Block-C: A block-based visual environment for supporting the teaching of C programming language to novices

Kyfonidis Charalampos[1], Moumoutzis Nektarios[2], Christodoulakis Stavros[3]

[1] Computer & Information Sciences
University of Strathclyde
Glasgow G1 1XH, UK
{charalampos.kyfonidis@strath.ac.uk}c

[2, 3] School of Electronic & Computer Engineering
Technical University of Crete
Chania, Crete 73100, Greece
{stavros@ced.tuc.gr, nektar@ced.tuc.gr}

Abstract: Many barriers exist for novice programmers when confronted with the C programming language, such as its low level orientation, cryptic syntax and ambiguous compiler error messages. This paper presents the design and development of a block-based visual shell for the C programming language following the recognition over recall design pattern to eliminate syntax errors and enable the effective internalization of C programming language constructs. The evaluation studies provide evidence of the effectiveness of this shell, in tutorial/lab settings without the presence of human tutors.

Keywords: *C language, teaching programming, learning, block programming, visual programming, syntactic error prevention*

1. INTRODUCTION

Teaching introductory programming courses has received much attention the last years. This is mainly due to the ubiquitous use of computers, the proliferation of the so called cultures of participation [7], end-user programming [19] and end-user software engineering [14]. These trends are addressing software tools that provide powerful scripting languages to enable flexible customization and rich interactive content development by end-users. In this respect, knowledge of computer programming concepts is nowadays necessary for most knowledge workers including scientists, engineers and technologists. Consequently, many higher education departments have included introductory programming courses in their curricula. Furthermore, many countries[1,2] extend their curricula in secondary or even primary education to address the development of basic programming skills. The importance of computer programming has received even more attention recently through an international computer coding campaign titled "*The Hour of Code*"[3] within the context of *Computer Science Week*[4].

The introduction of programming courses in educational curricula is much facilitated by the proliferation of Educational Programming Languages (EPLs). Through their engaging graphical environments, EPLs promote situated learning by enabling the development of video games, computer animations and other digital artefacts. Many such languages are based on a Lego-like block-based paradigm[5]. Code is assembled from command blocks that are handled visually like Lego bricks. This approach addresses some of the most significant shortcomings related to the use of general purpose programming languages to introduce novices to the concepts of programming [22]. Although very effective, EPLs cannot be always used in introductory programming courses due to the fact that in many cases such courses require the use of a general purpose programming language.

The work reported in this paper addresses the above issues by proposing an approach to exploit the positive experience drawn from block-based EPLs. In particular, it presents *Block-*

[1] http://www.computingatschool.org.uk/data/uploads/internationalcomparisons-v5.pdf
[2] http://www.nextgenskills.com/israel-leads-the-way-on-computer-science-in-schools/
[3] http://www.theverge.com/2013/12/9/5191162/hour-of-code-computer-science-campaign-obama-cantor-support
[4] http://csedweek.org
[5] http://blog.acthompson.net/2012/12/programming-with-blocks.html

C, an engaging visual environment on top of *C* language. This environment enables novice programmers advance effectively in their first programming steps. It provides guidance and prevents usual coding errors (e.g. syntax errors) and thus increases the productivity of its users. Last but not least, it leverages the positive experiences drawn from EPLs to offer a Lego-like block-based graphical shell over the *C* programming language following the recognition over recall user interface design pattern.

The rest of this paper is structured as follows: Section 2 presents related work with respect to languages and platforms that enable effective learning of programming concepts. Section 3 addresses the design and implementation of *Block-C* exploiting the block-based programming paradigm. Evaluation studies that document the effectiveness of *Block-C* are presented in Section 4. Section 5 concludes and presents plans for futures work.

2. RELATED WORK

Teaching programming to novices has been a subject of research for many years. Beginning with Smalltalk [12] in 1972 a whole new discipline started to evolve. Over the years researchers tried to find more intuitive ways to guide novices over programming. Visual Computing offers a firm ground for such intuitive approaches.

Following a concise classification of Visual Computing software [20] there are two specific categories that are relevant to the work reported here: Visual Programming Languages (VPLs) and Visual Environments for Textual Languages (VETLs). Some of the most known examples of VPLs for educational use are Scratch[21], Etoys[11], and StarLogo TNG [13]. Alice [20], Greenfoot [10], Java2Sequence [1] and many others are examples of VETLs, offering graphical shells over general purpose programming languages (mainly Java).

Many studies [5, 9, 10, 15, 22] have been conducted for VPLs and VETLs in order to support novices in computer programming, such as the ones listed above. These studies aim to investigate the effectiveness of the corresponding VPLs and VETLs in terms of aiding and guiding learning. Furthermore, there has been research on more specific issues such as the impact of syntax error in learning computer programming [3, 4] as well as more student/user-centric studies [2, 18].

The first general purpose VETL that supports the full range of capabilities of the underlying programming language, is HyperPascal [17]. One of the most complete works in the field and similar to *Block-C* is *blockly* [8]. *Blockly* is an online block-based editor which produces Javascript, python and XML code. Also some pre-made block-based games are hosted in the *blockly* project, such as a modern pre-programmed “turtle graphics” design platform resembling Logo [6]. To the best of our knowledge, there is no such approach reported in the literature for the *C* programming language, which is the focus of our work.

3. DESIGN AND IMPLEMENTATION

Block-C is a general purpose VETL designed for making more effective the learning of the C programming language for novices. The major aspects of this work is the elimination of syntax errors by providing a block-based interface that promotes recognition over recall. *Block-C* specifically targets novices that are introduced to programming through the C programming language. Its aim is to facilitate their first steps in programming while at the same time enable the internalization of core programming concepts and C language structures so that after successful introduction to programming, the novices can proceed with ordinary plain-text C code. In this respect, *Block-C* is to be considered as a transient tool to support novices. This transiency is in many aspects closely related to *scaffolding* [23] as defined and used in the educational sciences. Scaffolding refers to learning situations where learners are gradually constructing their knowledge on their own through the facilitation of a more knowledgeable peer. The facilitator provides only hints and prompts which are later removed when the learners develop a certain level of autonomy and self-motivation. *Block-C* is especially useful in cases when human facilitators cannot be physically or virtually present meaning that scaffolding could occur through a technological intervention that will mitigate the absence of a knowledgeable human peer.

Block-C is implemented in Java as an extension of OpenBlocks [22], which is an open source JAVA framework released by STEP laboratory of MIT[6]. OpenBlocks could be used to

[6] http://education.mit.edu/openblocks

build VPLs dynamically by describing the properties of the blocks and the graphic environment in an XML file.

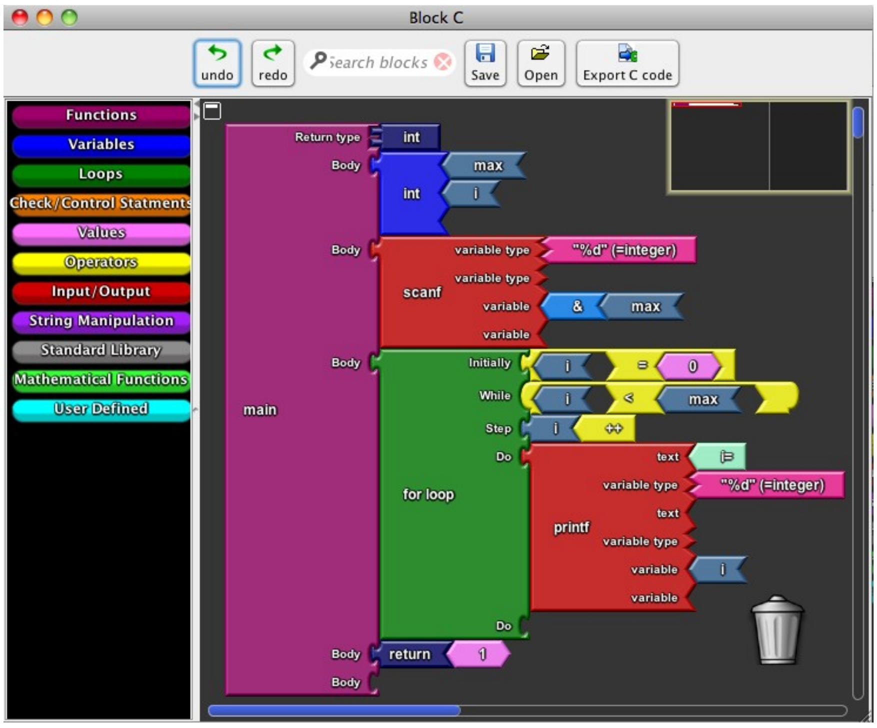
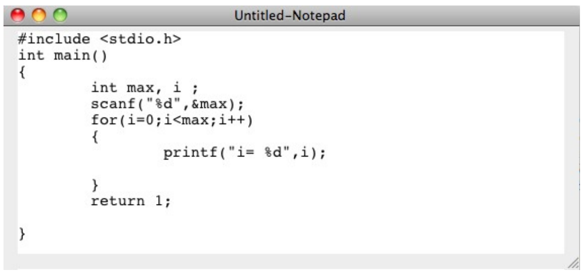


Figure 1: A simple program in *Block-C*, and its exported code in C.

Figure 1 above *presents* an overview of the user interface offered to the users of Block-C while Figure 2 below shows the corresponding plain-text C code for the program depicted above.



```
#include <stdio.h>
int main()
{
        int max, i ;
        scanf("%d",&max);
        for(i=0;i<max;i++)
        {
                printf("i= %d",i);

        }
        return 1;

}
```

Figure 2: A simple program in *Block-C*, and its exported code in C.

3.1 DESIGN DECISIONS

Block-C aims at providing learning efficiency and usability through a comprehensible interface that helps and guides learners. Therefore, the recognition over recall user interface design pattern [16] was adopted and the following design decisions were made:

- **Interaction with understandable notions only** - Learners should be relieved from notions that are not directly needed or used by them. Learners should not be expected to manipulate readily resources or tokens which cannot be easily understood such as libraries and the corresponding *#include* pre-compiler directive.
- **Syntactic error prevention** - To overcome the learning barrier of syntactic errors, syntax guidance should be given for the formation of correct *C* language expressions in a way that eliminates such errors. This is accomplished by the use of visual blocks that make explicit the structure of language expressions. Furthermore, to reduce the

complexity of the syntax of *C*, a restricted grammar is used in *Block-C* incorporating all structures that are necessary to introduce a novice to *C* programming.

- **Beautified code -** Guide learners to writing well-structured code. To this end automatically completion of parentheses, brackets, and code block limits are important aspects that should be supported. This is accomplished again through the use of graphical blocks. These blocks provide direct support for beautified code, balanced parenthesis and brackets as depicted in Figure 1.
- **Transitivity of the tool -** Enable learners to gradually move from using *Block-C* to using ordinary plain-text *C* programming. To this end, *Block-C* translates block-based code to *C* plain-text code and provides both viewing options. This way, it promotes the understanding of the correspondence between the command blocks and the actual *C* statements and their syntax (see Figure 2).

3.2 OPENBLOCKS EXTENSION

The core functionality of OpenBlocks could not address all usability requirements set at the design phase. Furthermore, several suggestions were given by test users during the first evaluation cycle (see next section for details). Most of them called for enhancements in the core functionality of OpenBlocks. Therefore, a number of OpenBlocks extensions were implemented (blocks' copying, colour coding, user-defined blocks, plain C translation, etc.).

4. EVALUATION

Block-C was initially evaluated through a usability evaluation process. This process led to the refinement of the intermediate versions. In order to measure the effectiveness of the final version of the tool in a tutorial/lab setting, a controlled experiment was conducted. In total 32 first-year volunteer students participated, all of them following the introductory programming course in *C* at the department of Production Engineering and Management of the Technical University of Crete. The distribution of the participants in the control (*C*) and test group (*Block-C*) was random; 16 students were assigned to each group. Both groups were asked accomplish three tasks of increased difficulty over a two-hours tutorial.

Table 1: Participants' profiles (programming experience and gender)

	None	Limited	Moderate	Sum	Percentage
Male	6	12	1	19	60%
Female	4	9	0	13	40%
Sum	10	21	1	32	100%

Those three tasks were chosen to cover the basic *C* programming concepts such as variables, conditional statements, boolean and arithmetic expressions, loops, input/output etc. Therefore, the first task required the implementation of a menu-driven program in which a number of if-else statements were required as well as the use of a while-loop statement. The second task required two nested for-loops. The third task required nested loop structures and a non-trivial operator for conditional checks.

During the evaluation four tutors were present but their interventions were restricted only to significant logical errors. No further assistance was given. The participants were asked to accomplish three tasks of increased difficulty in two hours. Statistics about the tasks accomplished and the time needed for each participants were recorded. The aggregated results which correspond to the number of users finished each task, are shown in *Table 2*.

Table 2: Number of users who finished each task with C and *Block-C*

Task	Method	Users Finished	Sum
1	Block-C	16	28
	C	12	
2	Block-C	9	14
	C	5	
3	Block-C	6	8
	C	2	

The total number of task finished with *Block–C*, in the same time duration, was significantly higher than with *C* (p=0.039). *Table 3* presents the improvement in the number of students that finished each task.

Table 3: Improvement of *Block-C* over C (based on the number of users who finished each task).

Task	1	2	3
Improvement	**x1.33**	**x1.8**	**x3**

According to these results, *Block-C* can be arguably considered as an effective tool that increases productivity of students in tutorial/lab settings by disencumbering them from the burden of syntactic mistakes. This fact is very important for the learning process of novice programmers because it enables them to focus on the logic part of their programs, rather than on the syntactic idiosyncrasies of the C programming language. As already stated, the tutors did not help learners with syntactic issues, a fact that reveals the effectiveness of *Block-C* in supporting self-study of novices without the presence of human facilitator.

5. CONCLUSIONS - FUTURE WORK

Educational programming languages are proven to offer an engaging learning environment for novices. *Block−C* aims to transfer this positive perspective in introductory *C* language courses. Employing the recognition over recall design pattern, *Block−C* succeeds in eliminating syntax errors and provide an engaging graphical shell over C. Its experimental evaluation demonstrates clear evidence of increased productivity, a fact that is of high significance when human facilitators are not present or unable to serve in a personalized manner a large number of students.

Following the promising evaluation results reported in this paper, further work is to be done to investigate the long term effects of using *Block−C* throughout a *C* programming course. Future plans include the integration of an execution panel and a graphical debugger directly in *Block−C*, as well. Step-by-step code execution through the debugger is expected to provide a deeper and more intuitive understanding of programs and offer an engaging demonstration tool for course lectures as well. Finally, a web-based version is to be considered exploiting the blockly project [Fraser et al. 2011].

ACKNOWLEDGMENTS

The authors would like to thank all students that participated in the experiments, the teacher and postgraduate students of the department of Production Engineering and Management of the Technical University of Crete. Especially the authors would like to thank Ms. Georgia Kyriakaki for the valuable help she provided during and after the experiment.

This work was partially funded in the scope of the following projects: pSkills (502843-2009-LLP-GR-COMENIUS-CMP), ALICE (518106-LLP-1-2011-1-IT-GRUNDTVIG-GMP). Block-C has been used to enhance learning interventions within the context of the Open Discovery Space project - http://portal.opendiscoveryspace.eu (FP7-ICT-PSP: 297229).

REFERENCES

[1] Barros J. P., Biscaia L., and Vitória M. Java2Sequence: a tool for the visualization of object-oriented programs in introductory programming. *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. *2011.* pp 369-369.

[2] Cardell-Oliver R., and Doran Wu P. UWA Java tools: harnessing software metrics to support novice programmers. *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. *2011.* pp 341-341.

[3] Denny P., Luxton-Reilly A., and Tempero E. All syntax errors are not equal. *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*. *2012.* pp 75-80.

[4] Denny P., Luxton-Reilly A., Tempero E., and Hendrickx J. Understanding the syntax barrier for novices. *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. *2011.* pp 208-212.

[5] Fesakis G., and Serafeim K. Influence of the familiarization with scratch on future teachers' opinions and attitudes about programming and ICT in education. *ACM SIGCSE Bulletin*. vol. 41. *2009.* pp 258-262.

[6] Feurzeig W. "Programming-Languages as a Conceptual Framework for Teaching Mathematics. Final Report on the First Fifteen Months of the LOGO Project". *1969*.

[7] Fischer G. "Understanding, fostering, and supporting cultures of participation". *interactions*. vol. 18. *2011*. pp 42-53.

[8] Fraser N. Blockly: A visual programming editor. Published. Google, Place. 2013.

[9] Garlick R., and Cankaya E. C. Using Alice in CS1: a quantitative experiment. *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*. *2010.* pp 165-168.

[10] Henriksen P., and Kölling M. Greenfoot: combining object visualization with interaction. *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. *2004.* pp 73-82.

[11] Kay A. "Squeak etoys, children & learning". *online article*. vol. 2006. *2005*.

[12] Kay A. C. The early history of Smalltalk. *History of programming languages---II*. *1996.* pp 511-598.

[13] Klopfer E., Scheintaub H., Huang W., and Wendel D. (2009) StarLogo TNG, In *Artificial Life Models in Software*, pp 151-182, Springer.

[14] Ko A. J., Abraham R., Beckwith L., Blackwell A., Burnett M., Erwig M., Scaffidi C., Lawrance J., Lieberman H., and Myers B. "The state of the art in end-user software engineering". *ACM Computing Surveys (CSUR)*. vol. 43. *2011*. p 21.

[15] Kölling M. "Greenfoot: a highly graphical IDE for learning object-oriented programming". *ACM SIGCSE Bulletin*. vol. 40. *2008*. pp 327-327.

[16] Lidwell W., Holden K., and Butler J. *Universal principles of design, revised and updated: 125 ways to enhance usability, influence perception, increase appeal, make better design decisions, and teach through design*. Rockport Pub. (2010).

[17] Lyons P., Simmons C., and Apperley M. Hyperpascal: A visual language to model idea space. *Proceedings of the 13th New Zealand Computer Society Conference*. vol. 492. *1993.* p 508.

[18] Mohamed Shuhidan S., Hamilton M., and D'Souza D. Understanding novice programmer difficulties via guided learning. *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. *2011.* pp 213-217.

[19] Myers B. A., Ko A. J., and Burnett M. M. Invited research overview: end-user programming. *CHI'06 extended abstracts on Human factors in computing systems*. *2006.* pp 75-80.

[20] Pausch R., Burnette T., Conway M., DeLine R., and Gossweiler R. "Alice: A rapid prototyping system for virtual reality". *Course Notes*. vol. 2. *1994*.

[21] Resnick M., Maloney J., Monroy-Hernández A., Rusk N., Eastmond E., Brennan K., Millner A., Rosenbaum E., Silver J., and Silverman B. "Scratch: programming for all". *Communications of the ACM*. vol. 52. *2009*. pp 60-67.

[22] Roque R. V. "OpenBlocks: an extendable framework for graphical block programming systems". *2007*.

[23] Sawyer R. K. *The Cambridge handbook of the learning sciences*. Cambridge University Press. (2005).