>
<thead>
<tr>
<th>Class</th>
<th>Test 1 $\rightarrow$ Test 2</th>
<th>Test 2 $\rightarrow$ Post</th>
<th>Test 1 $\rightarrow$ Post</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOP</td>
<td>1.322</td>
<td>0.755</td>
<td>1.902</td>
</tr>
<tr>
<td>CC</td>
<td>2.131</td>
<td>-0.180</td>
<td>1.956</td>
</tr>
</tbody>
</table>

### 4.7 Exit Surveys

There were two sets of the exit survey, one for IR and the other for EE, asking students to rate and comment on the materials used in class. Each survey was conducted in the OOP and the CC classes after the Post-Test. Problems in the exit survey were divided into four categories: (1) evaluating the tool in general terms, (2) evaluating the components of the tool, (3) reporting the use such as number of times and number of minutes of the tool, and (4) open questions such as installation issues, general comments and suggestions. Answers to the first two categories were: 1: Strongly Disagree, 2: Disagree, 3: Neutral, 4: Agree and 5: Strongly Agree. The exit surveys for IR and EE will be discussed separately and use $Q_1|\eta|Q_3$ to denote the first and third quartiles $Q_1$ and $Q_3$, and the median $\eta$. The ranges $[Q_1, \eta]$ and $[\eta, Q_3]$ contain 50% of the data points.

#### 4.7.1 IR Exit Survey

IR’s exit survey has four categories: (1) questions asking students to answer whether (a) IR helped see how integer type conversion are performed, (b) the student understood...
integer conversion better after using IR, (c) the student was able to identify concepts that were not understood, and (d) IR enhanced the course. It is interesting to point out that OOP and CC had the same results for these four questions and the results were 3/4/5. Therefore, more than 75% students believed that IR helped them improve their understanding in integer representation and about 50% students rated IR highly (i.e., 4 or 5).

The next five questions asked students to evaluate each component of IR: (i) the IR window was easy to read, (ii) the IR interface was intuitive, (iii) the depiction of the range of values for signed and unsigned integers was easy to read, (iv) the depiction of the range of values for signed and unsigned integers helped me understand conversion between integers types, and (v) the Increment and Decrement functions were useful. Evaluation from both classes was 3/4/5 for questions (i) and (v). For question (ii), the IR window being intuitive, the survey result was 3/3/4. However, the histograms indicated that students chose Neutral more often, but the distributions were skewed left. For question (iii), class CC offered very concentrated answers 4/4/4 (i.e., more than 75% students rated the depiction of the range of values highly). The CC class rated question (iv) even higher at 3/3/5, meaning more than 50% students rated this function very highly.

The last three questions asked students to report (I) the time needed to understand the integer conversion using IR, (II) the number of times the student used IR, and (III) the total time the students used IR. The medians of the OOP class were: (I) five to ten minutes, (II) one to three times, and (III) five to 15 minutes. The CC class reported the same for (I) and (III) but students only used IR once. This is because students in CC have the experience in using a simple system. However, the total time reported was still the same (i.e., 5-15 minutes). We also looked at the correlation between the total time of using IR and other questions, and found that the Spearman ρ’s were moderate and all below 0.5.

Students in the OOP class did not provide many comments on IR. Some of them reported known bugs. Most of them hoped to see pop-up windows to explain the use of IR. On the other hand, the CC class offered more suggestions. Many of them praised IR being useful (e.g., clarifying some misunderstanding) and helpful (e.g., understanding 2’s complement), but text in the IR window was difficult to read. Some suggested that using a circle instead of a line segment would be better to show the wrapping of signed integers (i.e., INT_MAX + 1 = INT_MIN). However, the most important suggestion was making IR and EE a mobile app or a web app. A student in CC suggested to offer the same for floating-point.

4.7.2 EE Exit Survey. The questions in EE’s exit survey are similar to those of IR. They still have the four categories. The questions are: (a) EE helped me understand when an integer coercion will take place, (b) with EE the student understood the resulting type from an integer coercion, (c) with EE the student understood integer coercion better, (d) with EE the student was able to identify concepts that were not understood, and (e) EE enhanced the course. Responses from both classes were similar, again. Answers to questions were 3/4/4 except for questions (a) and (d). The CC class answered question (a) with 4/4/4, which is higher than that of the OOP class 3/4/4. The OOP class responded to question (d) with 3/3/4, because most students chose Neutral but the histogram was still skewed left. This means in general the answers were still on the positive side. Thus, both classes rated EE highly.

The next seven questions asked students to evaluate EE’s components: (i) the EE window was easy to read, (ii) the EE window was intuitive, (iii) it was easy to enter expression, (iv) it was easy to enter values for variables, (v) it was easy to edit an expression, (vi) it was helpful to be able to save an expression, and (vii) it was easy to follow the steps of EE and the coercion that are performed. The responses from the OOP and CC were similar. The CC class consistently responded with 3/4/4 except for question (iv) (i.e., easy to enter values) with 3.75/4/4, a slightly higher first quartile. However, the OOP class offered lower median values for questions (i), (ii), (iii) and (vi). The rating of questions (i), (ii) and (vi) were 3/3/4 while the rating for question (vi) was 3/3/5. The histograms of the responses of these questions were still skewed left, which means students in both classes rated EE in a positive way.

The last three questions in EE’s exit survey were the same as those in IR. We found that students in OOP used EE more often than students in CC did. Students in the OOP class used EE 15-30 minutes while students in the CC class only used 5-15 minutes with the same reason as in the IR survey. The Spearman ρ's also indicated that the correlations between the total time a student spent on EE and other questions were at best moderate around 0.5.

Students in both classes reported known bugs of EE and most of them indicated that the current version of EE was good enough and simple/easy to use. In fact, the comments for EE were not very different from those of IR. The most important comments were still (1) students preferred native code app rather than Java, (2) generating expression trees (Figure 6) would be more useful, and (3) mobile app and/or web version would be more useful.

5 CONCLUSIONS AND FUTURE WORK

This paper presented our findings in using the IR and EE materials at NTUT in a lower-level class and an upper-level class. Students in the upper-level class rated the materials higher than the students in the lower-level class did. Results showed that our materials were effective and successful. Some existing bugs made students to rate the visualization tools lower; however, the overall ratings were still positive. These known bugs were discussed or presented on the class slides and students may not be aware of them.

The most interesting suggestions from the students were developing a web-based app and/or mobile app, and a similar system for the floating-point arithmetic. In fact, we developed a simplified visualization tool for floating-point arithmetic [3] that is capable of showing the serious issues from improper use of floating-point arithmetic such as rounding, truncation, cancellation, and the failure of associative law and distributive law. However, we did not mention this visualization tool in the OOP nor the CC classes. This visualization tool can be developed further to make it more extensive and more helpful. This will be our future work.

Web-based versions of IR and EE are available via the following links: irweb.cs.wmich.edu and eevis.cs.wmich.edu.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under grants DGE-1523017, DGE-1522883 and DUE-1245310.
REFERENCES