AC 2012-5297: WORK-IN-PROGRESS: PROGRAMMING CONCEPT VISUALIZATION USING FLASH ANIMATIONS

Dr. Lin Li, Prairie View A&M University

Lin Li is an Assistant Professor of the Computer Science Department at Prairie View A&M University. He received his Ph.D. in computer science from the University of Nebraska, Lincoln, in 2004. Before that, he received his B.S. and M.E. from Beijing Institute of Technology and Chinese Academy of Sciences in 1996 and 1999, respectively. Currently, his research interests are in computer educational technology, green home, and network communications.

Jaime Israel Juarez, Prairie View A&M University

Jaime I. Juarez is a Software Engineer who graduated from Prairie View A&M University with a bachelor's and a master's degree in computer science. His expertise is in web design and interactive computer education technologies. He is currently working on an intelligent testing/tutoring system development. Juarez is interested in teaching computer science at the collegiate level.

Prof. Yunggao Yang, Prairie View A&M University

Yunggao Yang is a tenured Associate Professor and the Interim Department Head in the Department of Computer Science at Prairie View A&M University (PVAMU) in Texas. He joined the CS Department at PVAMU as an Assistant Professor in Aug. 2002 immediately upon receiving a Ph.D. degree in information technology and computer science from George Mason University (Fairfax, Va.). Yang's research interests include computer graphics, scientific visualization, distributed virtual reality learning/training environment, and computer networks. Yang can be reached through email at yoyang@pvamu.edu.

Work in Progress: Programming Concept Visualization using Flash Animations

Abstract

Critical programming concepts such as recursion and inheritance are often difficult to grasp for beginners. As a result, students perform poorly in introductory programming courses and the attrition rate has become a severe nationwide problem¹. Surveys reveal that many students are unable to comprehend “what is happening to their program in the memory” and “each instruction is executed in the state created by the previous instructions”^{2,3}. To help students understand the essence of programming, revamping the teaching materials to effectively deliver the knowledge becomes essential.

This paper presents a work that is currently conducted at the Computer Science Department of ABC University, which, through pilot testing, showed to be efficient in increasing student engagement and supporting teachers’ instructional needs. The key strategy is to use new courseware to enhance student learning. Developed by applying Flash and animation technologies, the new courseware and programming learning modules can: (1) make programming interesting while retaining the underlying contents; (2) visualize programming logic and memory change; make abstract and intricate concepts “visible” and “touchable”, and thereby, easy to understand; and (3) foster self-study, stimulate critical thinking, and improve students’ learning effectiveness outside class meetings. The goal is to ensure that computer majoring students, especially freshman and sophomores, can develop correct understanding of programming concepts. To better organize the courseware, a training system is implemented to manage the learning modules and support online access. A summary of the programming topics, courses impacted, and samples of the designs of the learning modules are discussed.

Background

Programming serves as the foundation of computer system design and software engineering. No matter how the curriculum is updated, programming has always been attached the greatest importance in computer science education⁴. However, the educational outcomes are disappointing. Studies indicated that there were severe deficiencies in the performance of students who had passed one or a few programming courses in computer programs^{5,6}. The problems originated from misconceptions on early studies. Poor understanding of basic programming concepts, procedures and processes leads to a poor basis for advanced courses. Another consequence is the high dropout rate on programming courses. Many students have to retake the courses several times before passing them, which leads to a huge waste of time and resources both for students and for institutions.

Students living in the digital age are visual and active learners. They prefer knowledge imparted through multiple ways of information representation such as interactive text, images, sound and video. For many of them, the primary trouble in programming concept comprehension is that when a program executes, it is essentially a “black box”. Most of the programming environments can only display limited running environment information, and students cannot see what is happening in the computing procedure. The only clues to logic errors come from output of the

program. It is difficult for novice programmers to trace the problems back to the sources. Therefore, how to effectively engage students and how to easily deliver the programming concepts become the key to address the issues.

In the past, numerous approaches have been proposed to improve programming teaching and various tools have been developed to augment programming environments^{7, 8, 9, 10, 11}. Among these efforts, visualization proved to be effective in explaining critical concepts¹². Our teaching experience also confirmed this. Meanwhile, it brings to our attention that programming environment augmentation tools only provide general enhanced platforms which can still be fussy to inexperienced students. Not much work has been done on using memory visualization to explain specific programming concepts. Hence, we explore using Flash animations to develop new course materials for programming logic explanation, based on which renovated teaching schemes are investigated to enhance student learning.

Goal and Objectives

The overarching goal of this work is to create engaging programming learning modules using Flash and animation technologies, through which we can improve students' performance in programming courses and make them better prepared for advanced computing courses.

The specific objectives of the work include:

1. Develop Flash animation learning modules to enrich the teaching resources in programming courses; make programming attractive while still retaining the underlying contents
2. Improve the delivery of laboratories and lectures, make abstract and non-intuitive programming concepts "visible", "touchable", and thereby, easy to understand
3. Increase students' passing rate in programming courses
4. Foster students' interest and promote active learning inside and outside class meetings.

To achieve the goal and objectives, we designed and developed a learning module management system to manage the learning modules and schedule teaching and training activities. The project consists of state-of-the-art technologies that simplify the process of complicated concepts delivery and facilitate teaching innovation.

Flash Animations and Learning Modules

We adopted Adobe Flash professional to develop the animations for each learning module. Different from simple animations made by PowerPoint, Flash animations integrate much more sophisticated programming elements and are good at depicting complicated concepts. Plus, Adobe Flash is the de facto standard for online animations which can be easily embedded into web based systems. All learning modules and training questions are created based on the real examples from our lectures in programming courses. Generally, the modules can be categorized into two types. The first type is for tutoring and lecturing activities, and the second type is for practicing and training. Accordingly, each module consists of two components: (a) concepts explanation (type one) or problem description (type two); (b) interactive animation and instant comments/feedback. The first component is to review or present specific programming topics. The second one is to illustrate the concepts step by step. Audio can be integrated to emulate tutor

explanation. Students can interact with the animations to observe program running logic and dynamic memory change as if they were taking a tour inside the electronic computing procedure. Based on course levels, we categorized the modules for Computer Science I, Computer Science II, Data Structures (DS), and Programming Languages (PL). A list of the topics and associated courses is shown in Table 1. Many modules can be used in multiple courses.

Table 1. List of learning modules and associated courses

Topics	Targeted Courses	Topics	Targeted Courses
Control & Loop	CS I	Variable, Scope, & Binding	CS I, CS II & PL
Function	CS I & II	Reference & Pointer	CS II, DS & PL
Recursion	CS II, DS & PL	Parameter Mode	CS II & PL
Arrays & String	CS I, CS II & DS	Inheritance &	CS II & PL
Linked List	CS II & DS	ADT, Stack, & Queue	CS II, DS & PL
Class & Object	CS II & PL	Data Type & Storage	CS I, CS II, DS &

Samples of Learning Modules

Four learning modules and the scenarios on how we use them to explain recursion, linked list, parameter passing, and reference binding schemes are described respectively as follows:

Scenario 1: recursion and variables life scope. Recursive functions are the key for many algorithms, but the variable allocation/deallocation and stack memory change in recursion are not easy to trace. Thus, we clarify them with a C++ exercise in which students are asked to analyze the program logic and determine the output result, as depicted in Figure 1.

```

int A=5;
int Fun(int &B) {
    if (A>B) {
        A--; B++;
        return A + Fun(B);
    }
    else
        return A;
}

int main() {
    int B, C; B=2;
    int &D=B;
    C = Fun(B);
    cout << A << B << C << D << endl;
    return 0;
}

```

```

#include <iostream>
using namespace std;

int A = 5;
int Fun(int &B);

int main()
{
    int B, C; B=2;
    int D = B;
    C = Fun(B);
    cout<<A<<B<<C<<D<<endl;
    return 0;
}

```

D references B, no new allocation in memory, D acts as an alias of B.
To find the value of C we make a call to the Fun() function, 1st Call

STEP 1

main()

B 2

C

stack

A = 5

static

Figure 1. Recursive program and animated learning tutorial

To calculate the values of variable **A**, **B**, **C**, and **D** in main function, first we need to investigate their life scopes: **A** is not defined within the function block, thus it refers to the global variable; **B** and **C** are locally declared and their life scope is from declaration to the end of the function block; **D** is a reference of **B**, so their values are always same. Second, we need to check the variables defined in the recursion function: only a reference **B** is defined in the parameter list; **A** in Fun function refers to the global variable; **B** is local, but its value changes with the all variables associated with this reference. Finally, each recursive call of Fun will create a copy of the function in memory together with its own local variables/references; tracing down to the calls and returned values will bring the answer out of the mist.

Figure 2. Stack and variable life scope visualization in recursion

To help students understand the knowledge points, we designed a Flash animation to visualize the function call and return procedure. Being a *tutoring type* learning material, the module is divided into eight steps. Each step shows the change of current memory and highlights variable values along with the statement execution. Colored texts and drawings are used to emphasize what is happening in memory. Sub-steps are designed for key concepts to zoom in the details. The animation allows students to interact with the play. By clicking different step buttons, students can navigate through the module, repetitively watch the animation, and read the tutoring notes. The snapshots of the module are depicted in Figure 1 and 2.

Topic: Which of following operations will insert a node after the second of the double linked list?

Choices:

- A. temp→right→left = add;
add→right = temp→right;
add→left = temp;
temp→right = add;
- B. temp→right = add;
temp→right→left = add;
add→right = temp→right;
add→left = temp;
- C. add→right = temp→right;
add→left = temp;
temp→right = add→right;
temp→left = add;
- D. add→right = temp→right;
add→left = temp→left;
temp→right = add→right;
temp→left = add;

```

node add;
add = new node;
add->value = 100;
add->left = NULL;
add->right = NULL;

```

```

// searching for the second node
temp = head;
for (int i=0; i<1; i++)
  temp=temp->right;

```

```

// insert the node

```

- 1.
- 2.
- 3.
- 4.

Sample Student Solution: (A), two steps of the relevant animation are shown to the right

Figure 3. An example of animated training of double linked list

Scenario 2: A partial completed program is provided for double linked list node insertion. Given four candidate code blocks, please find the correct one that can insert the node as instructed. Technically, the problem can be solved by analyzing the relationship among the current node, preceding node, and the following node. Due to the special property of linked lists, the adjustment order of the pointers and variable values must be carefully considered.

Hence, we designed a *practicing type* learning module in which four different animations are created for the four choices. Each animation simulates the step-by-step execution of the four statements in that code block. Once a student makes a choice, the related animation for that code block will be played. The student can then observe how the memory and variables are modified along with the execution of the code. The student will also be prompted whether the selection is correct or not. Similarly, by clicking different step numbers, the animation allows the student to repetitively watch how the program will affect the node connection in the linked list. Different from *tutoring type* modules, practicing learning modules do not display tutoring notes. Some snapshots of the module are depicted in Figure 3.

Figure 4. Flash animation for parameter passing strategies

Scenario 3: Given a program, what is its output based on different parameter passing schemes? Parameter passing schemes determine the relationship between variables. In this *tutoring* case, we first review the characteristics of four typical passing strategies, namely, *pass-by-value*, *pass by reference*, *pass by value-result*, and *pass by name*. Then, for each strategy, we use an individual animation to show the difference on which variables are local, which is global, how the following calculation is affected, etc. Students can navigate through the steps to watch computing details and read highlighted tutoring notes (depicted in Figure 4). It should be noted that a salient advantage of our learning modules over normal programming augmentation tools is its independence of the language grammar. To explain concepts like this one, Instructors do not need to teach students a new language before moving to the knowledge point. Pseudo code will be enough for showing the programming logic. This is very useful for comprehensive programming language analysis course such as Programming Languages Pragmatics. To consolidate student learning, we also design practicing cases for students to exercise and double check the correctness of their understanding.

Figure 5. Flash animation for variable life scope and binding schemes

Scenario 4: Given a pseudo program, how different will the output be if the variables follow static binding or dynamic binding? Running environment reference is another vital concept. Different referencing schemes compare how variable life time and values are affected in various languages. It can deepen students' understanding in programming logic. Again, we created Flash animations to demonstrate how the variable names and memory values are associated through the program running flow (depicted in Figure 5).

Compared to diagram drawing and traditional program analysis, our Flash based learning modules are interactive and self explainable. Each module functions like a short tutorial/training targeted at specific programming concepts. Through these modules, students can study programming as if they had a tutor by their side. They can verify whether their thinking is right or not and observe the correct way of programming. Therefore, we can improve students' self-study efficiency outside classroom meetings. More importantly, the learning modules, once developed, can be easily adapted and reused for many terms.

Learning Module Management System

The authors at the Computer Science Department of ABC University initiated this work in spring 2011 to renovate the programming courses. The main purpose is to develop effective course materials and an entertaining learning environment. To better organize the animations and learning modules, a training management system is implemented to manage the questions and training activities. The system is web-based and developed with HTML, PHP, MySQL, and Adobe Flash. With this system, students can access the learning materials through the Internet. They can study by themselves at anytime and anywhere. Moreover, students do not need to install any software as long as their browsers are updated with the latest flash plug-in.

This Learning Module Management System (LM²S) was conceived in order to manage the modules and provide students with a training platform. It accommodates functions for three kinds of users: *instructor*, *student*, and *administrator*. The instructor and administrator interfaces are modeled after Content Management Systems (CMS) with links and forms used to complete the desired tasks: (a) An instructor can add, edit, and delete questions and learning modules; (b) An administrator can add, edit, and delete users, courses, and question types in the system; (c) The student interface is designed like an online testing system where different programming

concepts can be tried by students. Students can interact with the animation as if they were discussing a problem with a professor. Students can see their scores once a training module is completed. Other functionalities like class stats generation are also being integrated into the system. Some snapshots of the system and training activity are depicted in Figure 6 and 7.

Figure 6. Snapshots of a student taking a training of two interactive questions

Figure 7. Snapshots of an instructor editing learning modules and course information

Ongoing Project at PVAMU and Implementation

Our system and learning modules have unique features that other courseware does not have. Through pilot testing of several learning modules among students, we have seen very positive results. Many students felt excited about these materials and they thought the modules could help them learn programming concepts accurately. It truly encouraged us to put more efforts along this direction. We are now following three steps to achieve the goals and objectives:

Phase 1: Investigate critical programming concepts and student problems in understanding, develop Flash animations and learning modules for programming courses like CS I, CS II, Data Structures, and Programming Languages; test the courseware in targeted courses. Based on the student' feedback, we will refine the modules and animations. Currently, we have designed more than 20 study cases and animations and **we are adding more examples to our database.**

Phase 2: Implement the proposed learning module management system, design the web interface, develop the functions for different users, and test the system in targeted courses. To fully utilize the modules and engage students in class activities, we also plan to revamp the teaching plan. More interactive exercises and quizzes will be defined in formats that can stimulate students' interests and assist collaboration (e.g. Jeopardy Game). Furthermore, we are incorporating advanced testing functions to the system. We anticipate that in the long run, the system can serve as a truly comprehensive online training/testing platform.

Phase 3: System expansion, evaluation, and outreach. At this stage, we will broadly adopt more computing topics and develop related learning and training modules to reinforce computer science education. Discovery and results will be presented at the college and university levels.

We plan to conduct systematic formative and summative evaluation during the proceedings of the work. We will collect and compare course assessment data before and after applying the educational innovation to see the differences in teaching effectiveness. We will also administer student survey for subjective feedback on each targeted course to see the improvement in student learning and class engagement.

Conclusions

To summarize this ongoing project, we are developing new programming learning modules using Flash animation technologies to address the learning problems students encountered in programming courses. With these learning modules, abstract and difficult programming concepts can be well explained using visualizations. It makes the teaching of programming concepts attractive and easy to understand. A phased implementation plan is followed to ensure the attainment of the goals. Upon the completion of the project, we expect to have the following outcomes: (1) the developed courseware is applied to a sequence of programming courses; (2) Students are more engaged in classroom interactivities and have better understanding in breadth and depth of the programming concepts; (3) Students' motivation and passing rate in programming courses are increased.

Acknowledgment

This work is supported in part by National Science Foundation HBCU-UP award #1036325. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Bibliography

1. Beaubouef, T., and Mason, J., 2005, "Why the High Attrition Rate for Computer Science Students: Some Thoughts and Observations", *ACM SIGCSE Bulletin*, vol. 37, no. 2, pp.103-106
2. Lahtinen, E., AlaMutk, K., and Järvinen, H., 2005, "A Study of the Difficulties of Novice Programmers," *ACM SIGCSE Bulletin*, vol. 37, Issue 3.
3. Miline, I., and Rowe, G., 2002, "Difficulties in Learning and Teaching Programming-Views of Students and Tutors", *Education and Information Technologies* 7:1, pp. 55–66.

4. ABET Accreditation Commission, 2009, "Effective for Evaluating During the 2009-2010 Accreditation Cycle", *Criteria for Accrediting Computing Programs*.
5. Lister, R., et al., 2004, "A Multi-National Study of Reading and Tracing Skills in Novice Programmers", *ACM SIGCSE Bulletin*, vol. 36, no. 4, pp. 119-150
6. Tenenberg, J., et al., 2005, "Students Designing Software: a Multi-National, Multi-Institutional study", *Informatics in Education*, vol. 4, no.1, pp.143-162.
7. Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlinand M., Paterson, J., 2007, "A Survey of Literature on the Teaching of Introductory Programming", *ACM SIGCSE Bulletin*, vol. 39, issue 4.
8. Etheredge, J., 2004, "CMeRun: Program Logic Debugging Courseware for CS1/CS2 students", *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, pp. 22-25.
9. Campbell, A. E., Catto, G. L., and Hansen, E. E., 2003, "Language Independent Interactive Data Visualization", *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, pp. 215-219.
10. Stasko, J. T., 1990, "A Framework and System for Algorithm Animation", *IEEE Computer*, vol. 23, no. 9, pp. 27-39.
11. Zeller, A., and Lütkehaus, D., 1996, "DDD—a Free Graphical Front-End for UNIX Debuggers", *ACM SIGPLAN Notices*, vol. 31, no. 1, pp. 22-27.
12. Collins, C. and Tabrizi, M. H. N, 2007, "Integrating Visual Modeling throughout the Computer Science Curriculum", *International Journal of Social Sciences*, vol. 1, no. 4, pp. 264-273.