Using Algorithm Visualization to Improve Students' Understanding of Parameter Passing Methods

Jessica M. Gowey
Computer Science Department
University of Wisconsin Oshkosh
goweyj98@uwosh.edu

Orjola Kajo
Computer Science Department
University of Wisconsin Oshkosh
kajoo95@uwosh.edu

Abstract

Some of the most difficult concepts for students to master are the differences between parameter-passing methods, such as by-reference, copy-restore, by-name, and macro processing.

Algorithm visualization (*hereafter referred to as AV*) uses computer graphics to depict the actions of an algorithm. In this project, our work in AV is used as a tool to help students understand these parameter-passing methods.

We have developed a program that creates a random instance of a parameter-passing problem and then solves it by using either by-reference and copy-restore or by-name and macro processing methods. The solution to the problem is hidden with the purpose of being unveiled by the student through a sequence of snapshots supplemented with interactive questions that force the student to predict what will happen next.

Challenges of Parameter Passing Methods Addressed In This Project

There are two ways that a *function* (also referred to as *subprogram, procedure, routine, etc*) can gain access to the data that it is to process: through direct access to non-local variables (declared elsewhere but visible in the subprogram) or through parameter passing. An advantage to parameter passing is its flexibility compared to direct access to non-local variables. Moreover, extensive access to non-local variables can cause reduced reliability. Variables that are visible to the function where access is desired often end up also being visible where access to them is not needed [5]. Thus, parameter passing is an important topic in computer science and it is specifically studied in a junior-senior level course on programming languages. Some of the most difficult concepts for students to master in such a course involve the subtle differences between the various parameter-passing methods. Typically the following parameter-passing methods are studied – by-reference, copy-restore, by-name, and macro processing [7].

We have focused our attention on these four in-out mode parameter passing methods. More specifically we have emphasized comparing and contrasting these methods in groups of two: by-reference versus copy-restore and by-name versus macro processing. The rationale for this approach is based on the similarities between each of the parameter passing methods within the same group. Despite their similarities, these methods take different approaches to solve a problem and thus produce different results under the same conditions.

The goal of this project is to create a thorough understanding of the similarities and differences between the four parameter passing methods mentioned in the preceding paragraph through visualized, interactive, problem solving. Ultimately we will statistically test the following hypothesis - ``Students who use AV learn parameter-passing significantly better than students who use no visualization".

By-Reference and Copy-Restore

The parameters in a function call are referred to as *actual parameters*, or simply, *actuals*. The *formal parameters*, or *formals,* are placeholders for actuals.

In the by-reference parameter passing method, a formal parameter denotes the location of an actual parameter [6]. Thus, assignments to the formal parameter within the body of the function directly affect the value of the actual parameter. One way to look at this is using the formal parameter as another way to refer to the actual parameter.

In copy-restore, also known as copy-in/copy-out, actual parameters with locations are treated as follows:

1. *Copy-in phase*. Both the values and the locations of the actual parameters are computed. The values are assigned to the corresponding formals and the locations are saved for the copy-out phase.

2. *Copy-out phase*. After the procedure body is executed, the final values of the formals are copied back out to the locations computed in the copy-in phase [6].

These two methods are similar because they both have access to the locations of the actual parameters and after the execution of the function body they are expected to have modified their values. However, the difference between the two methods stands in the type of access that they have to the locations of the formals. In pass by-reference, the function has direct access to the locations of the actuals at any particular instant throughout the execution of the function body. In copy-restore, the function has a one-time access to copy the value and the address of the actuals at the beginning of the function execution and a one-time access to modify their values before the function returns.

To illustrate by-reference and copy-restore methods, we will solve the following problem in a pseudo-code resembling C++, by using each of these two methods. This is an example of a case where each method produces a different result.

Example 1.1

```
void foo(int x, int y){
        x = 7;
        y = y + 2;     }

int main(){
        int j = 3;
        foo(j, j);
        print(j);     }
```

Under call *by-reference*:

- x refers to j
- y refers to j
- i is assigned the value of 7 through x
- i is assigned the value of 7+2 = 9 through y
- after the function returns, 9 is printed as the value of j

Under call by *copy-restore*:

- x is assigned the value of j = 3
- y is assigned the value of j = 3
- the address location of j is stored as the copy-out address for both x and y
- x is assigned the value of 7
- y is assigned the value of 3+2 = 5
- the value of x = 7 is copied out to j
- the value of y = 5 is copied out to j, thus overwriting 7
- after the function returns, 5 is printed as the value of j

By Name and Macro Processing

When a formal parameter is passed by name, an actual parameter access function is passed to the called routine in place of the actual parameter. This access function provides a method for repeated access to either the value of an actual parameter or its address. In the body of the called function any reference to a name parameter is replaced by a call to the corresponding access function [1].

Macro processing uses textual substitution for parameter passing by doing the following:

1. Actual parameters are textually substituted for the formals.
2. The resulting function body is textually substituted for the call [5].

These two parameter passing methods are very similar since both use a form of textual substitution as an approach to evaluate a function. The difference between the two is the level of integration of the textual substitution during the function execution. In macro processing plain text is used to substitute the formal parameters with the names of the actuals and the function call with the function body. In the by-name evaluation the call to a function that calculates the address of an actual is placed wherever the corresponding formal parameter appears in the body of the function.

The following example in a pseudo-code resembling C++, illustrates how macro processing and by-name can produce different results when they are used to evaluate the same function.

Example 1.2

```
void foo(int p){
    int n = 2;
    p = n;          }

int main()   {
    int n = 3;
    foo(n);
    print(n); }
```

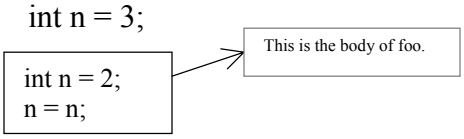
Under call *by-name*:

- memory location for n is declared in main
- function that calculates the address of n and returns a pointer to it is passed as a parameter to foo in the place of n
- memory location for n is declared in foo
- function that calculates the address of n in main is substituted for p and it is assigned the value of n=2 in foo
- after the function returns, 2 is printed as the value of n

Under *macro processing*:

- plain textual will produce a program fragment as follows:

```
int main()   {
     int n = 3;
     int n = 2;
     n = n;
     print(n); }
```



- the assignment n = n is bound to the latest declaration of n and does not affect the first declaration of n in main
- after the function returns, 3 is printed as the value of n

Development Stages of the Project

We have developed a set of visualization-based tools to help students master the by-reference, copy-restore, by-name and macro processing parameter passing methods. The project consists of three development stages.

Randomly Generating Problems Involving Parameter Passing Methods

During the first stage of this project we have developed a program that dynamically generates small parameter passing problems in a pseudo-code resembling C++. We have designed a template for this purpose. There are two different types of parameters in a template, *invariant* and *variable*. Invariant parameters are those that determine how problems are generated and may also be used to control the difficulty level of the problem [4]. The invariant parameters of our template are given in the following syntactic structure:

Syntactic structure:

```
<program> ::= <globals> <main> <function>
                | <main> <function>
```

Explanations of non-terminals:

```
<globals> ::= <variable_list>
<main> ::= <variable_list> <function_call>
<function> ::= <variable_list> <expression_list>
<variable_list> ::= <variable> | <variable> <variable_list>
<expression_list> ::= <expression> | <expression> <expression_list>
<expression> ::=   <variable> = <integer>
                 | <variable> = <variable>
                 | <variable> = <variable> + <integer>
                 | <variable> = <variable> + <variable>
<variable> ::= int <name>
```

Variable parameters of a template determine the current instance of the problem being generated. Their presence in the template and the randomization of their values assure the generation of a different problem each time [4]. Such parameters in our template are:

- the number of global variables
- the number of variables present in main
- the number of parameters to the function
- the number of local variables in the function
- the number and type of expressions in the function body

These numbers are randomly generated each time an instance of a problem is created. The type of each expression is randomly chosen out of the four possibilities presented in the syntactic structure. Also, the names of the variables and parameters are randomly chosen from a predetermined list. The presence of the global variables or arrays is used to control the difficulty level of the problem and is presented as an option to the student user.

As mentioned at the beginning of this paper, the goal of this project is to teach the differences between parameter passing methods classified in two groups, by-reference and copy-restore, and by-name and macro processing. To serve this purpose, the problems that are generated by this program are of two types - problems to be solved by using the methods in the first group and problems to be solved by using the methods in the second group. More specifically we want to generate problems that will produce different results when solved within the same group, as well as problems that produce similar results. Thus, the user first chooses the type of the parameter passing method, then the difficulty level. After these parameters are determined a problem is generated and displayed.

This program is able to solve each problem as soon as it is generated by using the two methods specified by the user. The algorithms used to solve the problems are the ones described in the first section of this paper. The solutions to the problems are stored as a sequence of steps and are temporarily hidden with the purpose of being unveiled by the student during the visualization of the algorithm.

This is an instance of a problem generated by our program that is to be solved with the by-reference and copy-restore methods using both globals and arrays.

Example 2.1

```
0    int j = 7;
1    int l = 3;
2
3    int main() {
4      int f = 9;
5      int p[4] = {3, 4, 2, 1};
6      int m = 5;
7      int b = 6;
8      int n = j + j;
```

```
9
10      By_Copy_Reference(b, b, f, p[3], m, n);
11    }
12
13
14    void By_Copy_Reference(int b, int c, int f, intl, int h, int g){
15      int h[4] = {8, 6, 9, 9};
16      int p = 1;
17
18      j = g + f;
19      b = j + j;
20      c = g + b;
21      l = c + j;
22    }
```

Algorithm Visualization

In the second stage we have developed the visualization tools that make possible the graphical presentation of the problem and the step-by-step solution as a sequence of snapshots. Snapshots are supplemented with interactive questions that force the student to predict what will happen next in the sequence that has been momentarily "frozen".

The generation of the problems, the solution algorithms, and the generation of the textual snapshots are implemented as a Java application (version 1.4.2 or later). Graphics are rendered in GAIGS (Generalized Algorithm Illustration via Graphical Software), an *algorithm visualization* scripting language that captures and renders *snapshots* of the state of an algorithm at critical points (*interesting events*) in its execution [2]. The GAIGS graphics are delivered by JHAVÉ, an algorithm visualization platform that supports several scripting languages including GAIGS and HTML. The text of the generated problem and the interactive questions are displayed in an HTML window supported by JHAVÉ. This entire system, including our parameter-passing visualizations, can be downloaded as a .jar file from the site http://csf11.acs.uwosh.edu/crew_project. In order to run the application, an internet connection is required to connect to the visualization server.

Description of the Graphics

At critical points of the execution, the program will output information about the state of all the variables created up to that point. This information is arranged in the form of textual snapshots in a file and is then downloaded by the JHAVÉ client, within which the GAIGS rendering engine renders a data structure picture for each event. The critical points we identify during program execution are variable declarations, the function call, assignment statements, and the function return. Every line of code that generates a snapshot is highlighted in the HTML window where the problem text appears at the time

the corresponding snapshot is displayed in the graphics window. At randomly spaced intervals, the student is asked a question about what will happen next in the program's execution.

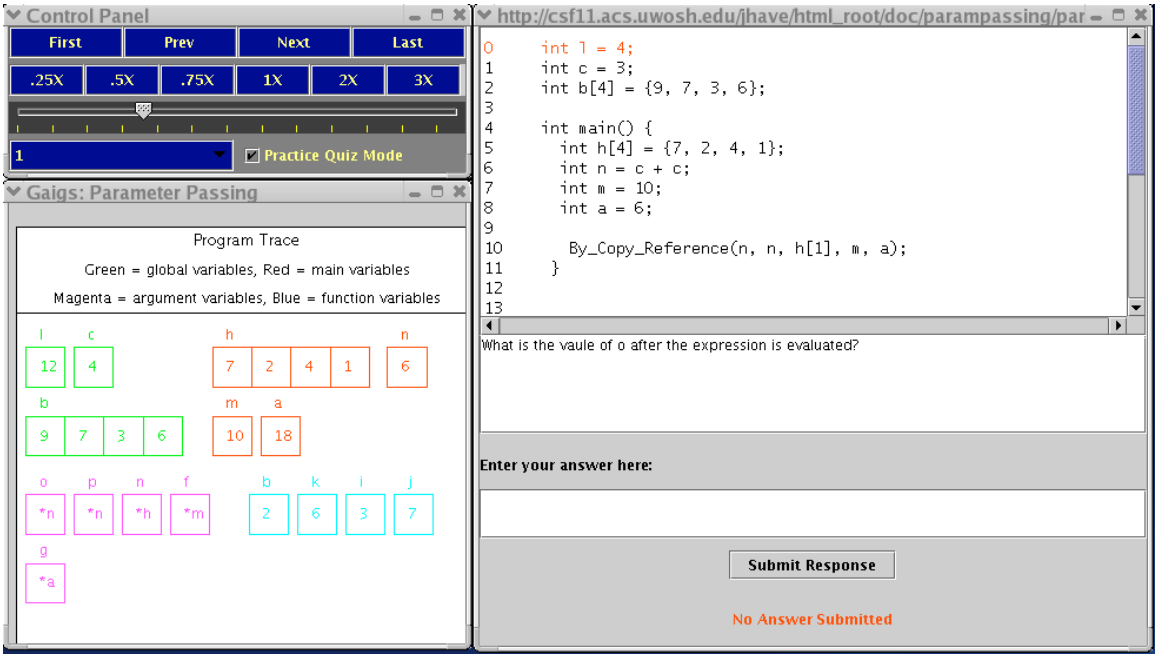


Figure 3.1 This is a screenshot of the program that is generating snapshots for a by-reference parameter passing problem.

Integration of the Engagement Taxonomy in Our Project

In the report of the 2002 ITiCSE working group on "Improving the Educational Impact of Algorithm Visualization" a taxonomy of the learner's engagement with visualization technology is defined. This taxonomy describes six different forms of engagement:

1. *No viewing*: No visualization technology is used at all.
2. *Viewing*: The learner passively views the visualization of an algorithm.
3. *Responding*: Answering questions related to the visualization is the form of engagement.
4. *Changing*: The learner's engagement involves modifying the visualization.
5. *Constructing*: The learner constructs his/her own visualization of the problem.
6. *Presenting*: A presentation of the visualization of a problem to an audience and discussion of the problem is required in this category [3].

Our project integrates the learner in both viewing and responding. As the graphics are displayed, questions appear in the HTML window and the program is momentarily "frozen" while waiting for the user's response. The types of questions asked are "What

will the value of variable be in the next snapshot?" and "What will the value of variable be after the function returns?". After an answer is submitted, the feedback of the program is in the form of a "Correct/Incorrect" answer.

Measuring the Effectiveness of the Tools

The third stage of this project involves measuring the effectiveness of the tools that we have developed. This stage will be completed by May 2004. We are going to base our statistical study upon the experimental framework for studying the effectiveness of AV that was developed by the 2002 ITiCSE working group on "Improving the Educational Impact of Algorithm Visualization" [3]. According to the metrics for determining the effectiveness of visualization developed by this group, we are going to measure the application level (Level 3) [3] of the learners' understanding of the four parameter passing methods – by-reference, copy-restore, by-name, and macro processing. In this level of understanding the learner can apply the learned material in specifically described new situations [3].

We will statistically test the following hypothesis – "Students who use AV learn parameter-passing significantly better than students who use no visualization". The algorithms that we have chosen are described in detail in the first section of this paper and the visualization tool that supports the form of engagement is described above. Since we have satisfied the assumptions for the framework, we can proceed with the following steps:

- *Participants*: The students from the spring 2003 version of the programming languages course will form the control group and those from the spring 2004 will form the test (AV-aided) group.
- *Materials Tasks and Procedure*: The control group has already studied the parameter passing methods and taken a test on the application of these methods. This test consist of a set of three parameter passing problems that the students are asked to solve. We already have the results of this test. After having time to study parameter-passing methods, the AV-aided group will be given a similar test. This test consists of a problem that will produce different output depending on the parameter-passing mechanism used in its function calls. Students are asked to predict the outcome of the program under each of the four parameter-passing mechanisms.
- *Evaluation Instruments*: This part of the statistical study will be completed after the AV-aided group has been tested.

Conclusion

In order to increase the learner's engagement with visualization, this project could be extended by allowing the learner to make changes to the code of the problem that is generated with the purpose of completing a request from the program. For example, one request could ask the learner to modify the code, without changing the parameter passing method, in such a way that the value of a variable is affected differently than it was previously. Questions of this nature will force the user to interact more actively with the visualizations. A future study on improving students' understanding of parameter passing

methods could utilize the existent program and an extension on it that increases the level of engagement of the user such as the one described above.

References

[1] Bergin, J., and Greenfield, *S. Teaching Parameter Passing by Example Using Thunks in C and C++.*

[2] Naps, T., and Swander, B. (1994). An Object-oriented Approach to Algorithm Visualization. *Proceedings of the SIGCSE Session, ACM Meetings*.

[3] Naps, T., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hunhausen, C., et al. (2003). Exploring the role of visualization and engagement in computer science education – Report of the working group on improving the educational impact of algorithm visualization at the 7th annual SIGCSE/SIGCUE conference on innovation and technology in computer science education. *ACM SIGCSE Bulletin Inroads.*

[4] Kumar, A. Dynamically Generating Problems on Static Scope.

[5] Robert, W. S. (2002). Subprograms, *Concepts of Programming Languages* (pp.352-353). Addison–Wesley.

[6] Sethi, R. (1996), Procedure Activations, Programming Languages: Concepts and Constructs (pp.155-165). Addison-Wesley.

[7] Weber, A. (2003). A practical Introduction, *Modern Programming Languages*. Franklin, Beedle, and Associates.

Acknowledgements

We wish to gratefully acknowledge the support that CREW (Collaborative Research Experiences for Women) has given us on this project.

We particularly thank Professor Thomas Naps at the University of Wisconsin Oshkosh for his invaluable guidance.