Ogata, H. et al. (Eds.) (2015). Proceedings of the 23rd International Conference on Computers in Education. China: Asia-Pacific Society for Computers in Education

Active Debugger: ITS to Teach C++ Comprehension Skills

Ryan James MARKER, Amali WEERASINGHE
School of Computer Science, The University of Adelaide, Australia
*amali.weerasinghe@adelaide.edu.au

Abstract: Active Debugger is an ITS that is deigned to teach C++ comprehension skills based on constraint-based methodology. Students are asked to explain the changes that take place in the memory for a given C++ program. Students have the freedom to ask for feedback at any point in the program. When the feedback is requested, the correct solution is dynamically generated until the line of code that a student has attempted. This solution is used to evaluate the student attempt together with the constraint-base. This generated-solution can also be viewed as a simulation to understand the changes that occur in memory after each line of code. An initial study was conducted to evaluate the effectiveness of the system in learning C++ comprehension. This study involved authentic students in a computer science undergraduate course. Results revealed a positive trend towards learning. Further evaluations needs to be carried out to evaluate the effectiveness of the system.

Keywords: intelligent tutoring system, programming language comprehension

1. Introduction

Programming is a core skill expected from every computer science graduate. Thus programming is taught in every CS undergraduate programme. It is generally taught at the first year courses and most first year courses consist of hundreds of students. Despite using various teaching techniques, novices still find it very difficult to develop programs from scratch. One of the reasons for this difficulty is the open-ended nature of the task (Mitrovic & Weerasinghe, 2009). A problem statement specifies the problem to be solved and the outcome to be achieved, but there is no well-defined procedure to move from the problem statement to the final outcome. Another reason is the necessity to deal with dual tasks – design the solution in a way that a computer can solve the problem and communicate the solution in a programing language. Having to deal with dual tasks imposes a heavy cognitive load on novices. Finally, these programming solutions require a certain way of thinking. Therefore, it is vital for students to have a correct and deep understanding of what a program is actually doing at each stage (Milne & Rowe, 2002). Such understanding can enable the students to reason at a higher level (Teague & Lister; 2014). Program comprehension is thus shown to be critical in student learning because it is required for students to perform tasks requiring higher reasoning abilities such as problem solving. It has also been shown to be critical in the development of debugging skills, which is a fundamental skill for a software engineer (Ahmadzadeh et al., 2005). However very little time and attention is paid to develop program compression skills in the traditional classroom settings.

This project aims to help enhance learning by providing a customized learning experience to improve their fundamental understanding of programming, which is needed to progress to higher levels of thinking. We have developed Active Debugger, which is an Intelligent Tutoring System (ITS), that is designed to help students to develop program comprehension skills within the context of C++.

Several ITSs have been developed to teach programming. These systems focus on different languages including Java and Lisp. Some tutors focus on helping students to develop programs using a problem statement. J-LATTE (Holland et al., 2009) and LISP Tutor (Reiser et al., 1985) are two such tutors. The focus of iList (Fossati et al., 2009) is developing programs to manipulate linked list data structures whereas ChiQat-Tutor (Omar et al., 2014) focuses on recursion. Thus iList and ChiQat-Tutor cover specific areas and have a narrower focus compared to J-LATTE and LISP tutor, which cover the general task of problem solving. Our aim is to provide an effective learning experience that help students to learn program comprehension that is vital for problem-solving. We want to focus on C++ as it is used in many software projects and taught in many CS undergraduate programmes.

2. Active Debugger

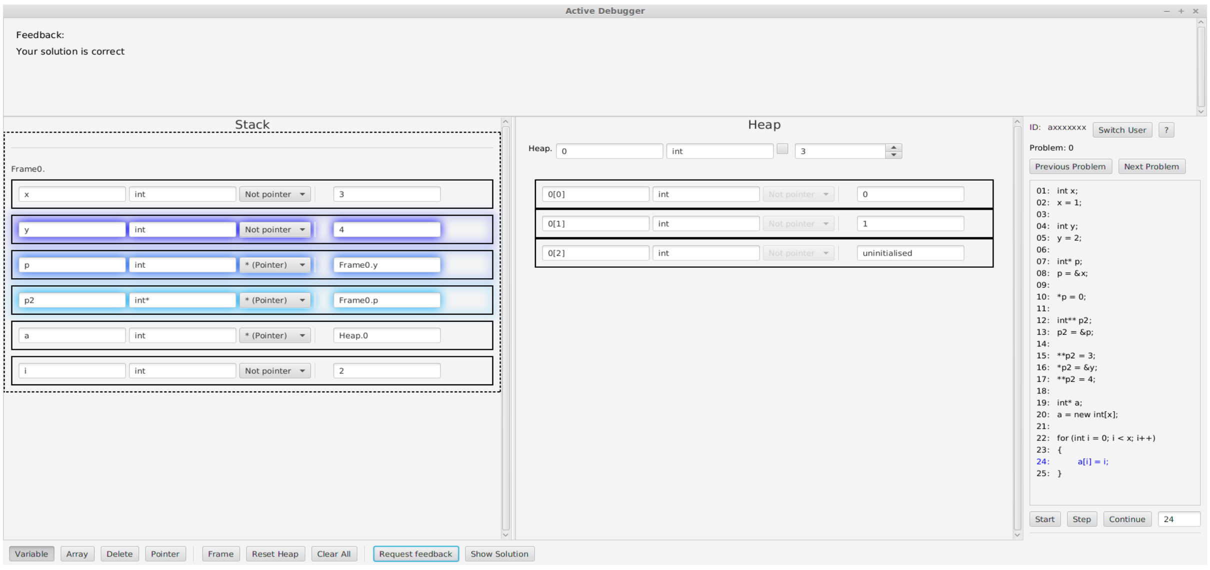
Active Debugger is an intelligent tutoring system that assists students learning C++ language comprehension. The system is designed to complement classroom teaching, and thus we assume that the students are already familiar with the fundamentals of C++ .

Active Debugger provides a problem-solving environment, in which students are given a program and asked to specify the changes that take place in memory when each line of code is executed. Students also have the freedom to skip to a specific line and specify the changes in memory up to that line of code. Programs provided by the system cover topics including pointers, recursion and parameter passing. Currently the problems are presented to the student in the increasing order of complexity.

The system is developed using constraint-based methodology (Mitrovic, 2011). The main components of the system are its user interface, pedagogical module, interpreter and the constraint-based modeller. Active Debugger contains a number of problems defined by a human expert. The ideal solution which is used to evaluate the accuracy of a student's attempt needs to be dynamically generated. This is needed because the student can request an ideal solution after going through any number of lines in the program. This requires generating the solution only up to the line of code the student has attempted. Thus having a pre-specified ideal solution is not possible. This is different to majority of the constraint-based which allows the human expert to specify a single ideal solution for each problem (Mitrovic, 2011). This is similar to NORMIT, which generates a solution for every step of the algorithm for data normalization. However Active Debugger has to generate solutions at a finer granularity than NORMIT as it can generate solutions for any line of code in the program.

The system assists students during problem solving and guides them towards the correct solution by evaluating the student attempt and providing feedback. The system supports individual work. The interface of Active Debugger is given in Figure 1. The interface consists of different areas. The current problem along with the corresponding controls is displayed on the right of the screen. Here the student is allowed to change the problem as well as step through the selected program. The feedback is displayed at the top of the screen. The middle is the workspace where user specifies the changes that take place in the memory. The middle is split into two areas called *Stack* and the *Heap*. The area on the left is a representation of the stack and the right represents the heap. These two areas collectively represent the memory for the program. This representation is designed to help the students to distinguish between these two different areas of memory. In addition, it is important for a student to know where each variable should be stored (i.e. in the heap or in the stack). If a student places a variable in the stack instead of the heap, he/she provided feedback about this error. Thus, our system helps students to learn the importance of understanding, which variables will be stored in which part of the memory. The student is able to edit all the relevant properties of the variables that are stored in each area using the interface controls in the bottom (Figure 1).

The line that would be executed next is highlighted in blue in the problem (Figure 1). While students can specify the changes after each line of code, they also can specify the changes after a few lines of code. For the latter option, they need to specify a line number that they want to move to. The blue line that indicates the next line to be executed will be changed accordingly. After moving to that line, a student can specify all the changes in the memory up to that line. Active Debugger is able to generate the ideal solution up to that line and it will be used to evaluate a student's attempt. Even though it is beneficial to provide this type of freedom to skip lines of code, it is also necessary to prevent students from progressing too far without realising they have incorrect knowledge. In order to support this, Active Debugger allows lines to be marked with a '#'. Students cannot pass such lines without providing the correct solution. Authors of problems are expected to use '#' at pedagogically appropriate places. For instance a pedagogically suitable place to include the '#' symbol is at the end of a *FOR* loop. This is similar to a breakpoint in a typical debugger.



Feedback:
Your solution is correct

Frame0.

x	int	Not pointer	3
y	int	Not pointer	4
p	int	* (Pointer)	Frame0.y
p2	int*	* (Pointer)	Frame0.p
a	int	* (Pointer)	Heap.0
i	int	Not pointer	2

Heap. 0 int 3

0[0]	int	Not pointer	0
0[1]	int	Not pointer	1
0[2]	int	Not pointer	uninitialised

ID: axxxxxxx

Problem: 0

```
int x;
x = 1;

int y;
y = 2;

int* p;
p = &x;

*p = 0;

int** p2;
p2 = &p;

**p2 = 3;
*p2 = &y;
**p2 = 4;

int* a;
a = new int[x];

for (int i = 0; i < x; i++)
{
        a[i] = i;
}
```

24

Figure 1: The interface of Active Debugger

The student can also request to view the correct solution. This will display the correct answer up to the line highlighted in blue, in a new window. This window will dynamically update as the student moves through the program. This enables the software to also be used as a simulator, which allows the students to monitor what effect each line of the program creates in memory.

There are several avenues for future research. One possibility is to extend the coverage of concepts provided by Active Debugger. Currently the problems cover parameter passing, pointers and recursion. The system can be extended to cover problems with objects. This will enable us to extend the system further to cover data structures. This will assist students to understand how data structures work by actively showing what happens in memory. We also have plans to evaluate the system with a bigger sample of authentic students in an authentic learning environment.

References

Ahmadzadeh M., Elliman D., and Higgins C. (2005) An analysis of patterns of debugging among novice computer science students.

Fossati D., Di Eugenio B., Brown C., Ohlsson S.,Cosejo D. (2009). Supporting Computer Science Curriculum: Exploring and Learning Linked Lists with iList. *IEEE Transactions on Learning Technologies*, 2(2):107–120.

Holland J., Mitrovic A., Martin B. (2009). J-LATTE : a Constraint-based Tutor for Java. In Kong S., Ogata H., Arnseth H., Chan C., Hirashima T., Klett F., Lee J., Liu C., Looi C., Milrad M., Mitrovic A., Nakabayashi K., Wong S., and Yang S., editors, Proceedings of the 17th International Conference on Computers in Education ICCE'09, (pages 142–146).

Milne I., Rowe G. (2002). Difficulties in Learning and Teaching Programming Views of Students and Tutors. *Education and Information Technologies*, 7(1):55–66.

Mitrovic A. (2011).Fifteen years of constraint-based tutors: what we have achieved and where we are going. *User Modeling and User-Adapted Interaction*, 22(1-2):39–72.

Mitrovic, A.,Weerasinghe, A. (2009). Revisiting Ill-Definedness and the Consequences for ITSs. Brighton, UK: 14th International Conference on Artificial Intelligence in Education (AIED2009) (pp. 375-382).

Omar A., Fossati D., Di Eugenio B., and Nick G. (2014). ChiQat-Tutor: An Integrated Environment for Learning Recursion. Technical report, Carnegie Mellon University, University of Illinois at Chicago.

Reiser B. J., Anderson J. R., Farrell R. G. (1985). Dynamic Student Modelling in an Intelligent Tutor for Lisp Programming. In IJCAI '85, (pp. 8–14).

Teague D., Lister R. (2014) Programming : Reading , Writing And Reversing. Proceedings of the 19th ACM conference on Innovation and technology in computer science education - ITiCSE '14, (pp. 285–290).