CGRAPHIC: Educational Software for Learning the Foundations of Programming

ANTONIO J. FERNÁNDEZ, JESÚS MILLÁN SÁNCHEZ

Dpto. Lenguajes y Ciencias de la Computación, ETSII, Campus de Teatinos, Universidad de Málaga, 29071 Málaga, Spain

Received 20 February 2003; accepted 7 July 2003

ABSTRACT: We present CGRAPHIC, an original software for learning the basic concepts of implementing (possibly educational) software. CGRAPHIC is being used in first-year university courses in engineering studies, and its main aim is to complement the role of the teacher and improve the quality of teaching foundations of programming. CGRAPHIC is a virtual tutor, a programming textbook (online), and an interpreter as well as a debugger of the programming language C, and can be executed on any navigator program in Internet or locally. This software enables different levels of learning (theoretical and practical) and allows the student to study—intuitively, graphically, and gradually—the principles of programming that are crucial to developing new software. © 2004 Wiley Periodicals, Inc. Comput Appl Eng Educ 11: 167–178, 2003; Published online in Wiley InterScience (www.interscience.wiley.com); DOI 10.1002/cae.10048

Keywords: teaching of programming; programming principles; educational software implementation

INTRODUCTION AND BACKGROUND

Recently, there has been a growing interest in the development of educational software in all areas of study. To develop new software, it is very important to have good programming skills as well as to follow an adequately structured programming style in order to save debugging time [1]. During the last decades, structured programming has been a matter mainly taught in university schools of computer studies (SCS), where this issue is very important to consolidate the basic concepts necessary to understand each topic of the full program.

However, implementation of educational software is not only a matter specific of experienced programmers but also of many engineers. In fact, nowadays any engineer is required to have an appropriate background about programming since this issue is essential to develop original software for professional applications or educational purposes. A direct consequence is that computer science, and more specifically programming, is becoming a basic discipline in
most of the schools of any engineering (not only SCs) that offer, generally in the first academic year, at least a subject related with the Foundations of Programming. This highlights the importance of learning the foundations of programming in the studies of engineering and explains why Education in Computing \([2–4]\) has emerged as one of the most exciting fields of research in recent years.

Generally speaking, the final objective of a subject intended to teach programming foundations is that the student is familiarized with the main notions of structured programming, has some ideas on the concept of 'good style' in programming, and understands the basic notions of abstract data types \([1]\). In the end of an introductory programming course, the student should be able to write and modify small-scale programs using an imperative programming language including management and design of libraries.

However, in engineering studies, programming is one of the subjects that more time requires for its understanding. A lot of concepts of programming (e.g., pointers, loops, or structures) are quite abstract and the students often do not understand them clearly. Besides, with respect to learning programming discipline, there is a clear absence of educational software to support it. One of the reasons is that it is paradoxical to implement educational software to learn how to design (possibly educational) software. However, the importance of educational software is well-known \([3,4]\) and, thus, the importance of knowing how to implement it correctly is critical.

This paper presents CGRAPHIC, an original educational software that acts as a virtual tutor to teach foundations of programming. CGRAPHIC is an interactive and graphical environment to be used in a first-year university course of programming. Its development and implementation have been completed and we think that it is ideal to help the student to explore gradually the basic concepts of programming.

**TEACHING OF PROGRAMMING: KNOWN PROBLEMS**

The first step to evaluate in the teaching of programming in engineering studies is to analyze the contribution(s) of computer science to engineer education. According to Ref. \([5]\), this contribution is done on three levels: (1) to develop specific skills in computing such as learning a programming language or a software tool, (2) to acquire general knowledge about computing tools, and (3) to know professional applications for computing technology.

The first level, which is related with both teaching and learning of programming, has three main problems. The first one is that, in traditional lessons, the lecturer explains concepts by using just a blackboard and/or slides to complement the explanation. These resources are valid for a lot of theoretical subjects but are insufficient in more practical ones such as programming. A direct consequence is that many students are not able to assimilate all the theory and tend to concentrate on superficial aspects as there is no time to go deeper in the foundations. Therefore, programming courses are usually complemented with practical lessons although these are usually insufficient to provide the student a deep knowledge of the programming principles (sometimes because the teaching rhythm is superior to the learning rhythm and the student is not able to follow the lecturer explanations).

The second problem comes from the election of the first programming language. One of the most elected in this sense is C \([6]\) because it combines a high flexibility with a worthwhile efficiency. Moreover, a lot of industrial applications are programmed via C and, therefore, it is an ideal programming language for engineers \([6]\). However, it is also well-known that C is, compared to other high-level languages such as MODULA or PASCAL, a language difficult to teach as well as to learn and this makes more difficult to learn the foundations of programming if the teacher uses a language like C.

The third problem, for a student with no experience in programming, is the difficulty to understand how a program works. A structured program is an amalgam of declarations (e.g., variables, pointers, compound types, etc), instructions (e.g., loops and conditional sentences), and function definitions that do not follow a concrete coding sequence. Specifically, declarations are difficult to understand because of the different nature of the types employed in the program; instructions are not easy to follow (specially loops) due to their different behaviors as well as to the evaluation of conditional expressions over which their behaviors depend; and functions are confused because they break the sequential execution flow.

To overcome all these limitations cited above, we have implemented CGRAPHIC, a software that helps to teach both the foundations of programming and C, and also allows the student to analyze, step by step and graphically, the execution flow of a program that leads to a better understanding of how a program is executed. The rest of the paper is dedicated to describe CGRAPHIC.
DESCRIBING CGRAPHIC:
TWO LEVELS OF LEARNING

CGRAPHIC provides two levels of learning (and thus teaching): theoretical and practical.

Theoretical framework

In this level, CGRAPHIC provides theoretical explanations online in a similar way as in a classical programming textbook, that is to say, firstly a new topic is introduced and, afterwards, practical exercises are developed in order to clarify it. This level has to be considered more than a textbook since, for each topic, CGRAPHIC also proposes related concepts as well as exercises that can be accessed immediately online by means of dynamic links.

In this conceptual level, the current version of CGRAPHIC (totally operative) deals with the following principles of programming:

- simple data types (e.g., character, integer, real, ...),
- compound data types (e.g., arrays, structures, ...),
- formatted input/output and buffered input,
- type checking, type equivalence, and type conversion,
- (local, global, dynamic, static) variables and constants,
- variable scope,
- associativity and operator precedence,
- (conditional) expressions and basic statements (assignment),
- logical operators and control structures (conditionals and loops),
- procedures and functions,
- parameter-passing mechanisms,
- dynamic memory management (e.g., allocation and deallocation, pointers, memory addresses),
- secondary storage devices (i.e., file handling),
- linked lists and
- abstract data types (specifically stacks).

Figure 1  Welcome window and table of contents of the programming course. [Color figure can viewed in the online issue, which is available at www.interscience.wiley.com.]
Figure 1 shows the welcome window with the table of contents of the programming course, and Figure 2 displays part of the explanation associated with the topic control structures and logical operators.

**Practical Framework**

In the practical level, each topic is associated with a set of interactive exercises where each exercise requires to implement a structured C program that deals with or is related to the topic. The user can directly access the formulation of one exercise by just clicking over the adequate link. For instance, Figure 3 shows part of the set of exercises related with the concepts of both variables and simple data types and, Figure 4 shows the formulation of one of its related exercises.

In this level CGRAPHIC can propose, to the requirement of a user that clicks in the link Execute (see Fig. 4), a solution for a specific exercise. Then this solution can be interpreted graphically. By a graphical interpretation, we mean to visualize different states of the program that were derived during its execution. The state of a program in one specific moment is represented by the content, in that moment, of all the variables declared in the program, independently of the nature of these (e.g., static, local, global, dynamic, etc.). This interpretation gives place to a transition between states in which new objects appear and other ones disappear as consequence of the execution of the code associated to the program. This process enables the monitoring of the abstract concepts of programming as each state and its transition to the next one can be analyzed in depth.

The graphical interpretation of an exercise can be done in two modes: direct and step-by-step:

- in the direct mode, the program is executed as in a classical environment of programming in C, that is to say, in one pass. The great advantage with
In the step-by-step or unhurried mode, the program is executed instruction by instruction (not globally as in the previous mode) and the resolution (i.e., traces) of each instruction is visualized graphically. Users can take a deep knowledge of the basis of programming since all the states of the program are shown graphically one by one, from the initial state to the final one. Each state is represented by a graphical and intuitive view that can be easily followed by any student. Also, the learning rhythm depends on the users as they have the control on this mode because they are in charge of indicating the advance of one state to the next one and, as consequence, they decide the time to dedicate for analyzing the transition between states. The transition is accompanied with a graphical representation that helps the users to consolidate the principles of the foundations of programming (e.g., the contents of variables may change, new local variables may be created or destroyed by function calls, pointers may be modified, the memory map may change, etc.).

**CGRAPHIC OBJECTS**

The graphical environment of CGRAPHIC is constituted by the following elements identified in Figure 5:
- a graphical window to show the states of the program;
- a code window that displays the C code of the program to be executed; in the step-by-step mode the instruction being executed is emphasized in red color;

![Image](image_url)

**Figure 4** Formulation of an exercise. [Color figure can viewed in the online issue, which is available at www.interscience.wiley.com.]

- a dialogue window that simulates the standard output device (i.e., the monitor) in a classical C program and that also allows user interaction as here CGRAPHIC prints instructions for a correct execution of the program;
- a button to control the execution mode (mode selection);
- an input window that simulates the standard input device (i.e., the keyboard) in a classical C program;
- a reinitiate button (Init) to re-execute the exercise (perhaps in a different interpretation mode);
- an action button (Action) to enable the users control the state transition process.

**Implementation Details**

To avoid an excessive number of pages, we do not describe completely the implementation of CGRAPHIC as we prefer to concentrate in the user level. However, here we provide a brief global view of the
objects used in the implementation of CGRAPHIC. This information can be used to extend this software or to adapt it to another programming languages (e.g., PASCAL, MODULA, etc).

CGRAPHIC was developed in the Java programming language [7] and only requires a navigator program (e.g., Netscape or Explorer) with a resolution of 800 × 600 to be used on Internet. This means that CGRAPHIC is widely available. We have implemented a set of graphical objects that can be used generically to develop new interactive exercises. CGRAPHIC provides support for the following objects: variable, 1-dimension array, 2-dimension array, function, pointer, memory map, file, and structure (record). Each object is implemented by means of a Java class (i.e., via a file object.java) and each exercise (e.g., the exercise Ex) has an associated C code in another Java class (e.g., Ex.java) that, during the execution, is charged in the code window (see Fig. 5). The code window is defined in the file...

Figure 6  Schema of the Java classes employed in the implementation. [Color figure can viewed in the online issue, which is available at www.interscience.wiley.com.]

Figure 7  Unhurried execution. Objects: memory map, functions, global and local variables. [Color figure can viewed in the online issue, which is available at www.interscience.wiley.com.]
Code.java. The rest of the graphical environment is implemented in the file GraphicBox.java, that is shared by all the interactive exercises (that also share the same template—implemented in Template.java).

Figure 6 shows the schema of all the classes employed in the development of CGRAPHIC and the relation between them. For more information about the implementation, the interested reader can contact the authors or visit the welcome window of CGRAPHIC in the following web address http://campusvirtual.uma.es/fundinfo/online/English_CGRAPHIC/default.htm that contains an online link with more precise information useful for experienced programmers, e.g., information about each Java class, the methods inherited from another classes, constructor details, method details, and links to the subclasses. We emphasize that CGRAPHIC can be extended (with new topics and examples) by just making use of the generic classes here defined.

CGRAPHIC BY EXAMPLES

So far we have described CGRAPHIC, our motivations and our aims. In this section, we show several examples of the execution of different exercises in the practical level to give an idea of the potential of this software. Specifically we show intermediate states resulting from unhurried executions.

Swap Two Numbers

Figure 7 displays a state derived from the execution step-by-step of a solution for the exercise “Swap using Functions” whose formulation was shown in Figure 4. In the code window, there is a solution, proposed by CGRAPHIC, to solve the exercise. The graphical box visualizes one of the states resulting from the execution of such a solution where it is possible to observe some of the graphical objects.

Figure 8 One intermediate state during File Writing. [Color figure can viewed in the online issue, which is available at www.interscience.wiley.com.]
provided by CGRAPHIC such as formal arguments—or variables declared in function headers (in green)—local variables (in blue), a piece of the memory map, the instruction to be executed next (in red), the return value (i.e., void) of the function Swap (in blue) etc.

**File Writing and Buffered Input**

Figure 8 deals with several topics such as file writing, strings, and buffered input. The formulation of this exercise is as follows: implement a program to write in the file “tested.txt” the content of a string that is provided by the user from keyboard. Open the file in writing mode.

For the requirement of the user, a solution for this exercise is displayed in the code window of Figure 8 that also shows an intermediate state of the program execution in the graphical box. Observe that this state is represented by means of the adequate CGRAPHIC objects such as the following: a 1-dimension array (denoted as file) to contain the name of the file (i.e., “tested.txt”), a (file) pointer (denoted as f) that addresses the content of the file, a file object to represent the content of the file in one specific state, and a global variable to represent the content of the variable \( ch \) of type char.

Observe also one of the nice features of CGRAPHIC to explain graphically the standard input of datas: the keyboard buffer. It is not the purpose of this paper to explain this well-known concept in programming so that we will just say that this buffer helps to explain how the computer acts in the case of a massive introduction of datas, as for example when the program asks for one character (e.g., as consequence of executing an input instruction) and the user provides a sequence of them. In summary, the keyboard buffer contains those characters that have not been read yet from input instructions (in Fig. 8 contains the string “ANALYZE FILE WRITING”).

Despite that, buffered input is one of the concepts...
more difficult to understand by a non-experienced student, this can be easily explained and understood during the graphical transition of states.

Moreover, any student that had followed the previous topics proposed by CGRAPHIC will be able to understand easily the evolution of the variable \( ch \) during the unhurried execution and will also note that (a) this variable contains the last character read as consequence of executing the instruction `scanf("%c", &c)` in the body of the loop do-while, and that (b) this last character is printed in the file by the instruction `putc(ch, f)` that is also in the body of the loop do-while.

To highlight the differences with other states of the program, we show the final state in Figure 9. Observe that the file is now close and the string "Written file!!!" is printed in the dialogue window as consequence of executing, respectively, the instructions `fclose(f)` and `printf("Written file!!!")`. Note also that, the objects file, file pointer, and keyboard buffer have disappeared from the graphical window and only those corresponding to the global variables remain.

**How Does a Stack Work?**

Figures 10 and 11 deals with memory addresses, pointers, functions, structures, linked lists, and stacks i.e., an abstract data type (ADT). This complex exercise explains specifically the behavior of classical operations pop and push in the ADT stack [1]. Basically, a pop operation removes the stack top whereas a push operation adds an element in the top of a stack. This is shown in Figure 12: the operation push(4) adds the element 4 as the top of a stack whereas the operation pop() removes the top of a stack.

Traditionally a stack is implemented via pointers as its size increases and decreases in time execution. In the solution proposed by CGRAPHIC a stack is implemented as a linked list where each node is defined as a classical structure containing an element

---

**Figure 10** State after executing the sequence push(5)-push(4). [Color figure can viewed in the online issue, which is available at www.interscience.wiley.com.]
of the stack and a pointer to the following element. The main function of the C code (i.e., main) consists of a simple sequence of push-pop operations. The operations pop and push are implemented as independent functions whose codes should be clear for any student that had followed the previous topics proposed by CGRAPHIC.

During the execution, the student controls the state transition and can analyze graphically both the allocation (i.e., creation of new nodes) and deallocation (i.e., elimination of nodes) of memory as well as the modification of pointers. Figures 10 and 11 show two intermediate states resulting from the unhurried execution of this exercise.

**CONCLUSIONS**

We have presented CGRAPHIC, an original educational software designed exclusively for learning, via the programming language C, to program. This software is a virtual tutor that provides to the student both a theoretical and practical learning of the foundations of programming inside a graphical environment that can be executed on any navigator program.

The current version of CGRAPHIC covers the main topics of a classical program in a first-year course in engineering studies and holds the main
characteristics of the educational software [8] such as originality (to our knowledge), ease of use, capacity to motivate the student, treatment of relevant issues, versatility, availability, and interactivity with the student.

The English version of CGRAPHIC is totally operative and accessible in the following web address: http://campusvirtual.uma.es/fundinfo/online/English_CGRAPHIC/default.htm. There also exist a Spanish version of CGRAPHIC that is currently being used at the University of Málaga in the following subjects: “Foundations of Computing” of the school of engineering, and “Elements of Programming”, “Methodology of Programming,” and “Practical of Programming” in the school of computer science. Also, this Spanish version is being used as virtual tutor of a programming course in the Computing Virtual Services of Málaga University.

It should be noted that CGRAPHIC complements the traditional methods of teaching and not replace them. We believe that CGRAPHIC reduces the existing gap between the assimilation of theoretical concepts in programming and its use in practice. Moreover, it should also be noted that this kind of educational software intended to teach programming is worth as it is not only valid as a complement to programming teaching but also it is the first step to teach the foundations on which the educational software in general has to be implemented.

**REFERENCES**


**BIOGRAPHY**

**Antonio J. Fernández** received his BS and MS degrees in computer science from the University of Málaga in 1991 and 1995, respectively. In 2002, he obtained his PhD degree from the same university under the supervision of Dr. Patricia M. Hill of Leeds University, where he spent long periods of time during four years. He has worked, as manager of data computing, for several private companies in fields as diverse as banking, television, and computer manufacturing. He is currently an associate professor in the School of Computer Science at the University of Málaga, where he teaches foundations and practicals of programming. His area of research is constraint declarative programming, constraint satisfaction, and, more recently, the hybridization of constraint filtering algorithms and genetic algorithms for solving optimization problems. He is also involved in developing educational software for engineering courses and has published several papers on this topic.

**Jesús Millán** received the BS degree in computer science from the University of Málaga in 2002 and has completed his master’s thesis under the supervision of Dr. Antonio J. Fernández. He is currently working for private company in the area of computing and has recently been involved in the development of teaching software.