A Friendly Online C Compiler to Improve Programming Skills Based on Student Self-Assessment

RAQUEL CEDAZO CECILIA E. GARCIA CENA BASIL MOHAMMED AL-HADITHI

ABSTRACT: This paper presents an online C compiler designed so that students can program their practical assignments in Programming courses. What is really innovative is the self-assessment of the exercises based on black-box tests and train students' skill to test software. Moreover, this tool lets instructors, not only proposing and classifying practical exercises, but also evaluating automatically the efforts dedicated and the results obtained by the students. The system has been applied to the 1st-year students at the Industrial Engineering specialization at the Universidad Politecnica de Madrid. Results show that the students obtained better academic performance, reducing the failure rate in the practical exam considerably with respect to previous years, in addition that an anonymous survey proved that students are satisfied with the system because they get instant feedback about their programs.

Keywords: black-box testing; C programming language; E-learning; interactive learning environments; online self-assessment

INTRODUCTION

The programming learning is a complex task, especially for first-year students [1]. The literature contains many studies citing the poor performance of students after completing their first programming courses [2]. The programming learning is characterized by the large amount of exercises that students are expected to practice intensively in order to develop good programming skills [3]. In the field of the Industrial Engineering, this basic subject is considered extremely important since it offers the fundamentals of programming, totally required for learning more complicated software concepts in advanced courses.

Nowadays, one advantage to learn programming is that students have lots of compilers for any language. In the case of C language, there are some commercial ones (Microsoft Visual C++ is the most popular, with a powerful debugger). Others are open source ones (GCC under GNU/Linux or the multi-platform Dev-C++), and even online ones [4,5]. However, one disadvantage remains that these compilers cannot analyze the source code (possible bugs, dead code, overcomplicated expressions, etc.) or validate the acceptance criteria for students programs. Therefore, the support of the instructor is still required in the learning process of the students. Nevertheless, the assessment and feedback provided by instructors is not possible for courses with a large number of students, affecting negatively students' learning motivation [6].

Facing the impossibility of assessing larger groups of students, in many occasions the tendency is to reduce the number of individual homework assignments proposed to the students

[7], which is detrimental to learning. However, with the rapid advent of new technologies, computer-based assessment (CBA) is considered to be a fast and accurate tool for the assessment of students learning [8]. Therefore, in response to the challenge of giving support to students, many instructors have been using ICT effectively in higher education. Some of them have built collaborative assessment environments where students share their responses, exchange their ideas [9] and peer review programs written by other students [10]. Others have developed software solutions to assist teachers in their tutoring and assessing tasks. There are already many systems described in the literature that adopt this approach to offer an online environment for programming learning, that is, for Java [11,12], Assembly [13], VHDL [14], Web [15], Matlab [16] or C Languages [17], among others. These systems have their particular automatic evaluators, some of them are based on executing a set of previously designed and configured tests, comparing the output with the one generated by the instructor, which is called *black box evaluation* [18–20], and others systems even are focused on compiler error messages understanding [21] or on programming style analysis [22].

The authors teach the course of "Programming" to first-year students where they learn the techniques of structured programming in C language. The course is for all students at the Higher Technical School of Industrial Design and Engineering, Universidad Politecnica de Madrid (www.etsidi.upm.es), that is, students of different degree programmes; Chemical Engineering, Electrical Engineering and Electronics and Automatic Control Engineering. Each academic year there are more than 350 students in this course, divided into six groups for theoretical lessons, that is, around 55–70 students per group. Large class sizes make difficult the progress of each student, since there are not enough personnel resources to manually assess all the exercises during the course. This disadvantage, typical of large courses, is more apparent with these programming assignments, because the student requires support to consider the validity of their programs. The authors, motivated by the poor performance of students in large programming classes, recognized the need to develop an attractive tool, which could offer support for the student learning.

This paper presents a friendly and novel online compiler for C programming, which provides an editor for writing, editing, compiling and executing programming code, so the students can execute practical exercises online. It offers syntax highlighting in editing code, which helps students to remember the syntax better. In addition, the proposed system has other remarkable advantages compared to the traditional desktop tools:

1. Any user, both student and instructor, are able to access to their programming environment through any web browser, without installing any software or desktop IDE. This in turn offers them an underlying architecture completely transparent to users.
2. The compiler offers students a WIDE (Web Integrated Development Environment) in order to write and execute the C programs. This lets them to carry out on distance the practical assignments proposed by the instructors.

   (a) With respect to the other systems mentioned before, the innovative approach is that students are those who must design their own tests, acquiring the ability to test software [23] so important in a programming course.
   (b) It allows students to compare their output generated with the instructor-provided version (black-box testing). Through this feature, students self-assess their programs getting an instant feedback about their codes.

3. The compiler allows instructors to manage the programming exercises and assignments. The instructors organize all the resources through the web interface, grouping and classifying the exercises with different educational purposes.

   (a) The compiler lets instructors to propose extra complementary exercises along the course in order to strengthen students' theoretical concepts.
   (b) The application gives instructors indicators of the work carried out by the students.
   (c) The tool detects potential cases of plagiarism among students based on the students' logs, as it is explained in detail at Plagiarism section. Plagiarism is a serious and growing problem on the web educational environments. Detecting it is a useful feature for large courses because, otherwise, instructors would require much effort to manually find such possible cases. In fact, there are other authors which offer particular solutions to detect and prevent the plagiarism [24–27].

The paper is structured as follows: second section presents the educational environment where the proposed compiler has been launched. In third section, a detailed description of the laboratory architecture and the technology used is presented. Fourth section describes the full functionality for both students and instructors. Fifth section examines the characteristics that contribute to the effectiveness of the proposed online compiler in comparison with other existing ones. Sixth section examines the impact of the first use of this online C compiler during 2013/2014 academic years. And, finally, seventh section devotes to the conclusions that can be extracted from the proposed work and possible future improvements for the application.

CASE STUDY: PROGRAMMING COURSE IN INDUSTRIAL ENGINEERING

The proposed application has been applied to the first-year Programming course in three-degree programmes at the Industrial Engineering specialization at the Universidad Politecnica de Madrid. Along the second semester, this course, with a total of 6 ECTS (European Credit and Accumulation System)[1] credits, is divided into three parts: (1) theoretical and problem-solving lectures (2.4 ECTS) in 55–70 students/classroom along 15 weeks, of 4 h weekly, (2) 9 practical assignments (0.6 ECTS) in laboratory where there is a computer per student, and (3) private work at home and examinations (3 ECTS). This structure is summarized in Figure 1.

Regarding to the practical part, each student must carry out 9 practical assignments of 1 h during the course. Each group consists of one instructor for 20–24 students. Each practice has a script, which is previously given to the students, which consists of a set of 4–5 exercises to be programmed in the laboratory during this hour. The script has been designed so a student would be able to finish all exercises during the session. However, our experience is that the majority of students are not able to complete all the exercises on time. On the other hand, the instructor cannot supervise all students and give them individual feedback about all their codes. As a result of all these factors, a significant number of students fail the practical exam each year.

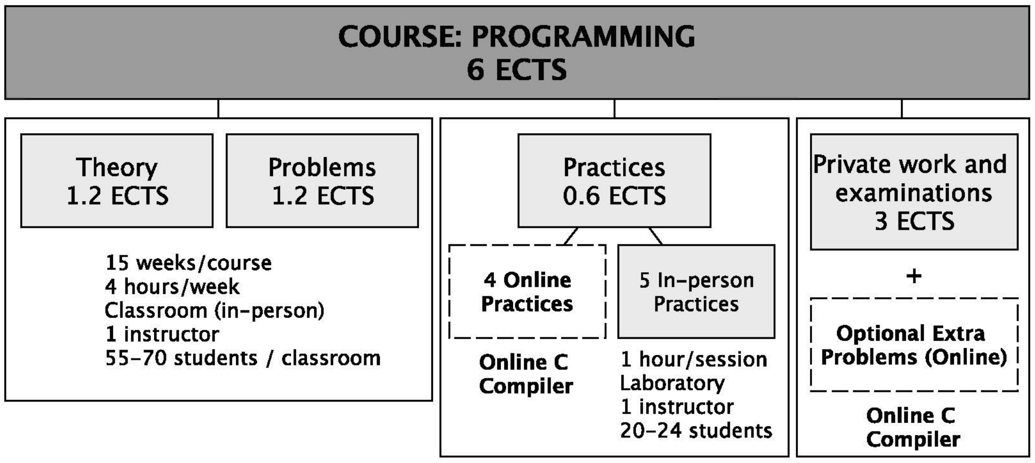


Figure 1 Programming course structure.

Since the inception of the Bologna Process at our School in 2010, there has been an tangible increment in enrollment with respect to the previous educational system. Since then, the maximum number of students enroll each year in our degree programmes. The high number of newcomers (250–260 students) imposes an overload to our department. This number must be added to those students who have repeated from previous courses, and that represents a substantial volume: 115 students (41.39% of the total) in 2011/2012, and 100 students in 2012/2013 (28.17% of the total). As statistics reflect, significant failure rates are shown, mainly in the Electrical Engineering Degree (34.45% is the average of all the previous academic years) and in the Chemical Engineering Degree (32.07%). However, the failure rate in the Electronics and Control Degree is considered within normal rates (19.23%), due to two factors worthy of mention: (1) their special predisposition for the programming, and (2) because of their appreciable higher access marks than the others two degrees.

In spite of the increase in incoming students, this has not been matched by a corresponding increase in instructors. Therefore, the ratio of students to instructors has been increased year after year, overloading the workload of the instructors with respect to the number of assignments and, consequently, the amount of assessments.

Motivated by the large failure rates mentioned before and the limited number of instructors, it was decided to introduce the system developed as an educational resource during the 2013/2014 course by first time. The instructors made several changes to the original curriculum, which are highlighted with dotted border in the Figure 1; (1) 4 online practices over the 9 in total were made available to perform them compulsorily through the online compiler proposed, and (2) a set of 40 extra exercises classified by different tags were published with the aim to serve as a complement to the private work at home. The results obtained during the pilot experience are detailed in Results section.

SOFTWARE ARCHITECTURE

Our department has a Weblabs Portal where remote laboratories of different subjects are integrated. Among them, Automatic Control, Digital Electronics, and Programming. Similar Internet-based networks of facilities are been used for practical learning activities in engineering education in the universities worldwide [28–31]. The web platform used is Liferay Portal Community Edition, distributed under free software license. This platform is the common point of access for students of any laboratory, and offers lots of advantages. Besides solving the User Management (accounts, permissions, profiles, roles), Liferay is a powerful tool that includes a large set of components for creating forums, blogs, and wikis, among others, similar to any CMS (Content Management System).

At the same time, Liferay lets an easy LDAP (Lightweight Directory Access Protocol) integration. In this way, the Weblabs Portal has been configured in order to connect to the University's central repository to authenticate users using the corporate LDAP. This is an advantage because it solves the authentication in every labs integrated into this Portal, allowing users to login with the same username and password that for the rest of the university services.

Technologies

This section describes briefly the set of technologies that have been used during the development phase, both server and client side (see Fig. 2).

On the other hand, Figure 3 shows a scheme of the architecture of the application. As the figure reflects, both students and instructors access to the C compiler through the Weblabs Portal using simply any web browser (Firefox, Chrome, Internet Explorer, Opera, etc.). The application runs in a GNU Linux computer, concretely in an Ubuntu Server, where the following software packages have been installed:

- **GCC Compiler**. GCC (GNU Compiler Collection) is an integrated distribution of compilers for several major programming languages: C, C++, Java, Fortran, and Ada, among others. The proposed application only uses the front end for C to compile the code and execute the programs.
- **Node.js**. It is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications, especially in the server side. It provides asynchronous events and uses an event-driven, non-blocking I/O model that makes it very light. Then, Node.js is especially suitable to execute many independent background processes in a non-blocking way.
- **ExpressJS**. It is a framework for Node.js is used to manage requests, responses and sessions, among other features.
- **Socket.IO**. It is a JavaScript library which allows the use of the WebSocket protocol. It has two parts; a client-side library that runs in the browser, and a server-side library for Node.js,

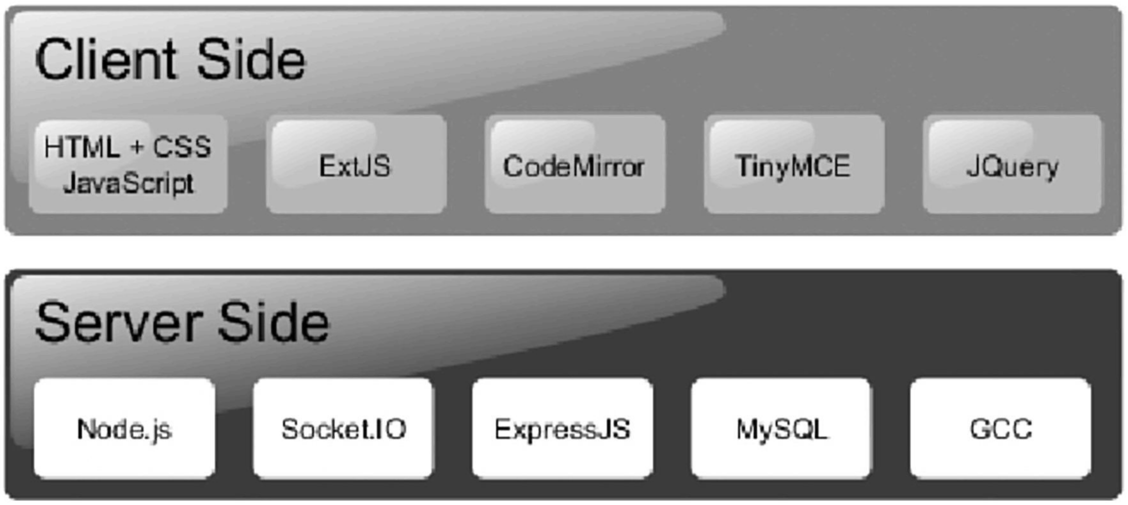


Figure 2 Layers of the technologies used in the application, both client and server sides.

providing full-duplex communications channels over a single TCP connection between server and client.

- **Ext JS 4**. It is currently one of the most powerful JavaScript web frameworks in the client side currently. It uses a model-view-controller (MVC) architecture and has hundreds of User Interface (UI) widgets with support for all major web browsers. Different widgets have been used in our application making it very attractive. For instance, ItemSelector with drag and drop, Grids with paging and sorting, etc.
- **TinyMCE**. It is a Javascript HTML WYSIWYG^2 editor. It allows users to write easily a rich content (text and graphics) through the web interface.
- **CodeMirror**. It is a versatile text editor implemented in JavaScript, specialized in editing code. It has been used to facilitate users to write the C programs since it includes features of any IDE, like automatic indentation and syntax highlighting.
- **jQuery**. It is a JavaScript library which includes features like HTML document traversal and manipulation, event handling, animation, and much simpler AJAX.
- **MySQL Server**. We have designed a relational model for storing all the persistent data of the application. MySQL is chosen as database server because it is considered the world's most popular open source database.

Finally, it should be stressed that all the software used is distributed under free software or for non-commercial use licenses. And, consequently, the source code developed of the application has been hosted as a free software project available at SourceForge website (https://sourceforge.net/p/compiladorc/).

DESCRIPTION OF THE APPLICATION

The system gives different levels of access to users, depending whether the user is *student* or *instructor*. The following sections describe in detail the whole system.

Exercises

Instructors are responsible for creating exercises using a friendly interface and they must specify the following contents associated to each exercise:

- A meaningful and clear **title** to describe the exercise.
- The descriptive **wording**, by means of a WYSIWYG editor, which indicates clearly the problem to be solved, that is, what is expected from the program to do.
- One or more **tags**, necessary to classify it (see more details at Tagging section).
- **Source code** with the solution of this exercise. The instructor writes the program directly in the online editor text and compile it. This code is stored into the server and will be used by students during their self-assessment process, as it will be described at Black-Box Testing section.

The interface lets instructors to manage the exercises and the practices, that is, they can add, edit, delete, publish/unpublish both exercises and practices. As it is shown in Figure 4, this interface allows the user to see different information: the name of instructor who created it, the date, the current status, and the list of the practices where the exercise is included.

Instructors have also the possibility to group exercises following some selection criterion, some different topics or several difficulty levels, etc. The set of these exercises is registered into the system as a "Practice." Therefore, an exercise could belong to zero, one or more practices, as it is indicated in the last column of the grid at Figure 4.

On the other hand, students can carry out both individual exercises and full practices. They can select exercises from a list or even search exercises by a specific tag (see next section). All the exercises carried out by a student are stored into an individual "Exercise Repository" to which they can access through the student interface.

Tagging. Since this application is able to store multitude of exercises, the authors think that there is a need for classifying exercises by categories or also called *tags*, so that the user would be able to search a particular type of exercise. This is useful if a student would like to strengthen some particular concept, like for instance loops, switch-case structure, pointers, etc.

There is a list of tags created and maintained by the instructors, who assign to each exercise as many tags as they want from the tag list.

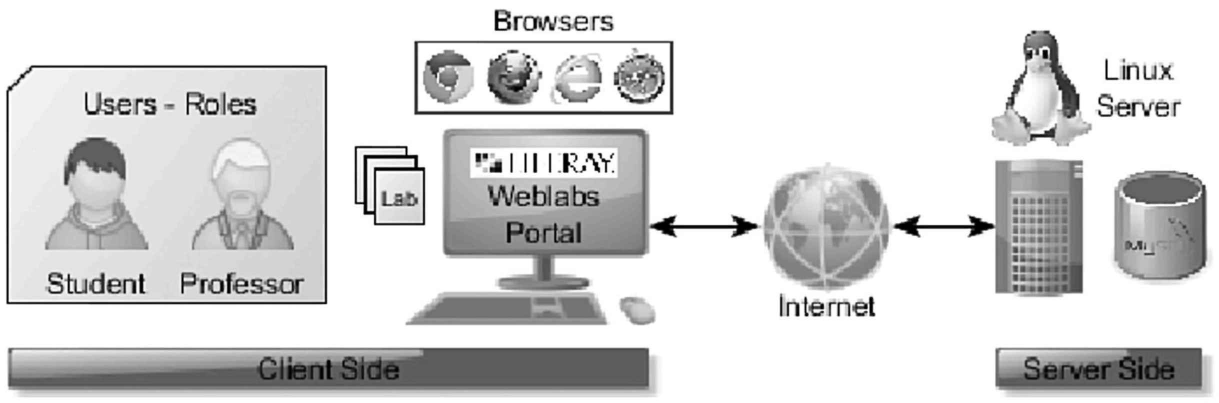


Figure 3 Overview of the architecture of the online C compiler.

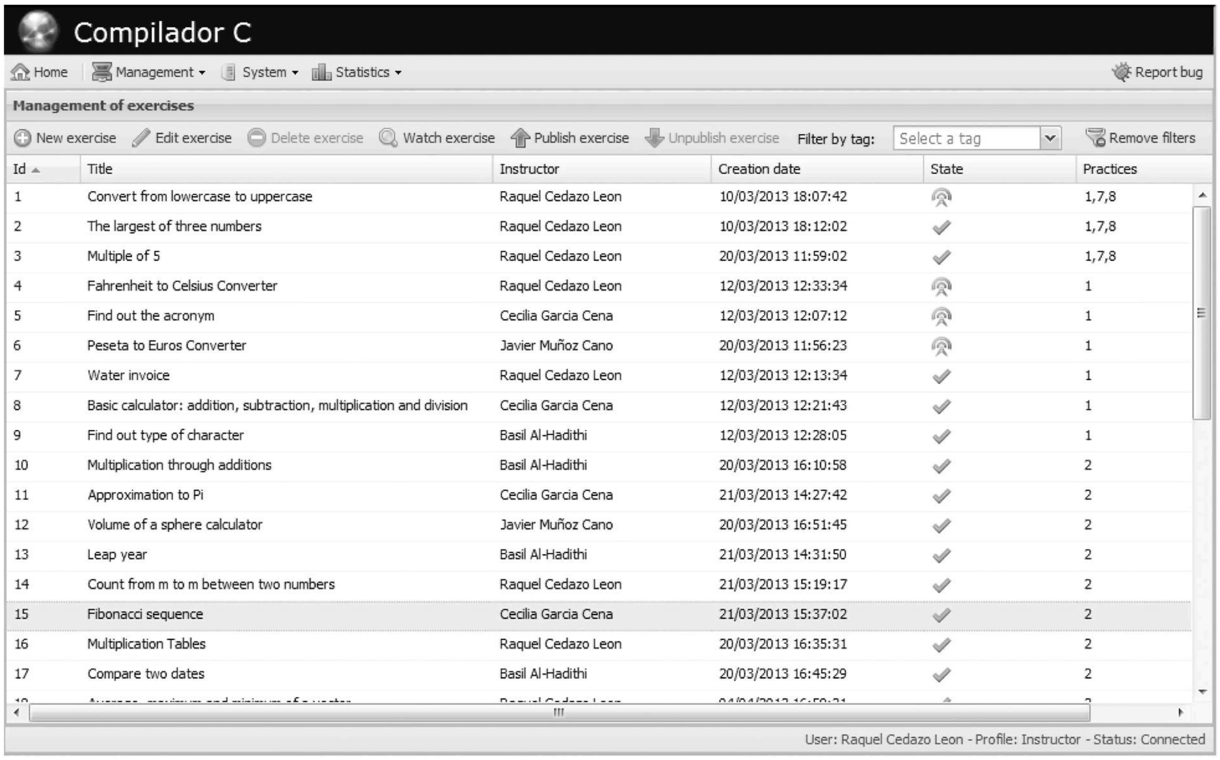
Self-Assessment Process

The exercises proposed are of short duration and moderate difficulty. These exercises are designed by the instructors so that the result is unique for the same input values. For example, possible exercises could be assigned as follows: (1) generation of *n* numbers of the Fibonacci sequence, (2) ordering in an ascending form the elements of a vector, (3) multiplication of two matrices. The majority of exercises requires interaction with the students through the standard input in order to introduce some data, and compulsorily prints the results to standard output.

As it has been mentioned before, the main and the most original contribution of this system is that students can self-assess their programs. In the next section it is explained in details how the proposed system works.

Black-Box Testing. The method used for the self-assessment is called "black-box testing" in computer science. It is a method of software testing that examines the functionality of an application without peering into its internal structures or workings. As it is shown at Figure 5, the student is aware of that a particular input returns a certain output but he/she is not aware of how the software produces the output. In fact, the program could have different codes, that is, a student could use a *switch-case structure* while another one would prefer an *if-else structure*. In case of loops, the student could use *while*, *for* or *do-while* statements.

The instructors' source code is inside the black box, so their programs are never shown to students. They only can see the output of the instructor's solution compared to their own output. Therefore, students are required to design their own tests to check the



Id	Title	Instructor	Creation date	State	Practices
1	Convert from lowercase to uppercase	Raquel Cedazo Leon	10/03/2013 18:07:42	◎	1,7,8
2	The largest of three numbers	Raquel Cedazo Leon	10/03/2013 18:12:02	✓	1,7,8
3	Multiple of 5	Raquel Cedazo Leon	20/03/2013 11:59:02	✓	1,7,8
4	Fahrenheit to Celsius Converter	Raquel Cedazo Leon	12/03/2013 12:33:34	◎	1
5	Find out the acronym	Cecilia Garcia Cena	12/03/2013 12:07:12	◎	1
6	Peseta to Euros Converter	Javier Muñoz Cano	20/03/2013 11:56:23	◎	1
7	Water invoice	Raquel Cedazo Leon	12/03/2013 12:13:34	✓	1
8	Basic calculator: addition, subtraction, multiplication and division	Cecilia Garcia Cena	12/03/2013 12:21:43	✓	1
9	Find out type of character	Basil Al-Hadithi	12/03/2013 12:28:05	✓	1
10	Multiplication through additions	Basil Al-Hadithi	20/03/2013 16:10:58	✓	2
11	Approximation to Pi	Cecilia Garcia Cena	21/03/2013 14:27:42	✓	2
12	Volume of a sphere calculator	Javier Muñoz Cano	20/03/2013 16:51:45	✓	2
13	Leap year	Basil Al-Hadithi	21/03/2013 14:31:50	✓	2
14	Count from m to m between two numbers	Raquel Cedazo Leon	21/03/2013 15:19:17	✓	2
15	Fibonacci sequence	Cecilia Garcia Cena	21/03/2013 15:37:02	✓	2
16	Multiplication Tables	Raquel Cedazo Leon	20/03/2013 16:35:31	✓	2
17	Compare two dates	Basil Al-Hadithi	20/03/2013 16:45:29	✓	2

User: Raquel Cedazo Leon - Profile: Instructor - Status: Connected

Figure 4 Screenshot of the instructor interface in order to manage the exercises.

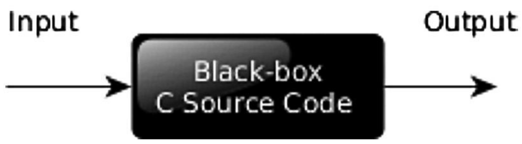


Figure 5 Black-box testing approach.

validity of the code. The authors choose this method mainly to force students to review their code as many times as necessary and to try to pass the tests again. It is considered that students will assimilate the concepts better which in turns improves the learning process. Moreover, students will be also capable of designing different test cases and achieving a good coverage with black-box tests, which means a high percentage of code checked.

The way that students have to check the correctness of their programs is to build their own test cases based on the specifications and requirements of the exercise. Thanks to the web interface developed, students are able to compare their solution with the instructor one. When students consider that their program fulfill all the test cases, then, they mark themselves the exercise as "Marked as solved" through a button in the interface (see Fig. 6). Then, instructors can see the exercises done by students.

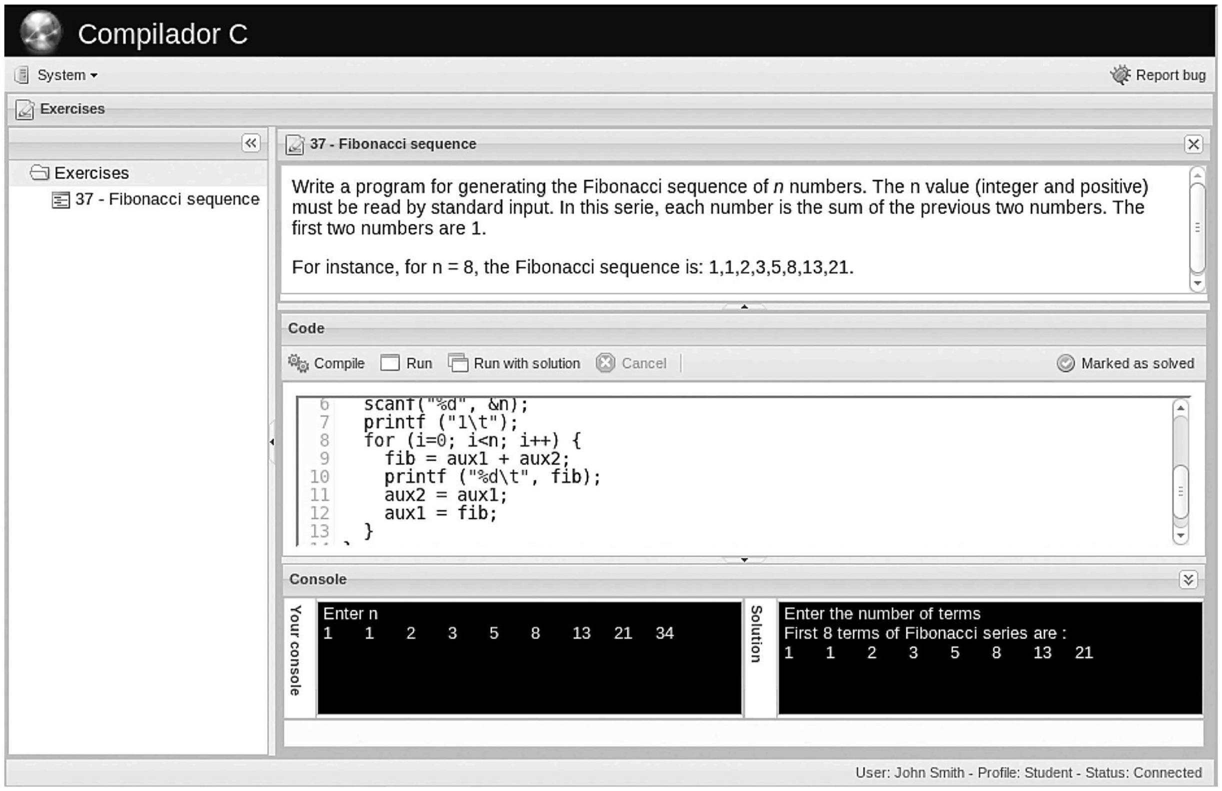
An example of how this process is implemented is shown in Figure 6. The interface is distributed in different areas: the wording on the top, the source code on the middle, the student output console on bottom left, the instructor output console on bottom right, and the input console on bottom in the yellow text area, which is the common console input for both student and instructor programs. For example, as it is shown in the wording of the exercise shown, the student has to generate the Fibonacci sequence of *n* numbers, being *n* an integer and positive number read by the standard input. The result indicates a wrong program since the output of the student program does not match the output of the instructor program, as it can be seen comparing the two output consoles. It is expected that the student will change the code and re-compiles it to get the proper output. This process, for different input values, will indicate to the student if the code is correct or not.

Statistics

All the interactions with the system, along with the exact time, are registered into the database in order to obtain information about the usage of the system and about the performance of the students. These statistics can be consulted by the instructors through the website. They include:

- Global information:
  - Total number of exercises marked as solved.
  - Total number of compilations, distinguishing the number of wrong and correct compilations as well.
  - Total number of executions.
- Statistics by exercise. It includes the number of times marked as solved, compilations, and executions.
- Statistics by practice. It includes the statistics of each exercise which is part of the practice.
- Statistics by student. Instructors can select a student and see the individual statistics about his/her work, giving the chance also of visualizing the source codes of the exercises, that is, the "Exercise Repository" of the student.

A proper interpretation of these statistics is fundamental for evaluating the students. For example: how many tests they have



Write a program for generating the Fibonacci sequence of *n* numbers. The n value (integer and positive) must be read by standard input. In this serie, each number is the sum of the previous two numbers. The first two numbers are 1.

For instance, for n = 8, the Fibonacci sequence is: 1,1,2,3,5,8,13,21.

```
scanf("%d", &n);
printf ("1\t");
for (i=0; i<n; i++) {
  fib = aux1 + aux2;
  printf ("%d\t", fib);
  aux2 = aux1;
  aux1 = fib;
}
```

```
Enter n
1    1    2    3    5    8    13   21   34
```

```
Enter the number of terms
First 8 terms of Fibonacci series are :
1    1    2    3    5    8    13   21
```

User: John Smith - Profile: Student - Status: Connected

Figure 6 Screenshot of the student interface in order to solve a programming exercise.

Table 1 Comparison Table of Online Environments for Learning Programming Languages

	Online editor	Syntax highlighting	Plagiarism detection	Battery of exercises for self-learning	Test cases	Code analyzer
The Online Jude [7]	×	×	✓	×	Automatic	×
JIST [11]	✓	×	×	✓	Automatic	✓
Mookshak [12]	×	×	×	×	Automatic	×
ALP Laboratory [13]	×	×	✓	×	Automatic	×
VHDL Laboratory [14]	×	×	×	×	Automatic	✓
Web Laboratory [15]	✓	✓	×	✓	Self-testing	×
Matlab Laboratory [16]	×	×	×	×	Automatic	✓
C Laboratory [17]	✓	×	×	×	Automatic	✓
C Online Compiler	✓	✓	✓	✓	**Self-testing**	×

done of each exercise, how many wrong compilations they have done before executing, the mean time for solving each exercise, and which are the easiest and most difficult exercises in general, among other useful information.

Plagiarism

In this online tool, any student could share his/her code with other students through the other external ways (e-mail, chat, pen-drive, etc.), and could be copied and pasted instantly onto other user account.

It is not easy to identify plagiarism automatically on distance courses. There are tools which detect plagiarism in programming based on the similarity of codes [32]. However, in this framework, it is not considered as a really decisive tool since the exercises are simple and the solutions are in most cases very similar. Therefore, the approach is different and consists of analyzing the exact time of starting, compiling, and executing a program, and see if the difference between times is significantly small. This could be a clue that the student has copied the exercise. If this behavior is repeated for different exercises, it can be considered as a real case of plagiarism. This has been the starting point set up in order to detect possible copies in the system.

During this first year of working, it was detected that 18 (6%) of the total of 301 students copied the majority of their programs, which reflects the need of including methods for detecting plagiarism cases.

Bug Reporting

Through the application itself, students can report errors and bugs, both technical (i.e., unexpected behavior at the application) and about the correctness of the exercises if the solution of the instructor is wrong. Users have a "Report bug" button accessible from the interface (see Fig. 6) and let users to fill out a form detailing the problem encountered, which is notified by e-mail to the webmaster. It is worth mentioning that the application was refined during the first weeks and some errors in the exercises were modified thanks to the bug reporting from the students.

THE EFFECTIVENESS OF THE PROPOSED C ONLINE COMPILER

In order to examine the validity of the developed compiler, in this section we will highlight the main advantages in comparison with other existent systems cited in the introduction.

Some of the advantages mentioned in the introduction about the system under study are also present in other many systems. Common benefits include immediate feedback, suitability for mass courses, 24-h availability, without time or place restrictions, and tracking of the students' progress based on the recorded information. Table 1 shows a comparison of the remarkable features among these systems and the proposed one:

- As it can be seen, some environments integrate an online editor (see column 1—Table 1), so students can program their exercises through the browser and deliver them directly. The rest of systems require a submission system to upload the files.
- Few environments give importance to the syntax highlighting. In fact, one system in addition to the proposed one (see column 2—Table 1) display the source code in different colors according to the category of the terms.
- Although the plagiarism is a common problem in the mass programming courses, it is not easy to find copy detection mechanisms embedded in all these systems, as it is reflected in the column 3.
- Unlike many other environments, the proposed system contains a battery of exercises freely available to the student (see column 4—Table 1). This has been designed in this way to serve as a self-learning environment, where students can select any exercise and program it as an extra work, apart from the compulsory practical assignments. In addition, some systems like our system classify and group the exercises by some criteria (category, tags, difficulty level, topic, etc.), which enable students to better customize their learning.
- Table 1 also compares these environments with respect to who and how to do the test cases. The evaluation of any code requires carrying out an exhaustive test set which should cover all the project specifications to assure the correctness of each assignment. In this aspect, it can be distinguished among those environments which integrate automatic evaluation tools, that is, the test cases are pre-defined by the instructors, and those where students are who must design their own test set in order to check whether programs have been implemented properly and work correctly. Our system belongs to the second type, less common, where students develop an important programming skill and it offers them a valuable experience for their future career.
- Finally, column 6 summarizes the systems that analyze the source codes submitted in order to check that certain requirements specified in the wording are properly met, as for example the usage of a *while* loop or other elements. This feature supposes a better feedback of the programming concepts.

As a conclusion, it has been clearly seen that our compiler outperforms the other system included in Table 1. Although, with respect to the "Code Analysis," this characteristic is planned as future work in next versions of the software (see Conclusions section).

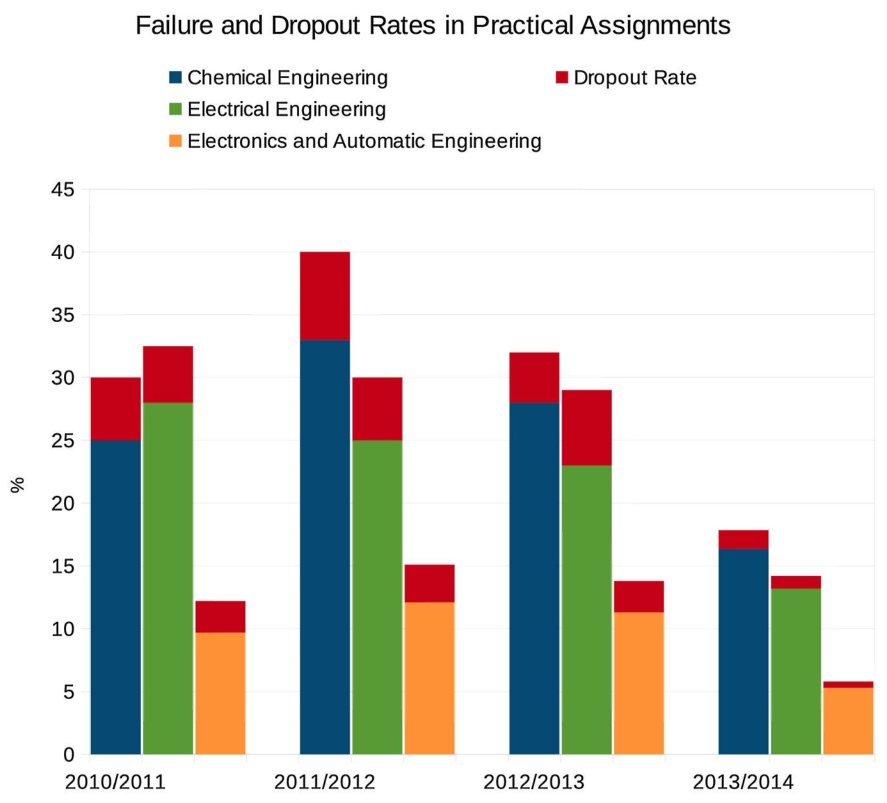


Figure 7 Percentages of failure and dropout rates in the practical assignments during the different academic years, separated by degree. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

RESULTS

In this section, the results of the case study presented at Case Study: Programming Course in Industrial Engineering section are exposed. A rigorous analysis was extracted from the database. The following results over a total of 301 students who carried out the practices were obtained during the 2013/2014 course:

- 256 students (85%) marked as solved all the exercises of the four practices. The rest of students only did some of the exercises proposed.
- A total of 35 students (11.62%) of all degrees failed the practical exam, a much smaller percentage than previous years: 20.9% in 2010/2011, 23.37% in 2011/2012, and 20.77% in 2012/2013. The Figure 7 shows the failure rates separated by degree. As it can be seen, the students of Chemical and Electrical Engineering are those with the highest failure rate, following the same trend for all academic years. The reasons of the difference with the students of Electronics and Automatic Control Engineering are exposed at Case Study: Programming Course in Industrial Engineering section. Notice that the statistics show a marked improvement in the last year thanks to the use of the system presented.
- Particularly important is that the dropout rate was greatly diminished, as it is shown at Figure 7.
- The students required an average of 13.3 executions until they marked the exercise as solved.
- Regarding to the extra exercises proposed, an appreciable percentage of 54% students did not participate. However, students who carried out some exercise, did an acceptable average of 9.4 extra exercises.
- 18 students (6%) copied the majority of their practices (as it was explained at Plagiarism section).

At the end of the practices, an obligatory online questionnaire was designed to measure students perception of the tool. The total of 301 students answered anonymously to 10 questions, within a scale from 0 (the minimum value) to 10 (the maximum value). The average of the responses are reflected in Table 2.

Table 2 Results of the Student Feedback Questionnaire During the 2013/2014 Course. A Total of 301 Students Answered the Questionnaire

Question	Average (Scale 0–10)
Q1. Do you think that the Online Compiler is an useful tool to learn programming?	8.8
Q2. *Before* using this tool, indicate what level of trust you had programming.	4.8
Q3. *After* using this tool, indicate what level of trust you had programming.	7.7
Q4. Do you consider useful the instructor's output console in order to check the correctness of your exercises?	8.0
Q5. Thanks to this tool, I have improved the "Problem Solving" competency.	7.4
Q6. Thanks to this tool, I have improved to "Test and verify software".	7.2
Q7. Indicate if it is a friendly interface.	9.2
Q8. Indicate if this Online Compiler could substitute the Desktop Compiler used during the course.	7.9
Q9. Would you recommend this tool to another student?	8.4
Q10. Assess the global level of satisfaction with the Online Compiler.	8.1

Analyzing the results, the first experience is considered as a successful one in two fold; firstly, due to the increase of the number of students who passed the practical exam and secondly because of the high satisfaction level shown by the students in the questionnaire. Concretely, through their answers (see Table 2), it can be concluded that students consider the Online Compiler as a helpful tool to learn programming, promoting also their trust and the skill for "Solving Problem" and "Testing and Verifying software." Even though, an important percentage of students think that the Online Compiler can be considered as a substitute of the Desktop Compiler. In this sense, the authors are aware that certain improvements are required to achieve a tool totally self-sufficient, which is explained in the next section.

CONCLUSIONS

An innovative online C compiler has been presented in this paper. It has been used by instructors to propose programming exercises and practices, store their codes and evaluate the work of the students analyzing the statistics gathered in the system.

On the other hand, the application allows students to carry out the exercises and practices totally through a web browser, compiling and running their programs without installing any IDE. It includes other useful features, such as search exercises by tags and store code in a repository. And, what is most important and original, is that it offers to the students a mechanism for self-assessment. This method consists of black-box tests, through which they must test their programs simultaneously against the instructor's code, comparing both outputs. Indirectly, this tool promotes the student to get one of the skills required for the software developer, that is, software testing.

This tool is specially indicated for classrooms with a high number of students, since currently instructors are not able to manually supervise the students' work and give them feedback. Therefore, it can be a very useful extra tool for any C programming course, so students can carry out exercises and get instantly an indicator of correctness. This would also allow totally online programming courses, since it would serve as a platform for instructors and students to exchange exercises and facilitate the evaluation.

During the 2013/2014 course, the Online C Compiler has proved to be a useful tool for the first programming course at three different degrees. As it has been exposed at Results section, the results has been highly positive: the failure rate decreased with respect to previous years. This is mainly because of the increased number of compulsory practical exercises done by each student through the system presented. The dropout rate also was decreased considerably due to important aspects such as the student motivation for an online environment and greater time flexibility. In addition, instructors have also shown greater satisfaction as it gives them for better tracking of the student.

Although it is already a full and robust system, the authors have proposed three future lines of work:

- On one hand, it is desired to provide an online debug interface using GDB, the GNU Debugger, which is the standard debugger for the GNU Operating System. The debugger is a fundamental piece for any software developer, since it helps to find and reduce the number of bugs. Therefore, the authors consider it as an essential tool to be developed in short time.
- A source code analyzer will be included, based on static ruleset that identifies potential problems or inefficient code, that is, dead code (unused local variables, parameters and functions), empty if/while statements, and duplicate code. This would allow students to realize potential improvements in the code and, consequently, learn to program better.
- And finally, an automatic correction of the exercises will be added as a complementary tool for the instructors, to provide automated grading to the system.

ACKNOWLEDGMENTS

The authors would like to acknowledge and thank Leandro DEL OLMO COTO, student of our School who developed the software of the system presented. The source code is maintained and hosted as a free software project at SourceForge (https://sourceforge.net/p/compiladorc/).