VisMod: A Beginner-Friendly Programming Environment*

Ricardo Jiménez-Peris
Marta Patiño-Martínez
Jorge Pacios-Martínez

Universidad Politécnica de Madrid
Facultad de Informática
28660 Boadilla del Monte (Madrid), Spain
{rjimenez, mpatino}@fi.upm.es

Keywords: Visual Debugging, User Interfaces, Interpreters, Laboratory Environments, CS1-2.

ABSTRACT

Although modern programming environments offer nice features that enhance the development and debugging of programs, they generally do not favor learning. Students need environments that facilitate the understanding of the dynamic aspects of programming. In this work, we present a programming environment, VisMod, to assist the student in learning programming as well as understanding his/her programs. VisMod visualizes the execution of programs. It allows students to develop their programs and see how they work. It also enforces a good programming style, checking style errors like functions with side effects.

1 INTRODUCTION

First year programming students often have great difficulties in understanding their programs' behavior upon executing it. The confusion is even stronger when more advanced topics, such as recursion, or dynamic memory are tackled. Students do not have interactive tools that facilitate the development and debugging of programs. Programming environments and debuggers show programs with these elements in a non-intuitive fashion. Commercial debuggers are void of pedagogical features that could be useful to beginning programmers. The debuggers are written for developers of professional applications and do little to assist the new comers in their learning process.

Visual tools facilitate learning programming. They increase the comprehension of the internal workings of the program. Several tools have been developed for that purpose. These include program animators, algorithm animators and computer science concept animators. A program animator [2, 7] provides the user with a detailed, highly visual view of the source code of a program in execution. Algorithm animators do not in general display source code, but provide a graphical picture of an algorithm in action [3 − 6, 8]. They are too complicated for novices, especially if they have to build the animations. Alternatively, the teacher could build them, which definitely takes a considerable amount of time and effort.

Computer science concept animators are specialized in animating a particular algorithm or kind of algorithms. For instance the CABTO system [1] (Computer Animation of Binary Tree Operations) animates basic algorithms for binary trees.

We have developed an environment, VisMod that animates programs in Modula-2. VisMod allows the student to concentrate on the development of the program, with no special attention to visualization, and then visualize its execution. Some environments are not intended to develop programs, but only visualize a set of program libraries [2], while others have limited graphical facilities [7].

VisMod provides most of the facilities found in commercial programming environments and is also equipped with an integrated visual debugger and a compiler that enforces good programming techniques. For instance, it does not allow functions with side effects, and displays the code with a predefined type of indentation.

VisMod can be used in lectures, laboratories and at home. It runs on PCs that are widely available for students. We have been using them in our lectures and our students have developed their projects with it. We find that students using VisMod understand and learn more quickly than students who are in traditional environments do.

In this paper, we present the properties of VisMod. We have divided them into three categories. Section 2 presents the environment. In Section 3, we illustrate the visual debugger, and in Section 4, we explain additional features. We conclude with some thoughts about our experience with VisMod.

*This work has been partially funded by the Spanish Research Council(*CICYT*), contract number *TIC98-1032-C03-01* and the Madrid Regional Council (*CAM*), contract number *CAM-07T/0012/1998*.

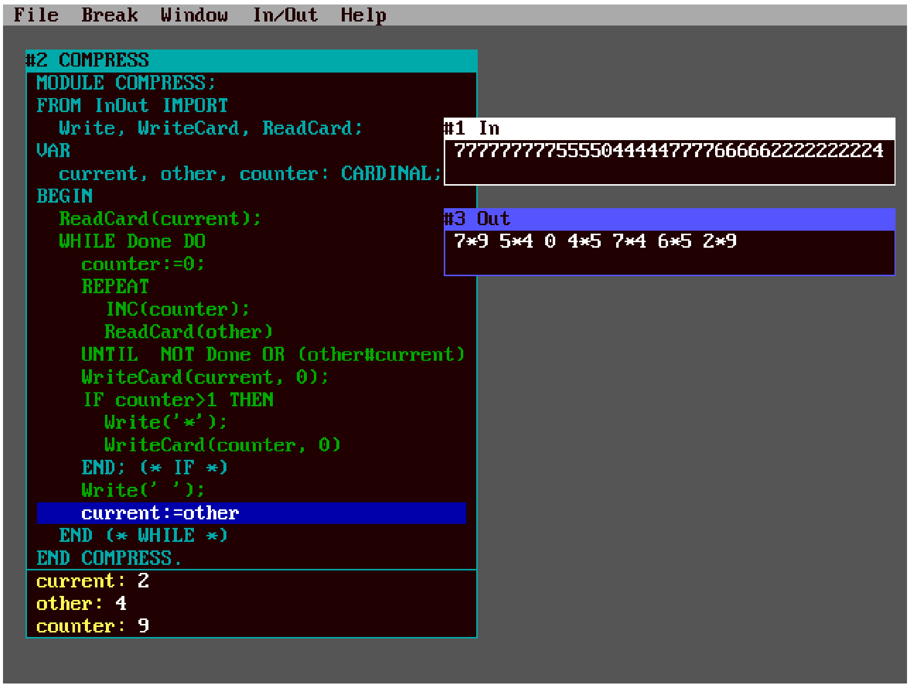

Figure 1: Compression of a number string

2 THE ENVIRONMENT

VisMod is a turbo-like environment. The main window has pull-down menus. The File pull-down menu provides access to existing files and the creation of new ones. Editing of a file is done like in other text editors. The compiler can be invoked from the environment clicking on the corresponding menu. After compiling, the program is ready to run. There are three ways of running a program: step-by-step, until a breakpoint or completely. In the step-by-step mode, the user controls the statement that is to be executed next. When some parts of a program are understood, there is no need to run in a step-by-step mode anymore. By setting breakpoints, one can focus on those parts of the code of interest only. The interaction with the user can be limited to input data, while the program is visualized, choosing the run mode until end.

Three kinds of windows can be displayed on the screen at any given time, during the execution of a program. One window is for the code, associated variables (variables of the main program and subprograms) and parameters. A second window displays the input/output activity and the third is devoted to dynamic memory. The code window has two panes, one with the program and other with variables declared in the scope of the code. In the code pane, a cursor bar highlights the statement in execution, while in the state pane, pairs of identifiers and values are shown. Every time a subprogram is called, a new window is displayed with its code, parameters and local variables. We also visualize structured types, like arrays and records.

Pointer variables and parameters are not visualized in the state pane, but in the dynamic memory window. Pointers are represented with boxes and addresses with arrows. Boxes are labeled with the variable or parameter identifier. To distinguish among pointer parameters and variables of different (usually recursive) calls, a number corresponding to the window label of the subprogram activation is appended to the parameter identifier.

Unlike other environments that have only one window for input/output, the input/output window in VisMod can be split into two different windows in order to distinguish input data from output (fig. 1).

Beginning programmers often have problems in remembering the syntax and semantics of the programming language [7]. For this reason, the environment also includes a contextual help system of the language syntax and semantics. It assists students in the development of programs.

3 VISUAL DEBUGGER

As previously mentioned, commercial programming environments and debuggers are not adequate tools for understanding the execution of programs. They were not designed for beginners and even advanced students are at a loss when the applications require more sophisticated data structures and nontrivial programming techniques. The programming environments generally provide poor visualization facilities. For instance, they do not display dynamic memory in a friendly way. Very often, they merely show memory addresses, and needless to say that one is not interested in addresses when trees, stacks or other data structures are being manipulated. Students are familiar with a very different kind of representation. In books, as well as in class, arrows

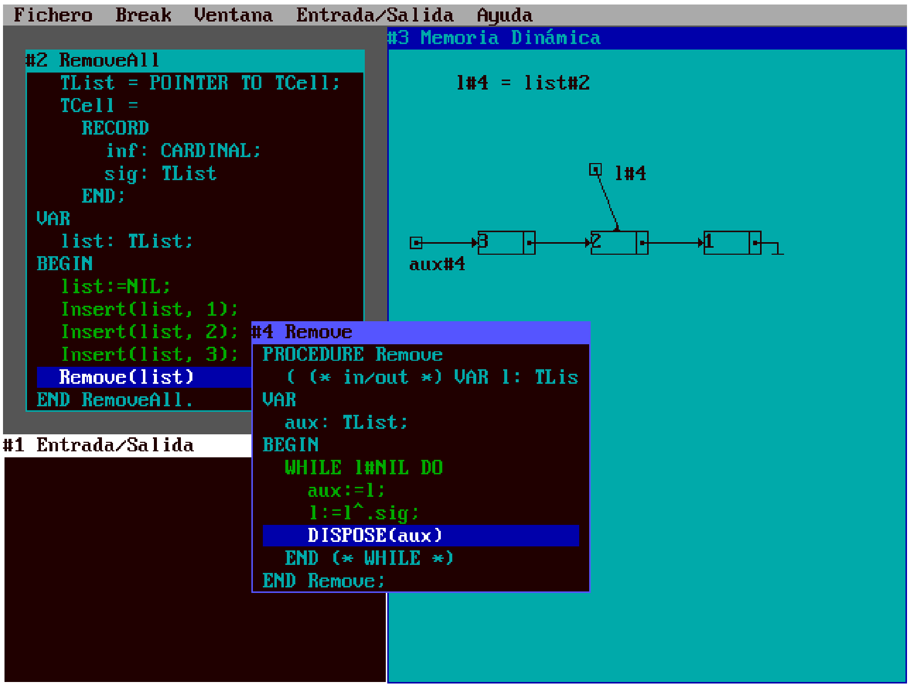

Figure 2: Iterative removal of a list

represent addresses, and the object pointed at are represented by rectangles. This is how dynamic memory is represented in VisMod.

To illustrate how dynamic memory is displayed, let us consider the problem of removing a list. In figure 2 there are two code windows, one for the program and the second for the procedure. There is also a window showing dynamic memory (including pointer variables and arguments). It corresponds to the time when the first item in the list is going to be disposed (the cursor bar shows the next statement to be executed). Notice that there is no state pane, as the only variables and parameters are pointers. Pointer variables are shown in the dynamic memory area, aux and l. Pointers passed by reference are shown providing a list of name equivalences (l#4 = list#2 in fig. 2). Parameters are only shown while the call is pending. When the call finishes, the box with the code also disappears and the cursor bar in the previous window moves to the next statement.

This kind of representation is automatically provided for linear data structures and trees.

Recursion is another difficult topic in which debuggers do not help much. The representation is generally the same as with non-recursive subprograms in that when a recursive call is made the cursor moves to the beginning of the subprogram, but pending calls are not shown. When the subprogram finishes, the same code is shown several times with the cursor moving up and down. In VisMod, every time a recursive call is made a new window with the subprogram code and data is displayed. Each window has a cursor bar, so it is very easy to see the state of each pending call. Furthermore, the students easily understand the iteration behind recursive calls as they see the unfolded windows, each with its own data. Let us consider the traditional factorial problem to illustrate recursion. As can be seen in figure 3, VisMod shows the state of pending calls, and when each call finishes, the value returned is shown on the right of the statement that returns the value. In commercial debuggers, one has to evaluate the returned expression (if this facility is available) before executing the return sentence. For beginners this constitutes an obstacle so, they will not be able to follow the code.

VisMod not only visualizes simple recursion, but can also handle mutually recursive subprograms. In figure 4, two mutually recursive functions, even and odd, are shown.

As can be seen in the next example, VisMod can also visualize more complicated algorithms that use both dynamic memory and recursion.

Figure 5 illustrates how a new element is appended at the end of a list. Pointer parameters of the different calls to Append are distinguished by their names followed by the window number of the activation (e.g. l(#4) indicates parameter l of the activation shown in window #4).

Algorithms involving recursion and dynamic memory are difficult to follow for beginners. The situation becomes even worse when there are pointer parameters passed by reference or there are statements after the recursive call. As has been shown, these problems are more easily understood using VisMod.

4 OTHER FEATURES

Students learn more readily language syntax and semantics than programming style. Methodological issues are more challenging to teach than mere syntax. In our

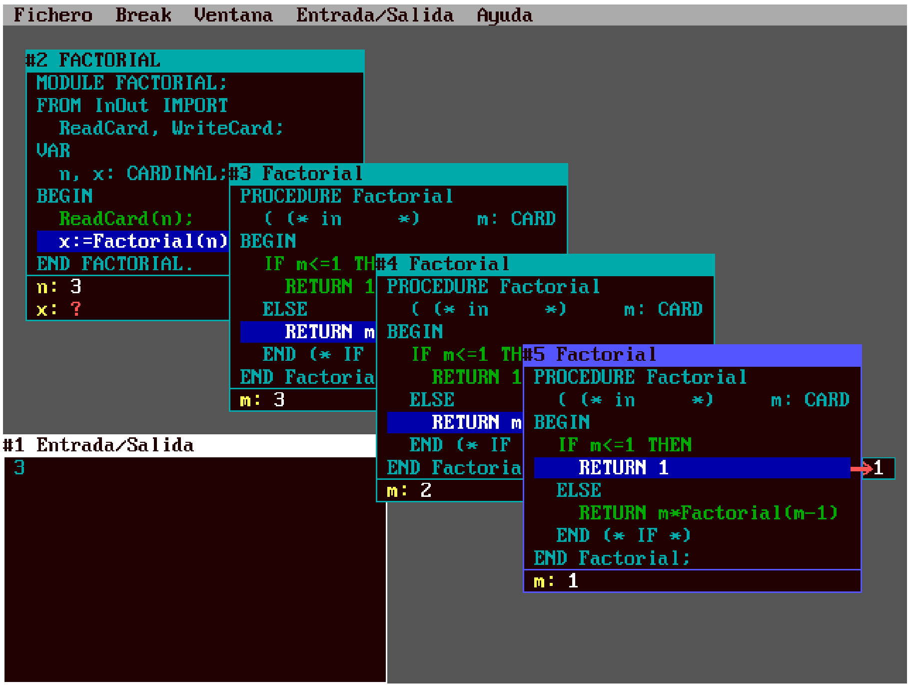

Figure 3: Recursive factorial

environment, some of these methodological issues get the same treatment as the syntax of the language. VisMod checks these issues and warns the user as if they were syntax errors.

One of the most common issues is the use of global variables instead of parameters. Most beginner programmers use them just because they do not understand parameter-passing mechanisms. This situation is treated as a semantic error. Of course, this can be useful for certain algorithms, but it can be overridden changing a configuration file. There is no menu option to prevent students from changing it.

VisMod also checks that all variables in an expression have been previously initialized. It also makes sure that all the declared variables are used.

In Modula-2, the RETURN statement is used to return the result of a function. This statement should be used just before the end of the function, but students tend to misuse it. For instance, they use it to exit from loops because they do not know how to write non-simple conditions. VisMod checks that they are only used as the final statement in a function or in all the branches of a selection statement, if it is the last one. In other words, it makes sure that the return statement does not prevent the execution of any ulterior statement.

Another problem found with beginners is that of indentation of programs. Most of them write programs as if it were narrative text, incurring a total lack of legibility. VisMod displays the code using a pretty printer. Thus, the students see their programs written with the same style as that used in lectures. As the student progresses, the pretty printer can be switched off, so that the environment will show the student's own code. This option is also indicated in the configuration file.

Finally, students tend to compare Boolean variables and expressions with the values true or false when they want to check their value. They write:

IF logical = TRUE THEN...

VisMod also checks this kind of redundancy and an error message is displayed.

5 CONCLUSIONS

A visual environment for learning has been presented. This environment is not only user-friendly, but also beginner-friendly; it has the properties of commercial environments and also animates programs. It is adequate to use during lectures and also for students to run their own programs and see how they work. It runs on a widespread platform, the Intel-based PCs. It can be used by beginning programmers as well as by more advanced students.

We have used VisMod in several of our classes, and have received very positive feedback from students. Although we have no formal statistics, we find that they learn concepts such as parameter passing mechanisms, scope rules, recursion and dynamic memory faster and better than with traditional settings or other packages. Consequently, they have fewer difficulties with projects, which involve most of these concepts.

We plan to add new features to VisMod, such as reverse execution, more templates to visualize data structures such as graphs and other kinds of trees and adapt it to other languages like Pascal and Ada.

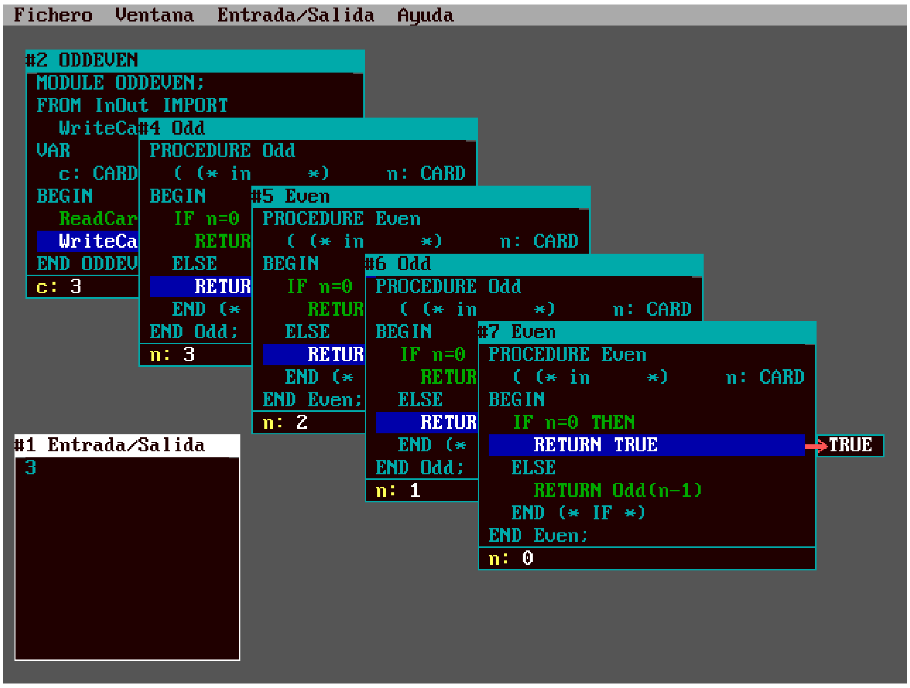

Figure 4: Odd and Even Mutually Recursive Functions

ACKNOWLEDGMENTS

We wish to thank to Sami Khuri for his help and comments, and to the rest of the members of the VisMod team: Carlos Illana and Jesús María Román.

REFERENCES

1. G. M. Barnes and G. A. Kind. Visual Simulations of Data Structures During Lecture. Proceedings of the 18th SIGCSE Technical Symposium on Computer Science Education, 1987, 267-276.

2. M. R. Birch, C. M. Boroni, et al. Dynalab: A Dynamic Computer Science Laboratory Infrastructure Featuring Program Animation. Proceedings of the 26th SIGCSE Technical Symposium on Computer Science Education, 1995, 29-33.

3. M. H. Brown. Exploring Algorithms Using Balsa-II. IEEE Computer. May 1988.

4. M. H. Brown. Zeus: A System for Algorithm Animation and Multi-View Editing. Proceedings of the IEEE Workshop on Visual Languages, 1991, 4-9.

5. E. I. Giannotti. Algorithm Animator: A Tool for Programming Learning. Proceedings of the 18th SIGCSE Technical Symposium on Computer Science Education, 1987, 308-314.

6. T. Naps. Algorithm Visualization in Computer Science Laboratories. Proceedings of the 21th SIGCSE Technical Symposium on Computer Science Education, 1990, 105-110.

7. I. Sanders and H. Gopal. AAPT: Algorithm Animator and Programming Toolbox. SIGCSE Bulletin 23, 4 (Dec. 1991), 41-47.

8. J. Stasko. Tango: A Framework and System for Algorithm Animation. IEEE Computer 23, 9 (1990), 27-39.

Ricardo Jiménez-Peris holds a MS degree in computer science from Universidad Politécnica de Madrid (1990). His interests include fault-tolerance, transactional systems, distributed systems, visualization and computer science education. He has been assistant professor in computer science at Universidad Politécnica de Madrid since 1990. He is member of the Association for Computing Machinery.

Marta Patiño-Martínez received a MS degree in computer science in 1990 from Universidad Politécnica de Valencia. Her interests include fault-tolerance, transactional systems, distributed systems, visualization and computer science education. She has been assistant professor in computer science at Universidad Politécnica de Madrid since 1990. She is member of the Association for Computing Machinery and the IEEE Computer Society.

Jorge Pacios-Martínez holds a degree in Computer Science from the Universidad Politécnica de Madrid. He is currently software engineer at GMV (Group of Flight Mechanics). His research interests include compilers, visual debuggers and programming environments.

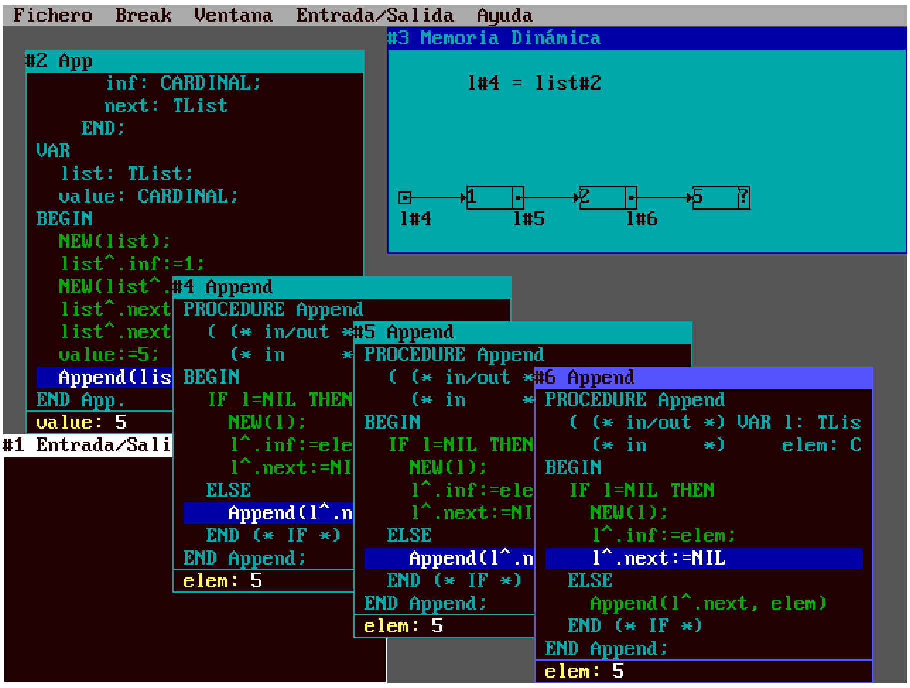

Figure 5: Recursively append an element to a list