Pre-publication draft of a paper which appeared in the Proceedings of ICCE'98, Volume 2., 213 - 216

Overview of a low-level Program Visualisation Tool for Novice C Programmers

Philip A. Smith and Geoffrey I. Webb
Deakin University.

Abstract

As a programming novice attempts to attain expertise in programming she must develop adequate mental models and knowledge structures of the programming process. Unfortunately, many of the computerised tools to which novice programmers have access are designed by expert programmers for experts and as such do not meet the needs of novices. Low-level program visualisation tools make explicit the internal workings of program execution and as such can serve as conceptual models onto which novices can assimilate information about programming. This paper discusses the need for such a tool, what features such a tool may include and gives a brief description of an evaluation of a low-level program visualisation tool developed at Deakin University.

1 Introduction

One of the necessary requirements for progression beyond programming novicehood is the development of appropriate schemata and mental models of the programming processes. It is the task of the educator to provide conceptual models that will promote the development of such internal knowledge representations. This is usually done through laboratory sessions and blackboard teaching.

The computerised aids available to novice programmers in laboratory situations are often commercial debuggers designed for expert programmers. While many computerised aids designed specifically for novice programmers have been developed, their use is not widespread. One of the possible reasons for this is that their benefit has not been sufficiently well demonstrated (Price et al 1993).

Probably the most vulnerable stage of a programmer's development is the "raw novice" stage at the very beginning of her learning experience. Computerised aids that reinforce mental models of the programming process are more appropriate for novice programmers than commercial environments, which have been developed by expert programmers for use by expert programmers. Such aids should comply with duBoulay's principles of "simplicity" and "visibility".

One of the concepts with which learners of procedural languages need to come to terms is that of the sequential nature of program execution. While novices are usually quick to grasp the concept of a program being a sequence of events they can forget that each instruction is executed in the environment created by the instructions that preceded it (Du Boulay, 1986). Hille & Higginbottom (1983) also commented on the difficulty that students have in understanding the "sequential nature and dynamic aspects" of programming and suggested that a lot of errors result from this lack of understanding.

Program visualisation tools are designed to make visible aspects of programming often hidden from the programmer. As such they are capable of promoting "low-level" models of programming such as models of execution. They can reinforce a model of program execution by explicitly showing how the execution of a statement affects the program state and hence the environment in which the following statement is executed.

2 Specific Problems with Teaching C

For years Pascal was the language of choice as an introductory teaching language and it is still the most popular language to be used as a first language for students in the nineties (Brilliant & Wiseman, 1996). However, many institutions have moved away from using Pascal in this way. One of the main reasons for the change was that it was felt that Pascal was no longer able to adequately demonstrate the programming concepts needed in an introductory course.

The use of C as an introductory programming language is becoming more widespread in recent years as the use of Pascal wanes and alternatives are sought. The problems experienced by novice programmers learning a procedural language such as Pascal as their first programming languages are exacerbated when learning C. With few exceptions (Newlands, 1992), most of the researchers who have discussed the use of C as a first programming language concede that it is more difficult to learn than Pascal (Gilbert & Forouzan, 1996) and that it raises many pedagogical problems when trying to convey programming concepts (Mody, 1991). Many of these researchers vilify its use and simplified versions of C have been created to overcome the pedagogical problems it poses (Ruckert & Halpern, 1993).

In the institutions in which C was introduced it was felt that despite concerns about its pedagogical fitness its use in industry was so widespread as to justify its incorporation as an introductory programming language (Newlands 1992). Since Deakin University was one of those which converted to C, it was C that was chosen as the programming language for the low level program visualisation tool aimed at assisting novice programmers to be used to evaluate the potential of such tools. The following sections describe in detail some of the key techniques and features that might be included in such a low level program visualisation tool and how they were implemented in the tool developed at Deakin University.

3 Features of a Glass-box Interpreter for Novice Programmers

The design objectives of the current project were to

- evaluate the efficacy of a low level program visualisation tool as a learning aid for novice programmers; and
- develop and evaluate features which might further enhance the benefit of a low-level program visualisation tool.

With these objectives in mind, a low-level program visualisation tool (called Bradman), designed specifically to provide a conceptual model of program execution for C programs, was developed (Smith & Webb, 1995). The model it presents is that of a program (written in a procedural language) being a dynamic entity which obtains its results by means of sequential change of program state through the execution of program statements. This model provides a cognitive framework onto which the student can assimilate the new information about programming that she is learning. As this study of the problems facing novice programmers highlighted the importance of this issue, Bradman emphasises the nexus between the execution of program instructions and the subsequent change of program state.

Bradman is an interpreter which makes visible aspects of the programming process which are normally hidden from the user. For this reason, we call it a “Glass-box Interpreter”. Bradman is designed to enable students to develop their own programs as well as run those created by educators for pedagogical purposes. Bradman also incorporates novel features designed specifically to assist novice programmers. These features include:

- An explicit depiction of a program statement as an active agent modifying program states.
- A textual explanation of how each statement achieves its effects. This feature can be used to assist students who have difficulty understanding why a particular statement caused the effect it did.

- More visible input/output facilities. In particular, information about what is contained in the input buffer and some facility to inform the user that the program is waiting for input before continuing execution.

A more ambitious purpose for Bradman was to evaluate which individual features of such tools would be of benefit to novice programmers and assist them in their learning experience. It is possible for novices to become confused when faced with the sophisticated array of options available in systems designed for expert programmers (Freund & Roberts, 1996). Hence, it is important that only features from which novice programmers benefit are included in such a tool. This complies with Du Boulay et al.'s (1981) principle of simplicity.

Bradman has been developed to fulfil specific research objectives and assisting novice programmers with program syntax which lies outside the scope of the current project. Hence the source code that Bradman is called upon to analyse must be syntactically correct. If the source code contains syntactic errors, then Bradman displays an error message and exits.

The syntactically correct source code is parsed producing an efficient internal representation in the form of a syntax tree. This internal representation is never presented to the user, the corresponding source code being referred to whenever appropriate. As each statement is executed, code embedded in the run-time machine sends information to the various windows which provide different views of the program. Four of these windows are displayed at all times while others appear when certain conditions arise.

4 Evaluation

Bradman was evaluated in an experiment in which data based on performance measures were gathered (Smith & Webb, 1998). This experiment, involving twenty four volunteers from Deakin University's introductory programming unit, was conducted over a period of three weeks. In each of these weeks the participants performed programming tasks in a two hour laboratory session. The students were split into two groups - a test group which had access to Bradman during the experimental phase and a control group which did not. By the end of the experimental period the test group was performing significantly better than the control at desk-checking exercises - tasks designed to evaluate programming expertise that involve manual interpretation of a program. Evidence also suggested the possibility that access to Bradman actually enhanced the subject's mental model of programming.

5 Conclusion

Program visualisation tools provide different views of program execution. Many of the views provided are of aspects normally hidden from the user. As such, these tools can be of assistance to novice programmers by providing conceptual models of program execution. These conceptual models provide a framework onto which the student can assimilate new knowledge of programming. However, whether the provision of such models is of significant assistance to novice programmers is not yet proven. There is little empirical evidence as to the efficacy of such tools under controlled conditions.

Bradman is a glass-box interpreter which was developed as a testing tool to obtain such evidence. Bradman contains many of the features of standard visibility debuggers but also contains novel features which are designed solely to assist novice programmers. These features include a variables display which reinforces a model of a program as an active entity which achieves its results through an on-going process of program state changes brought about by the execution of program statements. Also included is a verbal explanation of each statement as it is executed. Experimental evidence based on performance measures demonstrates that students with access to Bradman can perform significantly better at programming tasks than students without access.

References

[1] Brilliant, S.S. & Wiseman, T.R. (1996). **The First Programming Paradigm and language Dilemma**, *SIGCSE '96*, 338-342.

[2] duBoulay, B., O'Shea, T & Monk, J. (1981). **The Black Box inside the Glass Box: Presenting Computing Concepts to Novices**. *International Journal of Man-Machine Studies*, 14, 23 7-249.

[3] duBoulay, B. (1986). **Some Difficulties of Learning to Program**. *Journal of Educational Computing Research*, I, 57-73•

[4] Freund, S. N. & Roberts E. S. (1996). **THETIS: An ANSI C Programming Environment Designed for Introductory Use**. *Special Interest Group on Computer Science Education*, 28(1), 300-304.

[5] Gilbert, R.F. & Forouzan, B.A. (1996**). Comparison of Student Success in Pascal and C language Curriculum**. *Special Interest Group on Computer Science Education*, 252-255.

[6] Hille, R.F. & Higginbottom, H. (1983). **A system of visible execution of Pascal Programs**, *Australian Computer Journal*, 15(2), 76-77.

[7] Mody, R.P (1991). **C in Education and Software Engineering**, *SIGCSE Bulletin*, 23(3), Sept., 45-56.

[8] Newlands, D.A. (1992). **C as a first programming language**. In *Proceedings of Australian Society for Computers in Learning in Tertiary Education*, Sydney, Australia, 339-345.

[9] Price, B. A., Small, I S. & Baecker, R. M. (1993). **A Principled Taxonomy of Software Visualisation**. *Journal of Visual Languages and Computing*, 4(3), 211-266.

[10] Ross, R.J. (1991) **Experience with the DYNAMOD Program Animator**. *Special Interest Group on Computer Science Education*, 23(1), 35-42.

[11] Ruckert, M. & Halpern, R. (1993) **Educational C**. *Special Interest Group on Computer Science Education*, 23(1).

[12] Smith, P.A. & Webb, G.I. (1995) **Transparency Debugging with Explanations for Novice Programmers**. *Proceedings of the 2nd Workshop on Automated and Algorithmic Debugging*. St. Malo, France

[13] Smith, P.A. & Webb, G.I. (1998) **Evaluations of a low-level program visualisation tool for Novice C Programmers**. *Technical Report, TR C98/14* Deakin University.