LittleC.js: A Lightweight, Minimal, Extensible, and Embeddable JavaScript Implementation of the C Programming Language

Stefano Federici
University of Cagliari, Cagliari, Italy

***Abstract*—Visual block languages have introduced new ways of learning computer languages. New Integrated development environments (IDEs) for standard programming languages such as C or SQL have been developed, derived from block languages and based on the metaphor of building blocks. New online IDEs, mostly used in online courses, have then made programming with standard, text-based programming languages such as C or Java, as easy as possible. Nonetheless, a gap still exists between learning computer programming in a Visual environment based on the block metaphor and a standard environment for a text-based programming language. In this paper, we propose a lightweight integrated development environment, developed for an Introductory Computer Programming course at the Faculty of Engineering of Cagliari, which can be used to gradually introduce students to the C programming language. The tool can be easily embedded in online resources that can also be accessed via mobile devices.**

I. INTRODUCTION

Computer programming is not a trivial task. This is clearly demonstrated by the large number of dropouts in university courses majoring in computer science. In order to make this task less painful, many efforts have been done. The availability of online, integrated environments where programs can be created, run and compiled on the web has had a great surge in the last decade. Therefore, when designing a course on computer programming, students can operate on a development environment that is easily accessible from everywhere and in which they can interactively assess the progress or their learning. However, if using online environments for standard programming languages is a good choice for advanced students, it can be of little help for introductory courses on computer programming. Learning what is behind even the simplest C program and then being able to write it and make it work without much struggling is beyond the wills of many university students (Fig. 1).

```
#include <stdio.h>
int main() {
   printf("Hello, world!\n");
   return;
}
```

Figure 1. The simple "Hello, world" program written in C.

To overcome the problems associated to learning a programming language, several excellent educational development environments have been designed in order to introduce students to the main concepts of computer programming by means of block languages such as Scratch (Resnick et al, 2009) or BYOB (Harvey & Monig, 100) (Fig. 2).

Figure 2. "Hello, world!" program built in Scratch.

Indeed, by using a block language, students do not have to learn and use unnecessary contextual elements that they do not immediately understand and can completely forget about syntactic errors -something that they feel as being frustrating and that causes many dropouts in computer programming courses- as they have just to snap together a few meaningful blocks. In order to make the transition to a standard programming language as smooth as possible, these environments (in particular BYOB) have then been extended in order to allow to run -meaningful subsets of- standard programming languages, such as C (Federici, 2009), in form of a block language (Fig. 3).

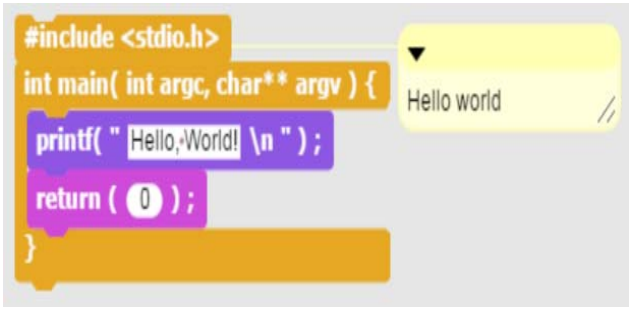

Figure 3. "Hello, world!" written in C as a block language.

Furthermore, to make these environments widespread, they have been recently redesigned by allowing them to run inside standard web browsers (Budger, 2014; Garcia et al. 2012; Federici & Gola, 2015). By means of these new

online versions, users can now avoid all problems such as downloading, installing and setting up the environments. For this very reason those newer environments can today be fruitfully used in online courses on introductory computer programming. Nevertheless, their usage is still somewhat limited in scope, as they do not fully cover all the needs of new learners of computer programming.

II. From Block Languages To Text Based Environments

Indeed, after the first introductory steps, students, and especially university students, must finally cope with standard text-based development environments.

```
#include <stdio.h>
int main() {
   int n, i;

   scanf("%d", &n);
   i = 1;
   while (i<=n) {
      printf("%d", i);
      i = i + 1;
   }
  return;
}
```

Figure 4. C program that writes the first n integers.

```
scanf(n);
i = 1;
while (i<=n) {
   printf(i);
   i = i + 1;
}
```

Figure 5. Simplified C program that writes the first n integers.

Hopefully, after having grasped the basic concepts of computer programming by means of a special purpose block language such as Scratch or Snap, and having been hopefully introduced to standard languages in a smoother way by means of the block version of those languages supported by BloP (Federici & Gola, 2015), they still have to move to real environments for standard programming languages. What they have to learn at this stage is mainly the ability to write syntactically correct programs, something that, as we have seen, using block languages can be instead completely ignored.

Facing this problem in a standard development environment is not easy. Therefore, usually teachers prefer to gradually introduce the elements of the language. For example, in the Introductory Computer Programming course at the Faculty of Engineering of the University of Cagliari, instead of starting by introducing libraries, variable declaration, and complex I/O functions of the C programming language, teachers introduce these concepts at a later stage, after having introduced and discussed at length the general syntax of the language. Therefore, instead of writing a program what fully works in a standard compiler or interpreter (Fig. 4), they start with a simplified version of the program (Fig. 5).

III. Integrated Development Environments For Online Course

Excellent online development environments are available for a low monthly fee, such as for example http://compilr.com. Other environments, with less features, are available for free (http://codepad.org, http://www.tutorialspoint.com/codingground.htm).

Usually these environments are based on web interfaces that have added to standard compilers or interpreters running on a webserver. By using those environments, designer of computer programming courses -and especially of online courses or courses equipped of online resources- must constantly refer to external resources and are then not able to embed interactive content inside their resources. Moreover, they must use the IDE as it is, without the possibility of adding a step-wise process that will make students able to gradually learn the syntax of the language.

A. Embeddable IDEs and Languages

Several very nice lightweight tools have been designed in order to build embeddable interactive content inside online courses about Python, Java, and JavaScript (Guo, 2013). These tools, running on a server, can be embedded inside online course modules and can be also customized and adapted by course designers to their own needs. An important added bonus of online courses based on these tools is their ability to run inside portable devices such as smartphones, so that students can follow their courses even when they are not at their PC. Some of these tools, such as for example Brython, a JavaScript implementation of Python 3 (http://www.brython.info/), run inside the web browser so that they can be also used offline.

IV. Building A JavaScript Interpreter For The C Programming Language

What is apparently missing is now a similar lightweight and embeddable IDE for the C programming language. Indeed, due to its exceptional runtime efficiency, the C programming languages is one of the most used programming languages for computer science courses and in online courses.

Building a C interpreter running inside a web browser, in principle, is not a very complex task. Starting from a full-featured or minimal open source C interpreter (such as for example iGCC, http://www.artificialworlds.net/wiki/IGCC/IGCC, or Picoc, https://code.google.com/p/picoc/), it is possible to compile it to an optimized JavaScript code by using Emscripten (http://kripken.github.io/ emscripten-site/). This process, that has been used, for example, to create WebGL, the JavaScript version of the OpenGL library, is not desirable for several reasons. Indeed, the final code is much larger than it is needed for an introductory C interpreter. Just to give an example, compiling the simple “Hello, world!” C

program will result in several hundred lines of code (https://cs-263-emscripten. readthedocs.org/en/latest/ simpleexample.html). However, we aimed at building an extremely lightweight C compiler that could be easily embedded and run also in a mobile device. Moreover, the compiled code cannot be easily modified as it is produced by keeping in mind efficiency, not readability.

The minimal C interpreter we need does not intend to replace full featured interpreters. Indeed, its main purpose is that of being able to run inside a web browser -so to be used even offline- and to be easy to customize and maintain following the specific needs of the teacher that will be then able to progressively introduce the syntax of the C programming language.

In the following, I will describe the development of LittleC.js, a minimal, lightweight, "progressive", easily extensible, and embeddable implementation of the C programming language made in JavaScript. Furthermore, a minimal implementation of an IDE for this language will be described. LittleC.js has been designed in order to support, as an introductory language and IDE, the Introductory Computer Programming course at the Faculty of Engineering of the University of Cagliari.

V. LittleC.js: A Minimal Implementation Of C Running Inside Your Browser

In order not to reinvent all the necessary components from scratch -that is writing a basic recursive descent parser and adding the necessary libraries- the development of LittleC has started from an existing minimal implementation of the C interpreter written in ANSI C by Schildt (1989) with the following features:

- Parameterized functions with local variables
- Recursion
- *if-else* statement
- *do-while*, *while*, and *for* loops
- Integer and character variables
- Global variables
- Integer and character constants
- String constants (limited implementation)
- *return* statement, both with and without a value
- A limited set of standard library functions.
- Several operators: +, -, *, /, %, <, >, <=, >=, ==, !=, unary -, and unary +.
- Functions returning integers
- Comments

```
for(a=0; a<10; a=a+1)
   for(b=0; b<10; b=b+1)
      for(c=0; c<10; c=c+1)
            puts("hi");
```

Figure 6. C program that cannot be interpreted by LittleC.

```
for(a=0; a<10; a=a+1) {
    for(b=0; b<10; b=b+1) {
       for(c=0; c<10; c=c+1) {
          puts("hi");
       }
    }
}
```

Figure 7. C program that is correctly interpreted by LittleC.

There are some small limitations. To give an example, in order to correctly interpret the code shown in Figure 6, it will have to be rewritten as shown in Figure 7 so that the targets of the if, while, do, and for are always "blocks" of code surrounded by beginning and ending curly braces.

The original code is divided in three files: recursive descent parser, C interpreter, and library of functions. The code has been manually converted to a single JavaScript file, by keeping the original structure and by adding the necessary auxiliary functions such as, for example, the C library functions isdigit, isalpha, and strchr; auxiliary functions to skip white spaces and to look ahead one position in the code stream. The final JavaScript code looks simpler and cleaner than the original C code as it is not based on pointers and structures. The full code, with comments and spaces, is about 31KBs. Once minified, the whole code is less than 13KBs.

Due to the lack of proper scanf and printf functions, necessary for the introductory course for which LittleC has been designed (the original interpreter only has a simple print functions without format parameter), several additions to the original C interpreter have been done:

- #include directive (to include the stdio library)
- printf
- scanf
- exit

The JavaScript implementation of printf and scanf functions has been designed on the "Sprintf for JavaScript" code (http://www.diveintojavascript. com/projects/javascript-sprintf) by Alexandru Marasteanu.

In order to make the usage of LittleC.js even simpler than the original interpreter, several error messages have been added to the list of 17 messages of the original interpreter. Those messages alert the user when the stdio library is needed but has not been included, when fewer arguments have been passed to a function or when an unknown character has been encountered in the C source code.

VI. Adding An IDE

In order to make LittleC very simple to use, a basic online IDE has been created based on the IDE of the "Online JavaScript Interpreter" (http://math.chapman.edu/~jipsen/js/) by Peter Jepsen. The

editor window of the original IDE has been enhanced by using the CodeMirror JavaScript editor (http://codemirror.net/). By using CodeMirror the editor is able to show line numbers and to highlight the C syntax. With these two additions, the overall size of the code of the full IDE (HTML, CSS styles, editor, printf/scanf library, and interpreter) is about 360KB.

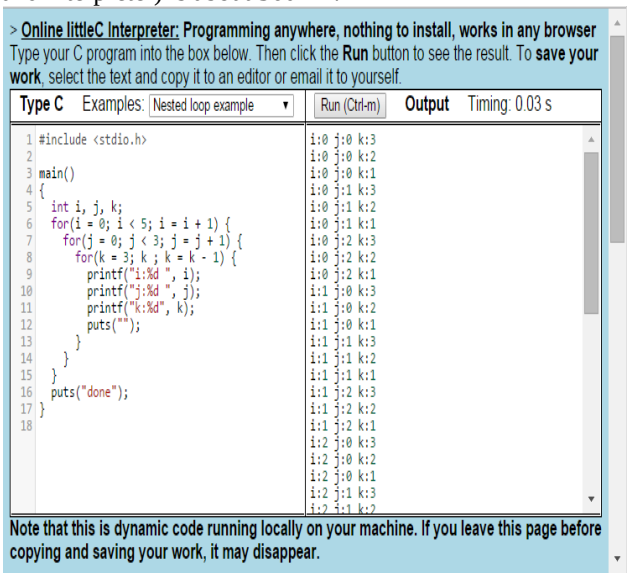

Figure 8. LittleC.js IDE: code editor and output area.

A. Using the LittleC.js IDE

The LittleC.js IDE is extremely simple (Fig. 8). At the left hand side there is the editor in which the user can type or paste the desired C code. Several sample codes are available by selecting them in the upper menu (Fig. 9).

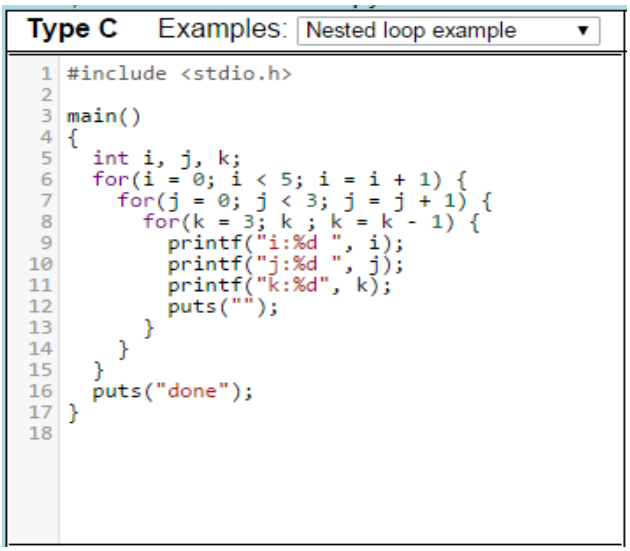

Figure 9. LittleC.js IDE: code editor.

The code is run by clicking the "Run" button available at the top of the right hand side area (Fig. 10). The result is shown in the *output area* below the button. If syntax errors (or some semantic errors, such as a missing include directive) are found, they are shown in the output area.

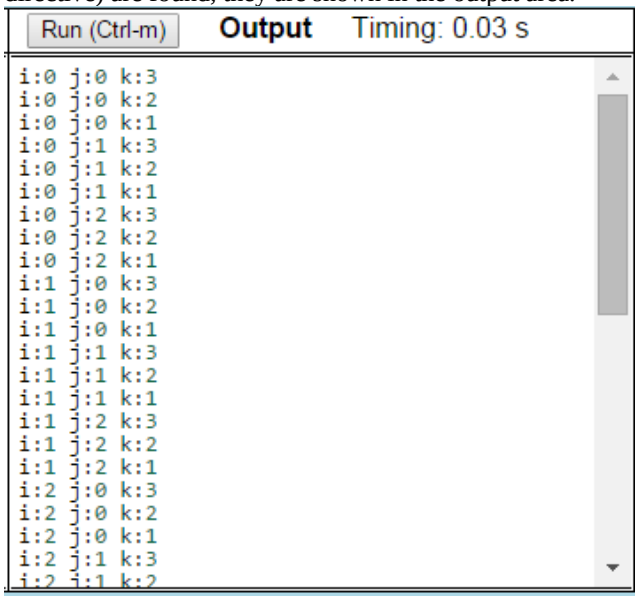

Figure 10. LittleC.js IDE: output area.

The code can then be modified or corrected and then run again. At the bottom of the page, the user has a short list showing the syntax of the C elements available in LittleC.js (Fig. 11).

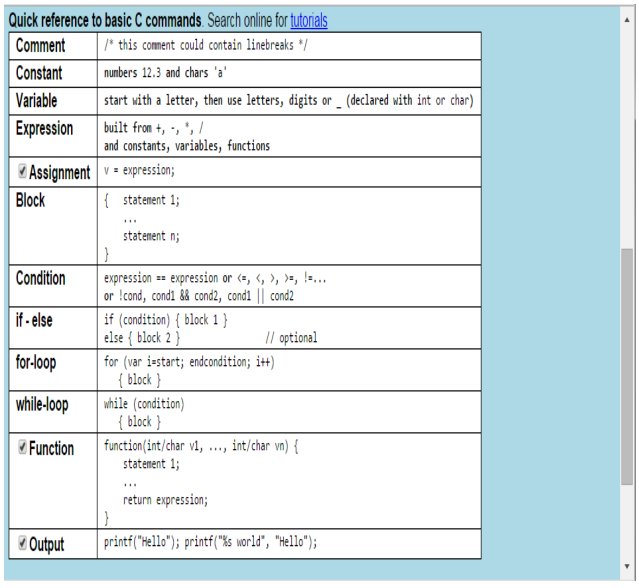

Figure 11. C elements available in littleC.js.

B. Using the Progressive Syntax of LittleC.js

As anticipated in the above sections, the C syntax illustrated in the introductory course of Computer Programming for which this interpreter has been developed, is gradually introduced to students. So, at the very beginning, variable declaration, main function, format argument of printf and scanf, and libraries are not introduced, in order to allow students to gradually experiment with C syntax without having to immediately understand concepts such as "library inclusion", "main function", "formatted output", "variable declaration". Indeed, the very same concepts are not included in the educational languages (such as Scratch or Snap) that they have used so far before starting to use LittleC. Therefore, to allow the teacher to illustrate progressively more complex C syntax, in the list of available C elements at the bottom of the LittleC IDE there are checkboxes for "Assignment", "Function" and "Output" (Fig. 12). Those checkboxes, when unchecked, do not raise errors if the users do not declare variables, do not use the "main" function, do not include libraries for input/output functions or do not use format arguments.

Moreover, if the teacher prefer not to leave the students the possibility to include/exclude the language features, a general flag can be changed inside the HTML page, so that the checkboxes are not shown and individual flags can instead be set inside the littleC.js code.

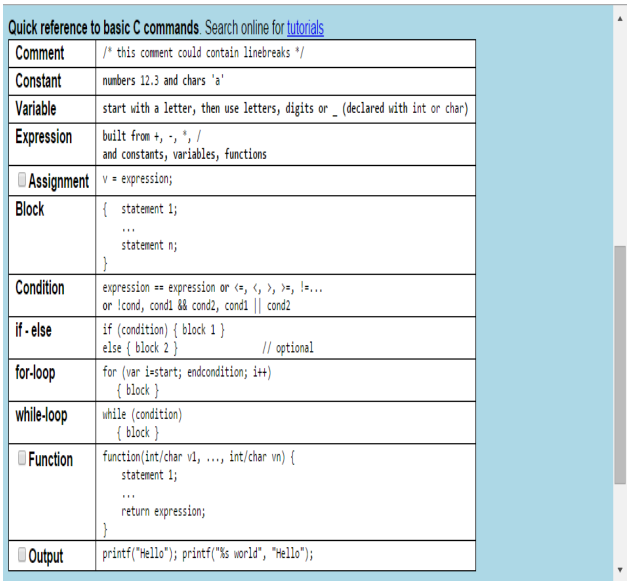

Figure 12. Selection of LittleC elements.

VII. CONCLUSIONS

Usage of LittleC.js for the introductory course of Computer Programming at the Faculty of Engineering of the University of Cagliari has shown that such a tool can be effective to allow students to interactively test their very first attempts in learning the syntax of the C programming language. The language implementation is clean and new features can be added very straightforwardly, as shown by the addition of the prinft/scanf and exit functions that were necessary for the course.

LittleC.js is extremely lightweight and it suits very well the limited capabilities of mobile devices such as smartphones by making it available to students always and everywhere.

LittleC.js and similar versions of other programming languages could fill the gap between block languages and standard text-based languages.

VIII. FURTHER WORK

In order to make LittleC.js even more useful for introductory computer science courses, several additional features (such as mono- and/or bi-dimensional arrays) could be added. Furthermore, the need to add curly braces around code blocks could be removed.

Finally, by adding some basic C++ elements such as cin, cout and passing parameters by reference, a C++ version of LittleC.js could be easily created.

IX. SOURCE CODE

LittleC is available at blocklanguages.org/littleC. The source code of the full environment can be downloaded by simply saving the webpage.

REFERENCES

[1] Badger, M. 2014. *Scratch 2.0 Beginner's Guide*. Packt Publishing; 2nd Revised edition (24 Mar. 2014).

[2] Beckford, C. & Mugisa, E. 2011. Towards Achieving an Ideal Environment for Teaching Programming Online. In *Proceedings of the International Conference on e-Business, e-Organization, e-Management and E-Learning* (Mumbai, India, January 28-29, 2011). IC4E 2011.

[3] Federici, S. 2011. A minimal, extensible, drag-and-drop implementation of the C programming language. In *Proceedings of the 2011 conference on Information technology education* (New York, USA, October 19-22, 2011). SIGITE 2011.

[4] Federici, S. & Gola, E. 2015. BloP: easy creation of Online Integrated Environments to learn custom and standard Programming Languages. In *Proceedings of SIREM-SiEL conference* (Perugia, Italy, November 13-15, 2014).

[5] Garcia, D., Harvey, B. & Segars, L. 2012. CS Principles pilot at University of California, Berkey. *ACM Inroads Magazine*. Volume 3, Issue 2, pp. 58-60. ACM, New York, NY, USA.

[6] Guo, P. J. 2013. Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education* (March 2013). SIGCSE '13. ACM New York, NY, USA.

[7] Harvey, B. & Monig, J. 2010. Bringing 'No Ceiling' to Scratch: Can One Language Serve Kids and Computer Scientists? In *Proceedings of Constructionism 2010* (Paris, France, August 16-21, 2010).

[8] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y., 2009. Scratch: Programming for All. *Communications of ACM*, 11

[9] Schildt, H. 1989. Building your own C interpreter. In *Dr.Dobb's Journal*. UBM Tech. Permalink=http://www.drdobbs.com/cpp/building-your-own-c-interpreter/184408184.