How much C can students learn in one week? Experiences teaching C in advanced CS courses

1st Rafael Corsi Ferrao
Computer engineering
Insper– Instituto de Pesquisa e Ensino
Sao Paulo SP, Brazil
rafael.corsi@insper.edu.br

2nd Igor Dos Santos Montagner
Computer engineering
Insper– Instituto de Pesquisa e Ensino
Sao Paulo - SP, Brazil
igorsm1@insper.edu.br

3rd Ricardo Caceffo
Computer Science
Unicamp
Campinas - SP, Brazil
caceffo@unicamp.br

4th Rodolfo Azevedo
Computer Science
Unicamp
Campinas - SP, Brazil
rodolfo.azevedor@ic.unicamp.br

Abstract*—In this paper Innovative Practice Full Paper we analyze the results of the application of a Concept Inventory for introductory C programming when teaching C for students in advanced CS courses and propose improvements based on misconceptions found and student perceptions. Even though Python is now the most used language for introductory courses, many advanced CS courses, such as Operating Systems and Embedded Computing, still benefit from using the C language. Thus, instructors now face a new challenge: teaching C to students who are already proficient in a higher-level language. Since this must be done in addition to their existing course, it is necessary to (i) be fast; and (ii) assure that students learned enough to be successful in the courses. An approach described in previous work was to concentrate all courses that benefit from using C (Computer Systems, Embedded Computing and Programming Challenges) in a single semester and joining their classes for the first week of the semester. This resulted in promising preliminary results, but lacked a more rigorous assessment in order to draw more meaningful conclusions. Concept Inventories (CI) can be used to both assess how many students have accurate knowledge of the subject but also to identify common pitfalls and misconceptions. In this work we combine three years of experiences in teaching C in advanced CS courses with data from the application of an Introductory C programming CI. We have received 109 responses from three consecutive offerings, detailing student performance in seven key areas. Satisfactory performance was obtained in Parameter use and scope, Iteration, Recursion, Structures and Pointers. Variable Scope and Boolean Expressions were the most challenging areas. We analyze the most common misconceptions and relate them to the material and students' previous experiences. A questionnaire answered by students (N=56) at the end of the activity is analyzed in order to understand the level of affinity of students with the C language before the course and their perceptions about it, we noticed that most students have not had contact with the language before and feel motivated to learn C. To verify the knowledge in the language on a practical project, we analyzed the first big delivery of code that they have to do after two weeks the end of the course , through the analysis of the students codes (N=58) we can measure that many students use more advanced resources of the language that were not seen in the course or taught officially by other courses, indicating that they were able to learn enough of the language to go it alone in more complex matters.

***Index Terms*—Concept Inventory, Misconceptions**

I. Introduction

Many advanced CS courses, such as Embedded Systems, Operating Systems and Algorithm Analysis, are traditionally taught using the C programming language, either for technological or pedagogical reasons. However, many introductory courses such as CS1/2 have migrated from C/C++/Java to using Python as a first language [1]. Thus, there's a need to teach C programming to students that are already proficient in other programming languages.

In this context, two important issues arise: (i) teaching C programming is not usually part of the Learning Objectives of advanced CS courses; (ii) C programming is required to accomplish those Learning Objectives. Advanced CS courses that rely on C must address these issues to some degree and many authors have explored related questions. Raj et. al. [2] describe a complete *Modern C++* course, including assignments and week-by-week program. Riley [3] proposes the use of mobile application development to teach Java. Song et. al [4]. describe a tool to help novice programmers to learn C. McGill et. al. [5] describes the design and implementation of a game to teach pointers. Pantaleon et.al. [6] describe a MOOC style course to teach CS1 in C. Both [7] and [4] describe each a tool to aid in teaching C as an introductory language.

Although these works offer valuable insight on how to teach programming languages in the academic context, they describe experiences where either the main focus appears to be learning that specific technology or are directed at novices. In our context, C programming is prerequisite for the courses and is not part of their Learning Objectives. These imply two important restrictions that, to the best of our knowledge, are not explored significantly in the literature:

1) Learning C programming must not occupy too much time, since it is not part of the Learning Objectives of the courses;

2) There's a need to assess how proficient students are at C programming: since being proficient in C is a requirement to accomplish the course's Learning Objectives, students must receive feedback regarding whether they are already at the minimum required level.

In this paper we describe an updated version of the C crash-course [8], a one-week joint program among three advanced CS courses, and analyze the following Research Questions based on offerings in three different semesters.

1) **RQ1**: can students learn the basics of C in one week?
2) **RQ2**: which are the most frequent/important misconceptions detected?
3) **RQ3**: can students complete practical projects after the crash-course?

We administered three times, one per class the Introductory C Programming Concept Inventory to assess students' C programming proficiency and help answer these three Research Questions. It is used both to evaluate whether students are proficient in the most basic aspects of C and to detect which misconceptions students make.

This paper is organized as follows. In Section II we review previous works related to both the C crash-course and the Introductory C Programming Concept Inventory. In Section III we analyze data collected from several crash-course offerings. In Section IV we discuss our research questions in light of the data. Finally, in Section V we present our conclusions and future work.

II. Background

In the fifth semester (out of ten) of the sequential course in computer engineering at Insper (Brazil - SP), students take the courses of Embedded Computing, Operating System, and Algorithms, all of which make use of the C programming language. Students arrive that semester with experience in python and java, but without ever having officially had C at school. In a survey conducted with students at the beginning of the semester (N=54) only 16% of students said they had programmed something in C before and 60 % said they had never seen any C code, the rest commented that they had already contacted with the language but without manipulating the code. To solve this problem, the three courses make a common effort and provide an intensive 13-hours course during the first week of classes to teach what has been considered the minimum for the courses to start. At the end of the week, an assessment is applied in order to identify students who have difficulty in language and may suffer from it with the course of the courses. The assessment is a concept inventory [9], [10] that allows us to identify misconceptions.

The first version of the course was created in 2018, but refactored in 2021 when we need to migrate to the hybrid format. In the beginning, most of the activities were carried out on the hardware of the embedded computing discipline, which made it difficult for the course to progress since many variables were involved (hardware bugs, debugger driver, IDE), and professors from other areas did not feel sure to provide the necessary support. The evaluation used was created by the teachers themselves and was a mix between multiple choices and programming on paper. Here we present the current version of the course that is analyzed in this article, in this version we removed the hardware dependency, created new handouts and made the course self paced.

A. About the crash course

The course is based on handouts were students are recommended to carry out in a schedule (Tab. I) that is most synchronous with the meetings of the disciplines, two handouts are to be done by students outside of class hours, they are usually small and of low complexity. Handouts have a mix of theory, quizzes and practical programming activities. In total there are 7 handouts covering: basic concepts, compilation, arrays, strings and arrays, pointers and *structs*, variable scope. Table I indicates the schedule followed in the course, with the amount and time foreseen for each stage.

The objectives of each handout are:

1) Basic syntax, conditionals and loops, functions, variable types.
2) complete skeleton of a C program; user input/output; math.h; functions
3) Gnu build toolchain, bash process substitution
4) arrays, strings, matrix
5) Pointers, structs, pass by value / reference
6) Recap and practice
7) Variable scopes: global, local; formal parameters

The practical activities where created to work with the GNU toolchain and indicated to be done on the Linux OS, before the beginning of the classes an email is sent to the students instructions for installing the OS as well as the indicated packages (*build-essential*, *libsystemd-dev*).

The course material consists of an interactive web page created with the active-handout tool [11], which allows the instructors to integrate quizzes and exercises into a handout, and a git repository with C codes with a unit testing system that validates students answers. The practical exercises are referenced in the handout, indicating where the student must program and how to perform the tests; each program must be compiled and executed locally. Below we detail each one of the handouts:

1) Basic concepts: In the first meeting instructors of the three subjects briefly present an overview of the course they will teach, part of the presentation focus on why the C language is important for each course. After the presentations, students start the first handout that aims the basic syntax of the language, as students have already had java before part of the introduction is done comparing C with java. The handout is created on top of manipulating a black and white image, all the reading and writing part of the image has already been provided, students only need to modify a function called int *process_pixel(char level)* which is called for each pixel of the Image. Students are guided to apply a series of simple image processing that exercises the handout content,

Day: Dedication time:	Monday 4-hour sync	Tuesday 30min async	Wednesday 4-hour sync	Thursday 2-hour sync + 30min async	Friday 2-hour sync
Handout:	1 - Basic Concepts 2 - Practicing	3- Compiling with gcc	4 - Arrays, strings and Matrices 5 - Points and Structs	6 - Putting it all together 7- Variable scope	Evaluation

TABLE I
PROPOSED COURSE SCHEDULE.

which basically covers: variable declaration, conditions and mathematical operations.

In one of the programming exercises, we asked students to adjust the brightness of an image, for that they must add an extra argument in the function and use it as the gain of the image's pixel. A very common mistake that happens is that the student just adds the value of the current pixel to the offset passed by the argument. This causes a pixel value overflow of 255, the students must then reason what is happening and come up with code that solves the problem. Fig. 1 is how the exercise is presented to the student.

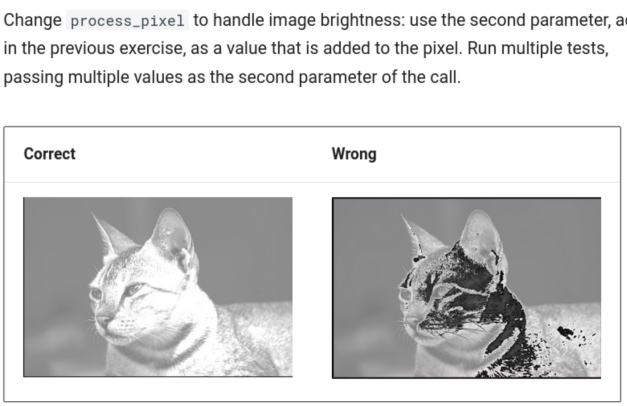

Fig. 1. Exercise statement to adjust the brightness of an image.

Every programming exercise has an explanation of what must be done, the file that must be manipulated and how the student performs the test.

2) Practicing: As a continuation of the first handout, the full skeleton of a C program is introduced: function main and includes. *Printf* and *scanf* is present as ways of interacting with the terminal and then the math functions (*math.h*). A series of small practical exercises are suggested for students to do during each introduction of content, such as: calculating sine and cosine; function that returns the modulus between two numbers and another to calculate the Manhattan distance. Finally, two slightly more complex exercises are suggested.

3) Compiling with gcc: It explains the main steps for compiling a C program using the gnu toolchain, not going into much detail in the *Makefile* (which will be covered throughout the OS course). To prepare students for upcoming handouts and facilitate program testing, introduce the bash concept of process substitution. We will use this so that students can interact with files as an alternative to reading files.

4) Arrays, strings and Matrixes: Introduce the concept of static memory allocation in C, starting with arrays and going through strings. The handout makes use of quizzes to validate the central part of the theory and programming exercises to practice. The concept that a vector is just a continuous region of memory is new to students, and difficulties arise when they have to scan a vector without knowing its size.

String exercises are kind of like getting a string from the terminal, calculating its length; count the amount that a letter appears. In arrays the approach is very similar, with a focus on how C allocates the array in memory and how indices work.

5) Pointers and Structs: In this handout, we recommend that students carry out the activities in pairs, as it is a subject that causes a lot of difficulty in understanding. In this material we abuse the use of quizzes and questions to validate the understanding of small snippets of code, taking a little longer to get into the practical part, having a larger theory section with several commented code examples. The theory starts with the concept of pointers, then introduces the syntax of how to declare a pointer-type variable or how to access the address of a variable. Students so far had used the *scanf* function, but did not know the meaning of '&', the material retrieves this question to indicate one of the pointer uses. Then it moves on to using pointers as a way for a function to return more than one value. Two pointer programming exercises are suggested: write a function that returns the addition and subtraction of two parameters; and another to calculate the height and width of a rectangle. Both programs must be written in full.

Following the *struct* concept and syntax are presented, students must then modify functions they have already performed that receive more than one parameter to now operate with *structs*. Next, it is introduced how to use pointers to *structs*.

6) Putting it all together: In this step, students will create a program from scratch that receives an image (via process substitution) and stores it in a *struct*. The *struct* must have information about the width and height of the image, as well as the image pixels stored in an array. The image must be processed and then saved to a *pgm* file running the following command:

```
./imageProcess < image_in.pgm > image_out.pgm
```

To facilitate processing, we chose the *pgm P2* format, where the image pixels are in *ASCII* and separated by a space. Students must parse the header to identify the properties of the image. We indicate a code structure and which functions must be created. Advanced exercises ask to create functions that perform image cropping, blur and border detection.

7) Variable scope: This is a short material and has no practical activity. It was created because we identified by the first CI response that students were having difficulty in this category. The handout deals with the subjects in the following sequence: global variables, local variables and formal parameters. The handout is a mix of explanation, example codes and

three quizzes that address the issues dealt with, Fig. 2 is an example of these issues.

Fig. 2. Scope exercise exercise built into the original handout in Portuguese, where: Exercicio is *Exercise* and *Enviar* is *Submit*.

B. Concept Inventory

An emerging approach within the educational environment is the identification of students' misconceptions, incorrect mental models crafted during the learning process. The origin of misconceptions can be both related to the way a certain subject is taught by the instructor and/or be a common problem for students at that point of learning. Some of these misconceptions are resolved naturally as the course advances, but some of them are incorporated by students. This can negatively impact the whole learning process.

There are misconceptions cataloged in some areas of knowledge, such as physics [12], math [13], and computer science [14], [15]. The identification and validation of misconceptions are usually performed through the analysis of exams/tests and interviews with students and instructors. The knowledge of what are the students' misconceptions can be used to improve the learning process, for example through interventions that target these misconceptions at the exact point of the course they are expected to happen.

Another possibility is to design Concept Inventories (CIs). A CI is usually a multiple-choice questionnaire administered to assess which misconceptions students have at some point in the course. Each incorrect question's choice (also called a distractor) is mapped to a specific misconception, so each question addresses more than one misconception.

This design decision exists because the number of questions in a CI must be limited, to allow it to be applied to students in a reasonable time. Also, since the same misconception appears in more than one question, it is expected that a student who has that misconception consistently selects that misconception in all (or almost all) its occurrences.

A CI can be used, for example, to assess the learning gain of students in a certain period. For example, it can be administered at the beginning and end of different classes (experimental and control groups) in some courses [16], thus supporting the assessment of different learning approaches, like Active Learning [9], [10] methods.

The entire process of elaboration of CIs, as well as the validation of their internal consistency, makes the development of this instrument something expensive and time-consuming to be done. In the literature, the first CI developed, and the most adopted, is the Force Concept Inventory (FCI) [12], related to the Newtonian Laws. In Computer Science, specifically in CS1, there are initiatives to develop CIs in C [17], [18], Python [19] and, Java [20], [21] languages.

In this study, we adopted the CI designed by Caceffo et al. [17], [18] for the C programming language. The CI is composed of 27 questions, addressing 33 misconceptions in the following topics: a) function parameter use and scope; b) variables, identifiers, and scope; c) recursion; d) iteration; e) structures; f) pointers and; g) boolean expressions.

III. Methodology

In this article we will analyze the last three crash course interactions: 2021-1 (N=40); 2021-2 (N=20) and 2022-1 (N=49). The course was applied at the same time as the undergraduate course (fifth semester, computer engineering) without substantial changes in the previous courses that the students took.

The material and exercises are basically the same between all interactions, apart from minor corrections in the content and the insertion of a new material in the 2021-2 version, which included a new handout of *variables, identifiers, and scope*. There was only one instructor change between the 2021-1 and 2021-2 versions, keeping the lineup the same for the rest.

Due to the coronavirus pandemic, the 2021 versions of the course took place in a hybrid way, while the 2022 version took place 100% in person. Fig. 3 is an overview of the different applications of the course.

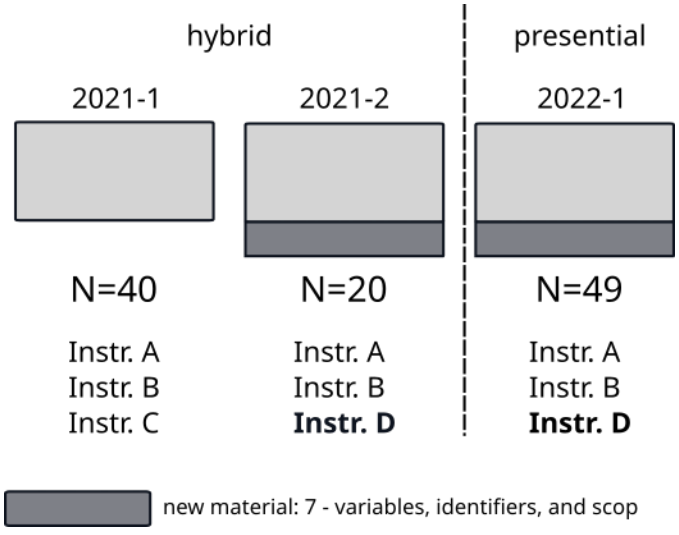

Fig. 3. Course interaction overview.

The CI test administered was the same between all interactions and was always carried out at the end of the course. The time reserved for the test was 2 hours and the students

performed it on the computer. In general, we will analyze the CI results as a single data-set with all iterations, but some items will be analyzed per semester to verify the impact of some interventions in the course. The test result will be explored by the seven categories of the CI and not individually by question. Misconceptions will be analyzed by the categories proposed in the test, and we will use the criterion that for a misconception to be relevant, it must appear more than once for the same student.

At the end of the CI evaluation, an online questionnaire with six multiple choice/ scale questions about the perception of the course was applied to the students, and one open-ended question, voluntarily 54 students from the three semesters responded: 2021-1 (N=28); 2021-2 (N=5) and 2022-1 (N=23).

To check if students can use the language and develop a practical project, we will use the results of the first large delivery of code that students need to make after the crash course. This happens 2 weeks after the end of the crash course in the embedded computing discipline. The project done in pairs and has requirements for the use of C language and code quality as students want to improve their grade. The codes analysis from 2021-1 (N=21), 2021-2 (N=10) and 2022-1 (N=27) will be used as a single block, since the project and the rubrics remained the same throughout the interactions. Along with the students code, a form that asked each pair to rate their difficulty with the C language on a scale, will be used to understand the student's perception of their resourcefulness in the language.

IV. Results

The main objective of the results is to understand who our students are and what level of maturity they reached in the language during the course, if students can learn the basics of C in one week, which are the most relevant misconceptions and if students can use the language to solve practical programming problems. For this, we will analyze a questionnaire that the students filled out, the concept inventory test and the misconceptions found and also the result of the first project of the embedded computing discipline that was delivered two weeks after the end of the crash course and that contains programming requirements.

A. Questionnaire: Students perception

Students answered the following questions in the questionnaire:

a) Prior contact with the C programming language: The answers show that 60% of the students indicated that they had no or very little contact with the language before the beginning of the course, only 2% of the students indicated that they used it regularly before the course.

b) How much extra time each student spent during the week: Regarding the extra time dedicated to studying for the preparation of the test: 11% did not dedicate any extra time, 55% dedicated up to 4 hours of study and the remaining 6 or more hours. This indicates students' engagement with learning, since the assessment has little relevance in the grade of each of the disciplines.

c) What motivated students to study: 22% answered that they studied only for the test; 41% who were motivated by a mix of learning the language and doing well on the test and 37% who just wanted to learn C language.

d) What materials did the students consult?: 43% of students consider that all the content needed to continue with the course is found in the material provided, while the rest of the students (57%) said that they consider that the material provided has most of the necessary material, for these students we asked which other materials they consult: sites like stackoverflow or geeksforgeeks were cited more than once.

In the part of the questionnaire that allowed students to write a comment about the course, some common issues appeared:

- seven students (13%) commented that they felt the lack of some type of specific content that was required in the CI test. Being three in 2021-1 commenting that they didn't know what it meant to declare a variable outside a function and that this was charged in the test. Others talked about: binary operation and recursion, subjects not covered in the course, but required in the test.
- another five students (9 %) said that it would be interesting to have access to the exercises solution, so that they could compare the results.

Other questions that appeared occasionally were: Some students did not like the test not being practical, and felt that it did not assess the ability to make a program in C; Others commented that handout 6 is confusing and needed to be explored further.

The lack of course content such as recursion and binary operations was a decision made when the course was created in 2018-1. It was considered the available time for the course and what we would be able to deliver and what would actually be the bare minimum for the three disciplines. Binary operations, for example, is covered in the first week of the embedded computing discipline, when students need to create masks to access specific configurations of the microcontroller peripheral; recursion is the subject of programming analysis and will be addressed by it.

B. Concept Inventory

Analyzing the test as a grade, the students had good results with an average of 8.07 out 10 ($\sigma = 1.07$) with 5 being the lowest grade. Fig 4 is the histogram of the grade, calculated as a simple average across all CI questions.

Table II summarizes CI results by each category of the test, showing the total times that the misconceptions appears and in the **Misconception %** column is the students percent that the misconception appeared more than once.

In category Function Parameter Use and Scope no misconception is relevant. However, if we look more broadly for the A.4 (N=27) and the impact of this in the subjects outside the crash course we can see that many students miss exercises of disassembly function call in the OS course, confusing the order of the arguments with the order in which the registers were changed. A.4 indicates a problem with *Incorrect order of function parameters.* here students believe the program will

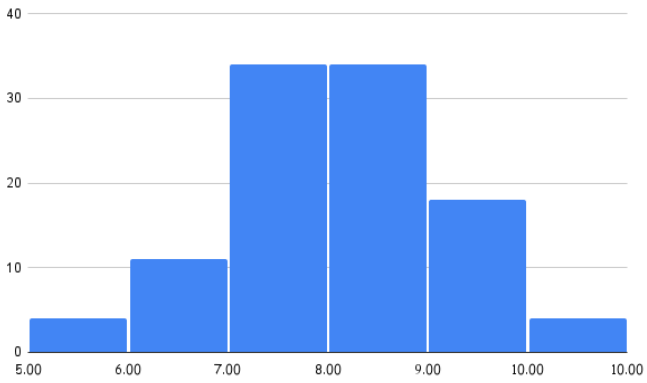

Fig. 4. CI final grade histogram

	ID	Total	Misconception %
Function Parameter Use and Scope	A.4	27	**0.95**
	A.5	17	
	A.1	2	
	A.3	1	
	A.2	1	
	A.6	1	
Variables, Identifiers, and Scope	B.2	80	**21.9**
	B.4	55	**19.05**
	B.1	14	
	B.3		
Recursion	C.2	16	**5.71**
	C.3	5	**0.95**
	C.1	5	
Iteration	D.2	12	**0.95**
	D.3	10	
	D.4	10	
	D.6	8	
	D.5	6	
	D.1	1	
Structures	E.3	7	**1.9**
	E.5	6	**0.95**
Pointers	F.2	35	**1.9**
	F.4	34	**7.72**
	F.5	14	**3.81**
	F.3	14	**1.9**
	F.1	7	**1.9**
Boolean Expressions	G.2	47	**10.48**
	G.1	31	**1.9**
	G.3	20	**1.9**
	G.4	5	

TABLE II
CI RESULTS OVERVIEW (N=105), **ID** IS THE MISCONCEPTION OF EACH GROUP OF QUESTION; **TOTAL** THE NUMBER OF TIMES THAT IT APPEARS ON ALL DATA AND **MISCONCEPTION** IS THE PERCENT OF STUDENTS IN WHICH THAT MISCONCEPTION APPEARED MORE THAN ONCE, EMPTY CELLS ARE MISCONCEPTION THAT DO NOT APPEARED MORE THAN TWO TIMES.

understand the semantics related to the variable names and adjust their order properly.

Variables, Identifiers, and Scope has two relevant misconceptions: B.2 on 21.9% of the students and B.4 on 19.05% representing together 40% of students. Misconception B.2 says that students consider the variables scope as local, and so when a global variable value is changed inside a function, it would not affect other parts of the code. B.4 indicates that students would feel compelled to pass global variables as arguments to functions, or even to avoid their use altogether. This misconception is also identified outside the crash course in the discipline of embedded computing when students create functions that make use of global variables (which were created for communication between interrupt and main), rare are the students who pass the variable as an argument of the function.

If we look at the data for 2021-1 separate from 2021-2 and 2022-1, when a new handout was introduced to address part of this issues, B.2 went from 13.9% to 26.08% with a worsening in student performance and B.4 improved by passing from 38.9% to 8.69%.

In **recursion** the C.2 misconception appears in 5.71% of students which indicates that they have a weak grasp on the concept of recursion itself, leading to the belief that the function does not need to contain a recursive call. Considering the background of students who have not officially had contact with recursion, knowing that only a small portion does not know the least about the subject is something positive.

Regarding **interaction** and **structs** we have no relevant misconception identified by the test, neither in general quantity nor in percentage of students. We can imagine the reason for this: for interactions the concept is practiced not in a single handout, but dispersed throughout the course; and for structs students can use the accumulated knowledge of other OO languages for variables in an instance, moreover, it's a subject that comes up once more (handouts 5 and 6).

The **pointers** category has one misconception relevant the F.4, which indicates that students consider the presence of a return statement sufficient to determine a function's return type as non-void; they do not realize a function can only return the type determined at the function declaration. The proper concept of what a function does is unclear. Pointers is one of the subjects that causes the most confusion among students [22], the syntax and concept is not trivial and is totally different from what they were used to, having no other reference that can be made to help understanding.

In the **boolean expressions** category the G.2 appears among 10% of students and says that they do not know how to evaluate Boolean expressions with multiple variables, so they build a sequence of if and else statements, each containing a single variable to be evaluated.

C. Post crash course analysis

In the first mini project of the embedded computing discipline, students need to create a firmware that plays monophonic music on a buzzer. The minimum project requirement is at least two songs and visually display through LEDs which song is being played, in addition to buttons for start and pause. To achieve higher grades, students have to deal with new issues of embedded systems, such as interruptions and displays. Besides of embedded systems requirements we ask students to use some features of the C language to better structure the code, analyzing the codes we have: 1) 76% of

students using *structs* for the songs (melody, tempo, name) and adapting functions to use *struct* pointer as an argument; 2) 76% of students creates *.h* files for each of the song; 3) 57% decouples hardware from software using whenever necessary #defines; 4) 42% structure code in *.c* and *.h* 5) 19% use arrays of *struct* pointers to store the sequence of songs.

Along with the delivery we asked the students to classify some of the difficulties foreseen in the project (hardware, logic, team work and difficulties with the C language), 45% of the students said they had no difficulty with the language, 39% had some difficulty but they overcome and 16% considered C language as one of the main difficulties of the project.

V. Discussion

Given the results obtained, we consider that the crash course introduces the minimum necessary of the C language for students to follow the specific courses. We do not consider the course equivalent to the long-term ones, and many C language issues still need to be addressed directly or indirectly in the following disciplines. In addition students need to have practice and fluency and this only happens with dedication of time, something we don't have enough during this single week.

The misconceptions detected make sense with the students' learning history and some are perceived outside the crash course, the view that students have difficulties in certain subjects helps teachers to better prepare the materials, activities and assessments.

The new handout that was introduced to improve students' understanding of variables, identifiers, and scope did not have the expected result. The handout was not created by analyzing the misconceptions, but rather from the point of view of what was important for the teachers of this subject.

We would like to analyze how changes in the material and in the teaching format impact the student's better understanding of a given issue and if it is possible to correct the misconceptions. A desired evolution for the course would be to create extra materials that would be indicated for students to do based on the result of the CI test or the progress of disciplines in the C language.

The first delivery of the embedded computing discipline is a short project with an average inclusion of 900 lines of code in C, many of these additions are copies of lab functions that occurred throughout the delivery, but integrating this into a project is something very positive and students are able to achieve good code quality, using language features that were not taught (such as separating *.c* and *.h* files), the use of *struct* pointers by 76% of students is something important as it is one of the most difficult language subjects and they manage to overcome.

None of three courses have a dedicated time to teaching only C out of context, students are in contact with the language and using more complex codes that were passed as an example. We believe that the students' good results and resourcefulness are a consequence of the crash course, which provides the basis for them to be able to evolve in the language.

As future work we intend to revisit the course schedule giving priority to pointers and removing subjects that are not immediately used throughout the courses, matrix is something taught in the crash course, but not immediately used by students. Another idea would be to not have a specific moment for the CI test, but to spread the questions through the handouts and make the assessment a practical activity thus having one more degree of analysis of students' competence in language.

References

[1] P. Guo, "Python is now the most popular introductory teaching language at top us universities (2014)," *Communications in ACM, Blogs*, 2017.

[2] A. G. S. Raj, V. Naik, J. M. Patel, and R. Halverson, "How to teach modern c++ to someone who already knows programming?" in *Proceedings of the 20th Australasian Computing Education Conference*. ACM, 2018, pp. 97–104.

[3] D. Riley, "Using mobile phone programming to teach java and advanced programming to computer scientists," in *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. ACM, 2012, pp. 541–546.

[4] J. Song, S. Hahn, K. Tak, and J. Kim, "An intelligent tutoring system for introductory c language course," *Computers & Education*, vol. 28, no. 2, pp. 93–102, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360131597000031

[5] M. M. McGill, C. Johnson, J. Atlas, D. Bouchard, C. Messom, I. Pollock, and M. J. Scott, "If memory serves: Towards designing and evaluating a game for teaching pointers to undergraduate students," in *Proceedings of the 2017 ITiCSE Conference on Working Group Reports*. ACM, 2018, pp. 25–46.

[6] C. B. Pantaleon, L. S. Feliscuzo, and C. L. C. S. Romana, "Mooc-ready system for a course on fundamentals of programming using c: Development and analysis," ser. ICDTE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 212–216. [Online]. Available: https://doi.org/10.1145/3369199.3369223

[7] D. Pawelczak and A. Baumann, "Virtual-c-a programming environment for teaching c in undergraduate programming courses," in *2014 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 2014, pp. 1142–1148.

[8] I. S. Montagner, R. C. Ferrão, E. Marossi, and F. J. Ayres, "Teaching c programming in context: A joint effort between the computer systems, embedded computing and programming challenges courses," in *2019 IEEE Frontiers in Education Conference (FIE)*. IEEE Press, 2019, p. 1–9. [Online]. Available: https://doi.org/10.1109/FIE43999.2019.9028687

[9] R. Caceffo, G. Gama, and R. Azevedo, "Exploring active learning approaches to computer science classes," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '18. New York, NY, USA: ACM, 2018, pp. 922–927. [Online]. Available: http://doi.acm.org/10.1145/3159450.3159585

[10] R. Caceffo and R. Azevedo, "Improving students' motivation and focus through the gamification in the computer science peer instruction methodology (cspi)," *Comunicações em Informática*, vol. 4, no. 2, p. 16–19, nov. 2020. [Online]. Available: https://periodicos.ufpb.br/index.php/cei/article/view/53115

[11] "Insper active handout tool, published at = https://insper-education.github.io/active-handout/, note = Accessed: 2022-04."

[12] D. Hestenes, M. Wells, and G. Swackhamer, "Force concept inventory," *The Physics Teacher*, vol. 30, no. 3, pp. 141–158, 1992. [Online]. Available: https://doi.org/10.1119/1.2343497

[13] V. L. Almstrum, P. B. Henderson, V. Harvey, C. Heeren, W. Marion, C. Riedesel, L.-K. Soh, and A. E. Tew, "Concept inventories in computer science for the topic discrete mathematics," *SIGCSE Bull.*, vol. 38, no. 4, p. 132–145, jun 2006. [Online]. Available: https://doi.org/10.1145/1189136.1189182

[14] T. Sirkiä and J. Sorva, "Exploring programming misconceptions: An analysis of student mistakes in visual program simulation exercises," in *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 19–28. [Online]. Available: https://doi.org/10.1145/2401796.2401799

[15] K. Karpierz and S. A. Wolfman, "Misconceptions and concept inventory questions for binary search trees and hash tables," ser. SIGCSE '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 109–114. [Online]. Available: https://doi.org/10.1145/2538862.2538902

[16] R. R. Hake, "Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses," *American Journal of Physics*, vol. 66, no. 1, pp. 64–74, 1998. [Online]. Available: https://doi.org/10.1119/1.18809

[17] R. Caceffo, G. Gama, R. Benatti, T. Aparecida, T. Caldas, and R. Azevedo, "A Concept Inventory for CS1 Introductory Programming Courses in C," Institute of Computing, University of Campinas, Tech. Rep. IC-18-06, March 2018, partly in English, partly in Portuguese, 107 pages.

[18] R. Caceffo, S. Wolfman, K. S. Booth, and R. Azevedo, "Developing a computer science concept inventory for introductory programming," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, ser. SIGCSE '16. New York, NY, USA: ACM, 2016, pp. 364–369. [Online]. Available: http://doi.acm.org/10.1145/2839509.2844559

[19] G. Gama, R. Caceffo, R. Souza, R. Bennati, T. Aparecida, I. Garcia, and R. Azevedo, "An antipattern documentation about misconceptions related to an introductory programming course in python," Institute of Computing, University of Campinas, Tech. Rep. IC-18-19, November 2018, in Portuguese, 106 pages.

[20] R. Caceffo, P. Frank-Bolton, R. Souza, and R. Azevedo, "Identifying and validating java misconceptions toward a cs1 concept inventory," in *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 23–29. [Online]. Available: https://doi.org/10.1145/3304221.3319771

[21] R. Souza, R. Caceffo, P. Frank-Bolton, and R. Azevedo, "An antipattern documentation about possible misconceptions related to introductory programming courses (cs1) in java," Institute of Computing, University of Campinas, Tech. Rep. IC-18-20, December 2018, in English, 42 pages.

[22] M. Craig and A. Petersen, "Student difficulties with pointer concepts in c," in *Proceedings of the Australasian Computer Science Week Multiconference*, ser. ACSW '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2843043.2843348