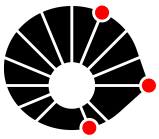
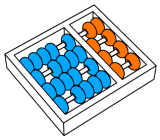




An Exploratory Questionnaire to Support the Identification and Assessment of Misconceptions in CS1 Courses Based on C Programming Language

Ricardo Caceffo *Steve Wolfman* *Kellogg S. Booth*
Guilherme Gama *Islene Garcia* *Tania Caldas*
Rodolfo Azevedo

Technical Report - IC-18-16 - Relatório Técnico
October - 2018 - Outubro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

An Exploratory Questionnaire to Support the Identification and Assessment of Misconceptions in CS1 Courses Based on C Programming Language

Ricardo Caceffo [*] Steve Wolfman [†] Kellogg S. Booth [‡] Guilherme Gama [§]
Islene Garcia [¶] Tania Caldas [‖] Rodolfo Azevedo [**]

Abstract

This Technical Report is part of an ongoing work to develop and assess a Concept Inventory (CI) for Introductory Computer Programming Courses (CS1) based on the C programming language. A CI is a set of multiple-choice questions that can be used to assess the students' comprehension on some topic at some point during a course. Each incorrect choice corresponds to a specific misconception – an inaccurate line of thought students often follow. CIs exist for several knowledge areas, such as physics, chemistry, and statistics. **In previous work** we identified, through the analysis of open-ended exams and interviews with instructors, 19 misconceptions related to CS1 courses in the C programming language. The misconceptions were grouped into 7 topics: function parameter use and scope; variables; recursion; iteration; structures; pointers; and boolean expressions. **In this work**, we: a) present in details the 19 misconceptions identified on the previous work and; b) present the Exploratory Questionnaire created to assess the previous work's misconceptions and also allow the identification of new misconceptions. This Technical Report will be referenced by an article that explains the Exploratory Questionnaire design and the results of its online administration, including the student's performance and individual think-aloud interviews with selected participants.

1 Introduction

This Technical Report is part of an ongoing work to develop and assess a Concept Inventory (CI) for Introductory Computer Programming Courses (CS1) based on C programming language. A CI is a set of multiple-choice questions that can be used to assess students' comprehension on some topic at some point during a course. Each incorrect choice corresponds to a specific misconception – an inaccurate line of thought students often follow. CIs exist for several knowledge areas, such as physics, chemistry, and statistics.

In previous work [1] we identified, through the analysis of open-ended exams and interviews with instructors, 19 misconceptions related to CS1 courses based on the C programming language. The misconceptions were categorized into 7 topics:

[*]Institute of Computing at State University of Campinas (Unicamp) - caceffo@ic.unicamp.br
[†]University of British Columbia (UBC) - wolf@cs.ubc.ca
[‡]University of British Columbia (UBC) - ksbooth@cs.ubc.ca
[§]Institute of Computing at State University of Campinas (Unicamp) - guilhermegama@gmail.com
[¶]Institute of Computing at State University of Campinas (Unicamp) - islene@ic.unicamp.br
[‖]Institute of Computing at State University of Campinas (Unicamp) - unicamp2010@ig.com.br
[**]Institute of Computing at State University of Campinas (Unicamp) - rodolfo@ic.unicamp.br

- **(A) Function parameter use and scope:** students do not understand function calls and parameters passing;
- **(B) Variables, identifiers and scope:** students do not understand declarations of variables or rules for assignment and scope in a program;
- **(C) Recursion:** students do not understand the application of recursion in problem-solving;
- **(D) Iteration:** students are not sure when iteration should be used or how the number of loop iterations is managed.;
- **(E) Structures:** students do not understand how to declare and use C-style structs;
- **(F) Pointers:** students do not understand the concepts of memory addresses and pointers, nor why and how pointers should be used.
- **(G) Boolean Expressions:** students do not understand boolean expressions' meanings or their evaluation;

This work is organized as follows: in section 2 we present in details the 19 misconceptions identified on the previous work [1] and; in section 3 we present the Exploratory Questionnaire created to assess the previous work's misconceptions and also allow the identification of new misconceptions.

This Technical Report will be referenced by an article that explains the Exploratory Questionnaire design and the results of its online administration, including the student's performance and individual think-aloud interviews with selected participants.

2 Misconceptions Details

In this section we present a more detailed description of each one of the 19 misconceptions identified in the previous work [1], including examples. The misconceptions were categorized into 7 topics, as described on section 1. Each misconception received an ID in the format *Topic.Number*. For example, the ID of topic A first misconception is A.1, the second one is A.2 and so on.

The misconceptions described on this section were used to support the design of the **Exploratory Questionnaire**, presented in the section 3.

A.1 Parameter value must be set by an external source

This misconception relates to the disbelief of some students that the parameter of some function already has some value attributed to it. On this situation, students set the parameter value with an external source, like a `scanf`.

Example:

```
1.   int func (int n) {
2.       scanf ("%d", n);
3.   }
```

A.2 Parameters are passed by reference.

This misconception relates to the assumption that parameters are passed by reference, *i.e*, changes made inside a function are reflected outside it.

Example:

```
1.   void addFive (int n) {
2.      n = n + 5;
3.   }
4.
5.  void main ( ) {
6.    int x = 10;
7.    printf(" The value of x is %d", x);
8.    addFive (x);
9.    printf(" The value of x plus 5 is %d", x);
10. }
```

A.3 Parameters can be accessed outside their scope.

This misconception relates to the assumption that parameters could be accessed outside their scope, anywhere on the program. On the following example, there is an error on the line 7, where the parameter `n` from `func1` is accessed outside its scope (inside `func2`).

Example:

```
1.   int func1 (int n) {
2.      n =  n  + 5;
3.      return n;
4.     }
5.
6.   int func2 (int x) {
7.      return x + n;
8.   }
```

B.1 Local variables can be accessed outside their scope.

This misconception relates to the out of scope assignment, where local variables are classified or accessed as if they were global variables. On the following example, the local variable `c` from the main function is accessed inside the function `func` (see line 2).

Example:

```
1.   int func (int a, int b) {
2.      return a + b + c;
3.   }
4.
5.   void main ( ) {
6.      int a = 5;
7.      int b = 7;
8.      int c = 10;
9.
10.     int r = func(a,b);
11.  }
```

B.2 Global variables are local to current scope.

This misconception relates to the out of scope assignment, where global variables are classified or accessed as if they were local to current scope. On the following example, the global `max` variable is considered local to the function func. Therefore the attributions made on lines 4 and 6 would be local, not affecting the global `max` variable value. The prints on lines 14 and 16 would display the same value (10).

Example:

```
1.   int max = 10;
2.   int func (int a, int b) {
3.      if (a >= b) {
4.         max = a;
5.      else {
6.         max = b;
7.      }
8.      return max;
9.   }
10.
11.   void main ( ) {
12.      int a = 5;
13.      int b = 7;
14.      printf ("The max value is %d", max);
15.      int r = func(a,b);
16.      printf ("The max value is still %d", max);
17.  }
```

B.3 Parameter confusion with same-name variable.

This misconception happens when a parameter has the same name of some local variable on the caller function. Students are confused about which variable will be accessed. On the following example, the parameter `c` on the function `addTwoInt` would have the value of the `c` variable on the main function (10), instead of the value of the b variable (7). Therefore the print on line 10 would display the number 15 instead of 12.

Example:

```
1.   int addTwoInt (int a, int c) {
2.      return a + c;
3.   }
4.
5.   void main ( ) {
6.      int a = 5;
7.      int b = 7;
8.      int c = 10;
9..     int r = addTwoInt(a,b);
10.     printf ("The result value is %d", r);
11.  }
```

C.1 Wrong formula used for recursive case return value.

This misconception relates to a wrong return value on a recursive function. On the following example, this would lead to an infinite loop or a stack trace error.

Example:

```
1.   // My recursive function
2.   int recursive (int a) {
3.      if (a == 0)
4.         return 1;
5.
6.      return recursive (a++);
7.   }
8.
9.   void main ( ) {
10.       int r = recursive (10);
11.   }
```

C.2 Recursive function never calls itself.

This misconception relates to a presumably recursive function that lacks a recursive call to itself. The function is called just one time, not being recursive. The following example illustrates this situation.

Example:

```
1.   // My recursive function
2.   int recursive (int a) {
3.      if (a == 0)
4.         return 1;
5.
6.   }
7.
8.   void main ( ) {
9.       int r = recursive (10);
10.   }
```

C.3 Function does not terminate for the base case.

This misconception relates to a presumably recursive function that lacks a base case (stop condition). On the following example, this would lead to an infinite loop or a stack trace error.

Example:

```
1.  // My recursive function
2.  int recursive (int a) {
3.
4.      return recursive (a--);
5.  }
6.
7.  void main ( ) {
8.      int r = recursive (10);
9.  }
```

D.1 Improper update of a loop counter (infinite loop).

This misconception relates to a wrong update of a loop counter, leading to an infinite loop. On the following example this issue is shown on line 5 (the correct counter update would be `i++`).

Example:

```
1.  int i = 0;
2.  int sum = 0;
3.  while (i < 10) {
4.      sum = sum + i;
5.      i = 1;
6.  }
```

D.2 Use of loop result before loop completes.

Students with this misconception use partial results calculated during the loop iterations as if they were final. On the following example, on line 4, for each iteration the partial result is unnecessarily printed.

Example:
Write a program that prints the sum of all numbers from 0 to 9:

```
1.  int sum = 0;
2.  for (int i = 0; i <= 9; i++) {
3.      sum = sum + i;
4.      printf ("The sum is %d", sum);
5.  }
6.  printf ("The sum is %d", sum);
```

D.3 Improper initialization of loop counter.

This misconception relates to an improper initialization of the loop counter, leading to a wrong loop behaviour. On the following example, the loop counter `i` is wrongly initialized on the line 2.

Example:
Write a program that prints the sum of all numbers from 0 to 9:

```
1.   int sum = 0;
2.   for (int i = -1; i <= 9; i++) {
3.      sum = sum + i;
4.   }
5.   printf ("The sum is %d", sum);
```

E.1 Struct comparison missing fields.

This misconception relates to the whole struct comparison, instead of the comparison of specific fields. Students with this misconception believe the program automatically compares all fields of the structs, thus not being necessary to specify any of them on the sentence.

Example: `if (structVar1 == structVar2)`

E.2 Struct comparison missing field in one struct.

This misconception relates to the comparison of a specific field of some struct variable against a whole struct. Students with this misconception believe because they had specified a field on the first struct, the program automatically knows they would like to access the same field on the other struct variable.

Example: `if (structVar1.field == structVar2)`

F.1 Using & instead of * to dereference pointer.

Students with this misconception are confused with the use of & or * to dereference a pointer. On the following example, on line 6, the address of variable `int* c` was changed, instead of the value dereferenced by it.

Example:

```
1.   void main ( ) {
2.      int* c;
3.      int* a;
4.      int* b;
5.      (...)
6.      // Changing the value pointed by c
7.      &c = *a + *b;
8.      (...)
9.   }
```

F.2 Not dereferencing pointer to get value.

Students with this misconception believe they don't need to do anything to dereference a pointer- it would be an automatic process.

On the following example, on line 6, the sum of values dereferenced by `a` and `b` should be assigned to `a`.

Example:

```
1.   void main ( ) {
2.      int c;
3.      int* a;
4.      int* b;
5.      (...)
6.      c = a + b;
7.      (...)
8.   }
```

F.3 Assigning invalid address to pointer.

This misconception relates to the assignment of an invalid address to a pointer. On the following example, the address assigned to `c` was created by the sum of the addresses of `a` and `b`.

Example:

```
1.   void main ( ) {
2.      int c;
3.      int* a;
4.      int* b;
5.      (...)
6.      &c = a + b;
7.      (...)
8.   }
```

G.1 Incorrect precedence for boolean operators.

This misconception relates to a wrong precedence order on boolean expressions.

Example: An expression that should be like `A && B || C` is written `(A && B) || C`.

G.2 Nested if-statements instead of boolean expression.

This misconception relates to the situation where students write a sequence of `if-else` statements instead of a boolean expression.

Example: An expression that should be like `A && B || C` is written like:

```
1       int result = 0;
2.      if (A) {
3.         if (B) {
4.           result = 1;
5.         }
6.         else
7.         if (C) {
8.           result = 1;
9.          }
10.     }
```

3 Exploratory Questionnaire

In this section, we present the Exploratory Questionnaire, composed of questions grouped 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.

Three questions were created for each topic, thus producing 21 questions. The first question for each topic (Q1) was an open-ended (OE) language-dependent (LD) question, designed to identify new misconceptions. The second question (Q2) was either a multiple-choice (MC) or an open-ended (OE) analogy (AN) question, not specific to any programming language. The third question (Q3) was multiple-choice (MC) and language-dependent (LD). The choices available on questions Q3, besides the right one, were mapped to one of the misconceptions identified for the respective topic (see section 2).

Questions received an ID with the format: *Topic.QX (question type)*. On this format, *Topic* is the letter correspondent to one of the 7 topics (A to G) and X is an incremental number. Question type indicates the question classification: OE for open-ended, MC for multiple-choice, LD for language dependent and AN for language independent (analogy). For example, A.Q1 (LD-OE) is the first question on topic A, classified as language-dependent and open-ended.

The questionnaire is exactly the same as the applied to the students, except for the formatting, that was adjusted to this media. Additionally, after each question are presented on this version an *Expected Answer* and/or a brief *Discussion* section, describing the misconceptions that were expected to be related to the question on the moment it was created.

3.1 Topic A – Function Parameter Use and Scope

3.1.1 Question A.Q1 (LD-OE)

Write a program in C that asks the user to enter two `int` numbers, `a` and `b`. Then, if `a` is equal or greater than `b`, calculate and print a-b. Otherwise, calculate and print b-a.

The subtraction must not be executed in the main function. To subtract values use the `subHelper` auxiliary function that performs a subtraction of 2 numbers, as described below:

```
1. int subHelper (int x, int y) {
2.     return x - y;
3. }
```

Expected Answer:

```
1. #include <stdio.h>
2.
3. int subHelper (int a, int b);
4.
5. int main {
6.
7.    int a, b, result;
8.    scanf ("Enter number a: %d", &a);
9.    scanf ("Enter number b: %d", &b);
10.
11.    if (a >= b) {
12.        result = subHelper (a,b);
13.    }
14.    else {
15.        result = subHelper (b,a);
16.    }
17.
18.    printf("Result = %d", %d);
19.
20.
21. int subHelper (int a, int b) {
22.    return a - b;
23. }
```

Discussion:

This open-ended question was created to identify misconceptions related to how parameters are passed and accessed. The `subHelper` function parameters have the same name (`a` and `b`) of the local variables of the main function in order to identify misconceptions related to parameters scope.

3.1.2 Question A.Q2 (AN-MC)

Let's imagine you have a brother who is 5 years younger than you, and you have a sister that is twice your brother's age. You have constructed a machine that can calculate your sister's age based on your age. It consists of:

a) a paper card, with the label "`n =  `" written on it and;
b) a scanner, which can read numbers from paper cards and create and manipulate math; variables and;
c) a printer, which can print any variable registered inside the scanner;

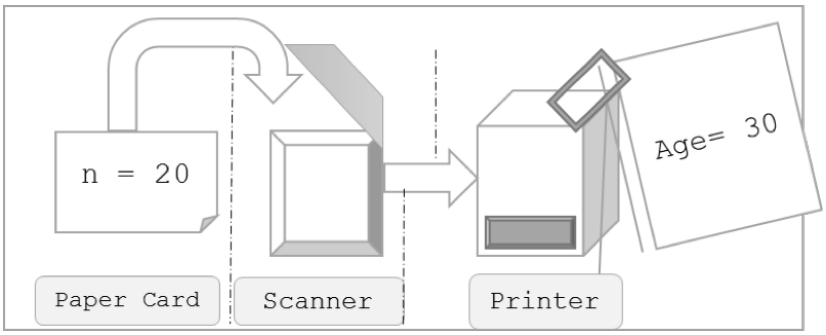
The machine follows these steps:

1. The user (you) writes your age on the blank card, after the "`n =  `" , and inserts the card on the scanner.
2. The scanner reads the n value on the card and performs the calculations.

3. The user removes the card.

4. The printer prints the final calculation result.

The next figure shows an example where the user writes on the paper card his/her current age (20), scans the paper card and then the printer prints his/her sister's age (30).



These commands can be executed in certain steps:

- `write age on card`: User writes his/her age on the card.
- `card scan`: The scanner scans the number written on the card.
- `insert card`: User inserts card into the scanner.
- `remove card`: User removes the card from the scanner.
- `print`: The printer prints some available on the scanner.

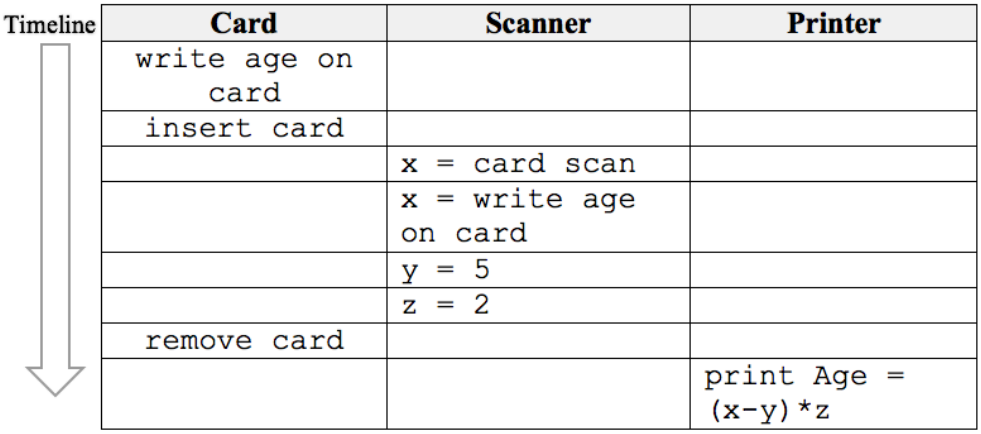
Select the choice that best describes a command sequence to calculate your sister's age based on your age:

a)

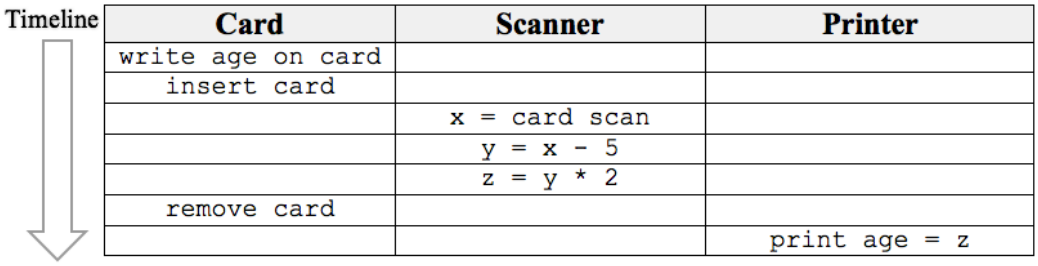
Timeline

Card	Scanner	Printer
`write age on card`		
`insert card`		
	`x = card scan`	
	`y = (n - 5) * 2`	
`remove card`		
		`print Age = y`

b)



d)



e) I don't know.

Discussion:

Besides the right answer (*choice d)*), each other choice matches a specific misconception in the following way:

Choice a) relates to the misunderstanding that parameters could be accessed outside its scope; in this case, the direct access to the n variable, without the `card scan` command. This choice relates to *Misconception A.3.*

Choice b) relates to the change of the original parameter value by an external source; in this case, forcing the `x` variable - that already contains the right value, retrieved from the card scan command - to receive directly the text wrote by the user, through the `write age on card` command. This choice relates to *Misconception A.1.*

Choice c) relates to the misunderstanding that parameters can be passed by reference; in this case, the value of y is changed directly on the card, which does not reflect on the scanner value. This choice also relates to a logic misconception: considering the card was not inserted again on the scanner, there should not be possible to change values inside it. This choice relates to *Misconception A.2.*

3.1.3 Question A.Q3 (LD-MC)

The following code includes a function that adds five to a number. This is called on `x` in the main function.

```
1.      int addFiveToNumber ( int n ) {
2.          int c = 0 ;
3.          // Insert a Line Here
4.          return c;
5.      }
6.
7.      int main ( ) {
8.          int x = 0;
9.          x = addFiveToNumber ( x ) ;
10.         return 0;
11.     }
```

The correct code to be inserted on line 3 is:

a) scanf ("%d", n);
b) n = n + 5;
c) c = n + 5;
d) c = x + 5;
e) I don't know.

Discussion:

Besides the right answer (*choice c)*), each other choice matches a specific misconception in the following way:

Choice a) relates to the change of the original parameter value by an external source. This choice relates to *Misconception A.1.*

Choice b) relates to the misunderstanding that parameters are passed by reference. This choice relates to *Misconception A.2.*

Choice d) relates to the misunderstandings that parameters could be accessed outside its scope. This choice relates to *Misconception A.3.*

Our interest in piloting this question is how attractive the misconceptions are. For example, the `scanf` misconception does not include any addition of 5. Will students avoid it for that reason? As we further develop the question, we will tailor it to try to elicit the actual misconceptions students hold.

3.2 Topic B – Variables, Identifiers, and Scope

3.2.1 Question B.Q1 (LD-OE)

Consider the following code:

```
1.  #include<stdio.h>
2.
3.  void printNumber (int a, int j);
4.
5.  int    i = 5;
6.  int    j = 10;
7.  int    k = 6;
8.
9.  int main() {
10.     int i = 20;
11.     int j = 30;
12.     k = j - i;
13.     printNumber (j,i);
14.     return 0;
15. }
16.
17. void printNumber (int a, int j) {
18.     int i = a + j + k;
19.     printf ("Value = %d", i);
20. }
```

a) List all local and global variables that you can find. Label the local variables with the name of the function to which they belong.

b) Determine the value that will be printed on the screen.

Expected Answer:

a)

```
Local variables:
 - main: i, j
 - printNumber: a,j,i

 Global Variables: i,j,k
```

b)

```
The value will be 60 (30 + 20 + 10)
```

Discussion:

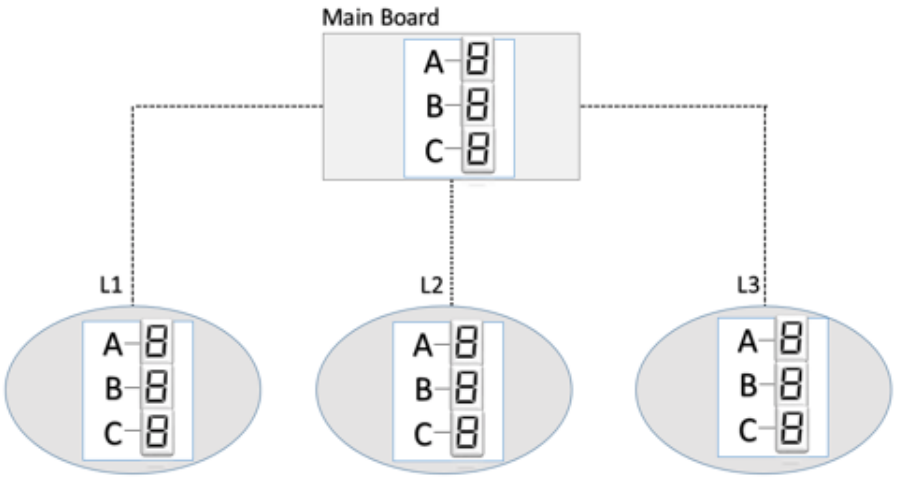
Choice a) could be related to the *"Out of scope assignment "* misconception, assessing if students can identify local and global variables. The `printNumber` prototype was placed just above the first

global variable `i` to verify if the students might consider prototypes as variables.

Choice b) could be related to the misconceptions *B1*, *B2* or *B.3*.

3.2.2 Question B.Q2 (AN-OE)

A special electric circuit (SEC) is composed of the main board and local boards (`L1`, `L2` and `L3`). The main board connects to each local board by wires. Each board contains 3 digital displays (`A`, `B` and `C`) that can be set with numbers (`0` to `9`) or can be blank.

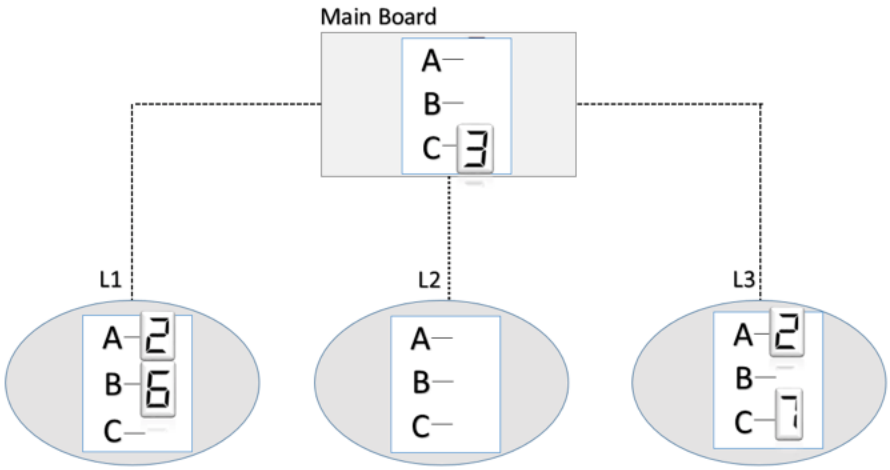


A reading pen can be pressed to some board to read its display's values. For example, if a board has its displays set to `A = 2`; `B = 3` and `C = 4`, the pen will read these same values:

For the local boards (`L1`, `L2` and `L3`) the reading pen first looks on the current board displays values. If a display value is blank, the pen connects to the main board and reads the respective display value from it. If that display value is also blank on the main board, the pen shows **nothing** for that display.

For the main board reading, if the pen does not find a number on some display, it sets that display value to `0`.

Given the following SEC, indicate which display's values the electronic pen will show when pressed to each one of the boards. **If a value in a board should be blank, leave that entry blank**.



Consider the following pressing orders:

a) Main Board, L1, L2 and L3
b) L3, L2, L1, Main Board
c) L1, Main Board, L2, L3
d) L1, L2, Main Board, L3
e) L3, Main Board, L2, L1

Expected Answer:

a)
MB: A = 0; B = 0; C = 3;
L1: A = 2; B = 6; C = 3;
L2: A = 0; B = 0; C = 3;
L3 A = 2; B = 0; C = 7;

b)
MB: A = 0; B = 0; C = 3;
L1: A = 2; B = 6; C = 3;
L2: A = ␣; B = ␣; C = 3;
L3: A = 2; B = ␣; C = 7;

c)
MB: A = 0; B = 0; C = 3;
L1: A = 2; B = 6; C = 3;
L2: A = 0; B = 0; C = 3;
L3: A = 2; B = 0; C = 7;

d)
```
MB: A = 0; B = 0; C = 3;
L1: A = 2; B = 6; C = 3;
L2: A = ␣; B = ␣; C = 3;
L3: A = 2; B = 0; C = 7;
```

e)
```
MB: A = 0; B = 0; C = 3;
L1: A = 2; B = 6; C = 3;
L2: A = 0; B = 0; C = 3;
L3: A = 2; B = ␣; C = 7;
```

Discussion:
The main board relates to the global variables scope and the local boards (`L1`, `L2` and `L3`) to the local variables scope. When a variable is not declared locally, but is present globally, its value can be accessed locally. If a variable was not declared in any scope, it is undefined (blank space).

3.2.3 Question B.Q3 (LD-MC)

```
1.  #include<stdio.h>
2.
3.  void printNumber (int a, int j);
4.  int    i = 5;
5.  int    j = 10;
6.  int    k = 6;
7.
8.  int main() {
9.    int i = 20;
10.   int j = 30;
11.   int k = j - i;
12.   printNumber (j,i);
13. }
14.
15. void printNumber (int a, int j) {
16.   int i = a + j + k;
17.   printf ("Value = %d", i);
18. }
```

The value that will be printed is:

a) 56
b) 60
c) 21
d) 66
e) I don't know

Discussion:

a) Right choice

b) In the `printNumber` function, if the student considers the `k` value from main instead of the global one. This choice telates to *Misconception B.2.*

c) In the main function, when calling the `printNumber` function, if the student considers the values of `i` and `j` from its global scope, instead of its local. This choice relates to *Misconception B.1.*

d) In the `printNumber` function, if the student considers the a parameter having as value the `j` value from main (which is correct), and the `j` parameter also having as value the main `j` value from main (which is wrong, because the correct would be the `i` value), the sum would be: *i = 30 + 30 + 6 = 66.* This choice relates to *Misconception B.3.*

3.3 Topic C – Recursion

3.3.1 Question C.Q1 (LD-OE)

The function `int sum (int n)` was designed to be a recursive function that returns the sum of the n first positive integer numbers. For example, `sum(5)` should return `1 + 2 + 3 + 4 + 5 = 15`.

Consider the following program:

```
1. #include<stdio.h>
2.
3. int sum(int n);
4.
5. int main(){
6.     int n, s;
7.     printf("Enter a number");
8.     scanf("%d", &n);
9.     s = sum (n);
10.    printf("Sum of %d first positive numbers = %d",n,s);
11.    return 0;
12.}
```

Write the function `int sum(int n)` in the space below:

```
1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
14.
15.
```

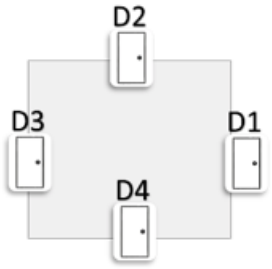
Expected Answer:

```
1. int sum (int n) {
2.
3.   if (n == 0) {
4.       return 0;
5.    }
6.
7.   return (n + sum (n-1));
8. }
```

Discussion:
This question was designed to verify whether students understand how recursion works. Specifically, we want to identify misconceptions related to the stop condition and the return values, but other misconceptions can also be identified.

3.3.2 Question C.Q2 (AN-OE)

You are a Robot named `R1`. You are lost in a maze composed of rooms. Each room has 4 metallic doors, as shown on the picture below. The maze has an exit. If you reach it, you will be transported to the robot's paradise. ;-) Legend also says there is a huge Bitcoin treasure hidden in the maze. If you find it, you keep it yourself.



Behind each door is a passage to another room or a wall. Once opened, a door will stay open forever - it is not possible to close it anymore.

You were programmed to follow these instructions, in the order below:

1. Analyze the room where you currently are.If you find a big E on the ceiling, you are at the exit and instantly leave the maze. If you find a big B on the ceiling, you get the treasure.
2. Open the doors in the following order: `D1`, `D2`, `D3`, and `D4`. You have to follow the "*Open Door Rules* " when opening doors.
3. After you have opened all doors:
   3.1. If your name is `R1`, you send a broadcast with the message: "I am sad. Maze has no exit."
   3.2. If your name is not `R1`, you send a "*Wake Up* " message to the robot that created you and then you enter the *Idle State*.

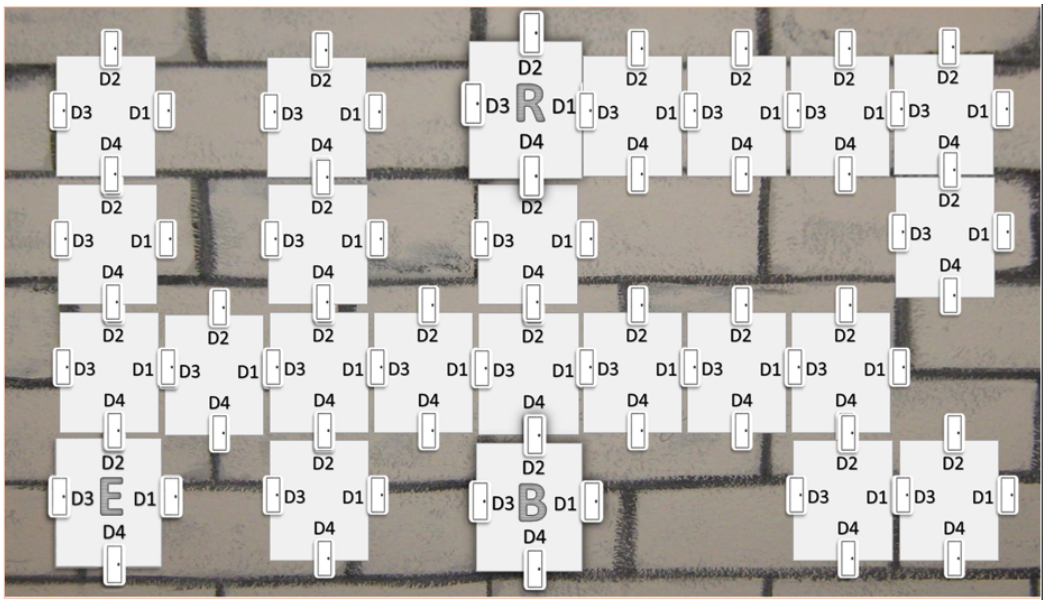
Robot States:

- `Normal State`: A robot operating in a Normal State executes the instructions in the given order and *starts listening.*
- `Idle State`: A robot in an Idle State stops executing any instruction and starts listening for messages. Once it hears a "*Wake Up*" message, it returns to the *Normal State*, and begins executing instructions again from the point it stopped.

Open Door Rules:

1. If you open a door and find a wall, you are immediately done with that door. As a result for example, if `D1` has a wall, you then open `D2`, and so on.
2. If by chance a door is already open, you skip that door.
3. If you open a door and find a passage to another room, you create a copy of yourself **in that room**. The new copy is named `RX`, where `X` is the number of robots in the maze, including itself. After you create a copy your state changes to *Idle State*. Once created, a copy will immediately be active and will start executing the instructions 1, 2, and 3, in that order but in the other room and in the *Normal State.*

Based on the robot and maze rules descriptions, consider the following maze:



The **R** indicates the `R1` robot location; the **B** indicates the Bitcoin treasure and the **E** indicates the exit.

Imagine you are the `R1` robot and start following the instructions you were programmed:

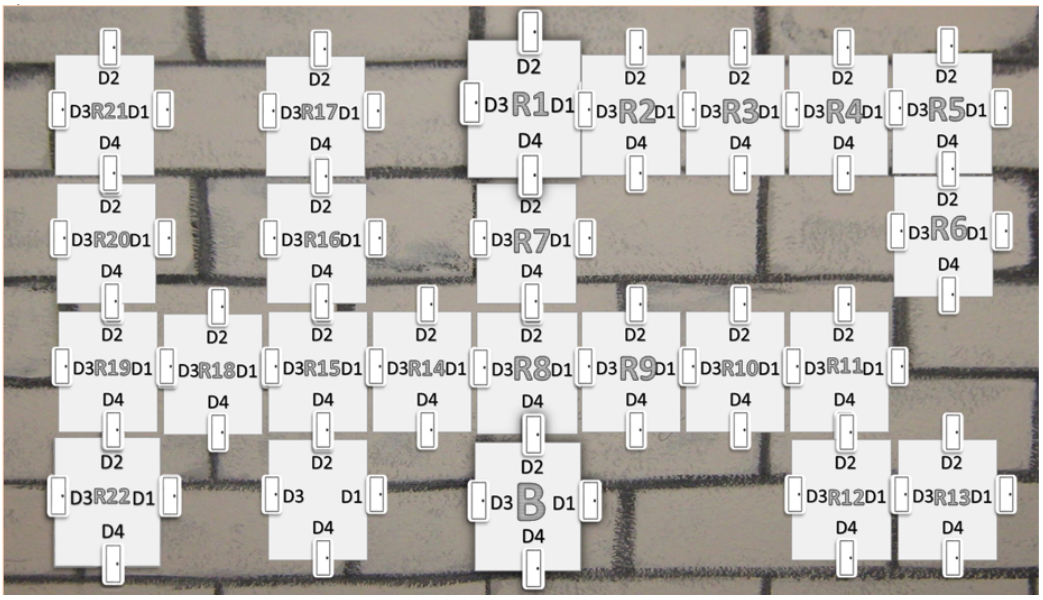
a) Indicate on the maze the robots' names that will occupy each room. Also, indicate for each opened door a number corresponding to its opening order (e.g., the first door opened on the maze will be labeled with the number 1, the second door with number 2 and so on).

b) Will a robot find the Bitcoin treasure? What is it name?

c) Will a robot find the exit? What is its name?

Expected Solution:

a)



The doors will be open on the following order: `R1` room, door `D1`; `R2` room, door `D1`; `R3` room, door `D1`; `R4` room, door `D1`; `R5` room, door `D1`, door `D2` and door `D4`; `R6` room, door `D1`, door `D3`, door `D4`; `R4` room, door `D2`, and door `D4`; `R3` room, door `D2`, and door `D4`; `R2` room, door `D2`, and door `D4`; `R1` room, door `D2`, door `D3` and door `D4`; `R7` room, door `D4` and so on.

b) No robot will find the Bitcoin treasure.

c) The robot named `R22` will find the exit.

3.3.3 Question C.Q3 (LD-MC)

Consider the following code:

```
1.      #include<stdio.h>
2.
3.      int fact(int);
4.
5.      int main(){
6.          int num,f;
7.          printf("\n Enter a number: ");
8.          scanf("%d",&num);
9.          f=fact(num);
10.         printf("\n Factorial of %d is: %d",num,f);
11.         return 0;
12.     }
13.     int fact(int n){
14.        if(n<=1) {
15.            // Insert a Line Here
16.        }else {
17.            // Insert a Line Here
18.        }
19.     }
```

The correct code to be inserted on lines 15 and 17 is:

a) LINE 15: return 1;
   LINE 17: return(n*fact(n+1));

b) LINE 15: return 1;
   LINE 17: // blank

c) LINE 15: // blank;
   LINE 17: return(n*fact(n-1));

d) LINE 15: return 1;
   LINE 17: return(n*fact(n-1));

Discussion:

a) The expression for a return value had an embedded recursive call with a parameter that was larger than the parameter to the current invocation, leading to an infinite loop because the number of levels of recursion will increase indefinitely. This choice relates to *Misconception C.1.*

b) Functions required to be recursive did not call themselves. This choice relates to *Misconception C.2.*.

c) A recursive function must have a stop condition and return a value on tis base case. This choice relates to *Misconception C.3.*

d) Right Answer.

3.4 Topic D – Iteration

3.4.1 Question D.Q1 (LD-OE)

The following function returns 0 if `x` is not divisible by `n` and 1 if `x` is divisible by `n`.

```
1. int xIsDivisibleByN (int x, int n) {
2.     if (x % n == 0)
3.         return 1;
4.     else
5.         return 0;
6. }
```

Create a program that asks the user to input a number and uses the function `xIsDivisibleByN` to determine if it is **prime**, that is, if it is divisible by no number except itself and 1. Assume that the user will input an integer greater than or equal to 2.

Expected Answer:

A prime number is a number divided only by 1 and by itself. By definition, 1 is not prime. The correct answer should have an iteration from 2 to `N` or 2 to `sqr(n)` (best case).

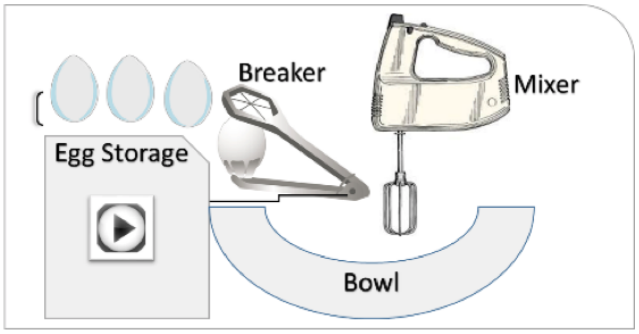
```
1. #include <stdio.h>
2. #include <math.h>
3.
4. int main ( ) {
5.
6.    int a;
7.    printf ("a = ", %d);
8.    scanf ("%d", &a);
9.
10.   int isPrime = 1;
11.   int root = (int)sqrt(a);
12.   for (int  i = 2; i <= root; i++) {
13.     if (xIsDivisibleByN (a,i)) {
14.        isPrime = 0;
15.        break;
16.     }
17.   }
18.   if (isPrime) printf("Is prime")
19.       else printf("Is not prime");
20.   }
```

Discussion:

A possible misconception here could be solving a problem that needs iteration without a loop (*e.g.* if the student only calls the `xIsDivisibleByN` once).

3.4.2 Question D.Q2 (AN-OE)

You are configuring a scrambled eggs machine. The machine has an egg storage, which the user initially loads with any number of eggs. After the user presses the start button, the machine counts how many eggs were loaded, and automatically starts dropping off an egg to the egg breaker, which delivers the egg contents to a bowl, automatically adding a portion of milk and salt. Then the machine beats the mix for 45 seconds, and the process is restarted until there is no more eggs on the egg storage. Finally, the machine beeps 3 times, indicating the process is over.



To correctly configure the machine, you can use the following commands:

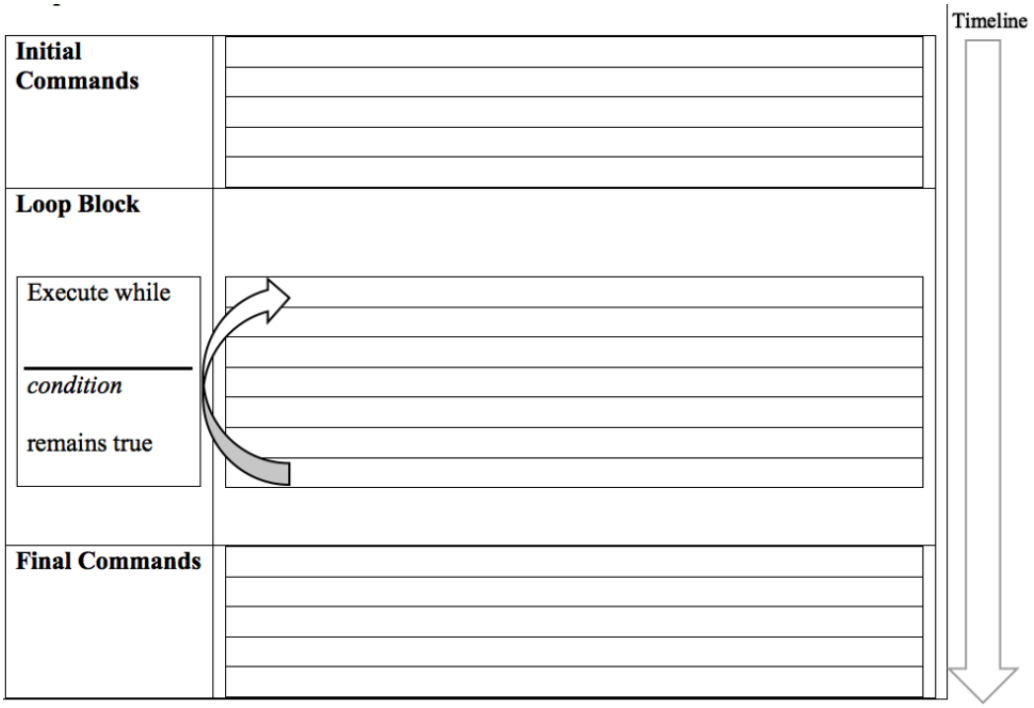
- `Drop one egg`: gets an egg from the egg storage .
- `Break one egg`: breaks an egg and add its contents to the bowl.
- `Add milk`: adds a portion of salt to the mix.
- `Beat the mix`: beats the mix for 45 seconds.
- `Finishing beep`: The machine makes 3 beeps, indicating the process is over.

There is also an x number that can be used to control how many eggs are available. To manipulate x, use the following commands:

- `x = Counter Eggs`: counts how many eggs are present on the egg storage at some time and sets x to this value.
- `Decrease number of x by 1`: decreases x by 1.

The template below is composed by 3 parts: initial commands, a loop block, and final commands. Each command executes individually in the sequence. The entire loop block will execute indefinitely while the *condition* remain true. The final commands only will be executed after the loop block is over (i.e after the *condition* not be true anymore).

Complete the template to program correctly the scramble eggs machine:



Expected Answer:

Initial Commands	`x = Counter_Eggs;`
Loop Block	

Execute while
X > 0
condition
remains true
	`Drop one egg`
	`Break one egg`
	`Add salt`
	`Add milk`
	`Beat the mix`
	`Decrease number of x by 1`

Final Commands	`Finishing beep`

Discussion:

This question relates to the student's ability to write a pseudo code algorithm with iteration. Possible misconceptions relates to the understanding of how a loop block works, and its initialization and stop conditions.

3.4.3 Question D.Q3 (LD-MC)

Which one of the following codes correctly prints the sum of each number from 0 to 9?

a)

```
1. int i = 0;
2. int sum = 0;
3. while (i < 10) {
4.       sum = sum + i;
5.       i = 1;
6. }
7. printf ("The sum is %d", sum);
```

b)

```
1. int sum = 0;
2. for (int i = 0; i <= 9; i++) {
3.     sum = sum + i;
4.     printf ("The sum is %d", sum);
5. }
```

c)

```
1. int i = 0;
2. int sum = 0;
3. while (i < 10) {
4.    sum += i;
5.    i++;
6. }
7. printf ("The sum is %d", sum);
```

c)

```
1. int sum = 0;
2. for (int i = -1; i <= 9; i++) {
3.     sum = sum + i;
4. }
5. printf ("The sum is %d", sum);
```

e) I don't know

Discussion:

a) Improper update of the counter (infinite loop). This choice relates to *Misconception D.1.*

b) Premature result calculation, before the loop is over. This choice relates to *Misconception D.2.*

c) Right choice.

d) Improper initialization of the counter. This choice relates to *Misconception D.3.*

3.5 Topic E – Structures

3.5.1 Question E.Q1 (LD-OE)

Consider the struct:

```
1. struct Employee {
2.     int role;
3.     float salary;
4. };
5. typedef struct Employee Employee;
```

Create a function with the following prototype:

```
int verifyEmployee(Employee e1, Employee e2);
```

The function has to obey the following rules:

- `R1`: return -1 if e1's role is less than e2's and e1's salary is also less than e2's.
- `R2`: return 0 if e1 and e2 have the same roles and salaries.
- `R3`: return 1 if e1's role is greater than e2's and e1's salary is also greater than e2's.
- `R4`: return 2 if none of the previous rules could be verified.

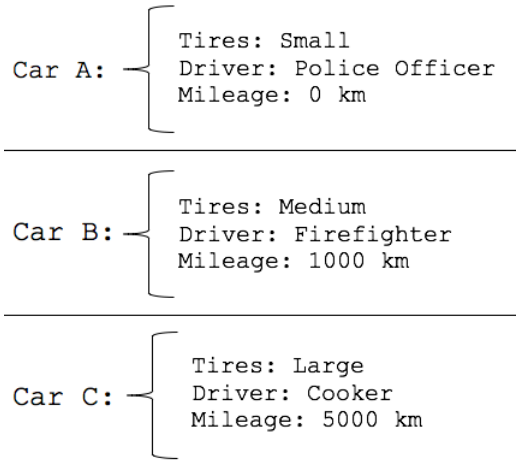
Expected Answer:

```
1. (int) verifyEmployee(Employee e1, Employee e2) {
2.
3.     if (e1.role < e2.role) && (e1.salary < e2.salary) {
4.         return -1;
5.     }
6.     else
7.     if (e1.role == e2.role) && (e1.salary == e2.salary) {
8          return 0;
9.     }
10.    else
11.    if (e1.role > e2.role) && (e1.salary > e2.salary) {
12.        return 1;
13.    }
14.    else
15.        return 2;
16.    }
```

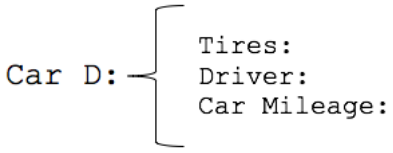
3.5.2 Question E.Q2 (AN-OE)

A customized Lego car allows the removal and replacement of 3 components: tires, driver, and car mileage. The tires can be small, medium and large; the driver can be any Lego figure (police officer, firefighter, etc.) and the car mileage is a number indicating how many km the car has traveled.

Cars `A`, `C`, and `C`, constructed using this approach, are organized as follows:



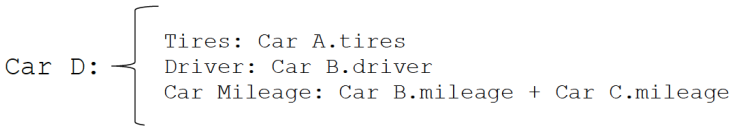
a) Build a new car (`Car D`) using components from other cars. To get a component from a car, use the following syntax: `CarName.component` (e.g., to get the tires from `Car C`, use `Car C.tires`). You can use the math operators - or + if you want. Your car must have the following configuration: small tires, a firefighter as driver and 6000km of car mileage.



b) Using the same syntax from the previous item, write math sentences comparing if `Car D's mileage` is greater than or equal to `Car A's`.

Expected Answer:

a)



b) `Car D. mileage` >= `Car A.mileage`

Discussion:
Choice A verifies if the students understand how to access fields on structs. Letter b verifies if the students understand that in order to compare 2 structs he/she have to access the specific field related to it. Possible misconceptions relates to the whole `struct` comparison, instead of the comparison of specific fields (e.g `Car` >= `Car A`).

3.5.3 Question E.Q3 (LD-MC)

Consider the struct:

```
1. struct Date {
2.    int day;
3.    int month;
4.    int year;
5. };
6. typedef struct date Date;
```

What is the best *if* statement to verify if two variables of type `Date` (`dateA` and `dateB`) have the exactly same date?

```
a) if (dateA == dateB)
b) if (dateA.day == dateB.day && dateA.month == dateB.month
                              && dateA.year  == dateB.year)
c) if (dateA.day == dateB && dateA.month == dateB
                          && dateA.year  == dateB)
d) if (dateB.day == dateA && dateB.month == dateA
                          && dateB.year  == dateA)
e) I don't know
```

Discussion:

Choice b) is right.

Choice a) relates to the whole `struct` comparison, instead of the comparison of specific fields. This choice relates to *Misconception E.1.*

Choice c) and *Choice d)* relate to the whole `struct` comparison against a specific field from other structure. These choices relate to *Misconception E.2.*

3.6 Topic F – Pointers

3.6.1 Question F.Q1 (LD-OE)

Consider the following main function:

```
1.      int main ( ) {
2.          int a = 10;
3.          // Insert a call to changeValues here
4.          printf ("The value of a is %d", a);
5.          return 0;
6.      }
```

Your goal is to make the `printf` on the main function to print the following string: "`The value of a is 20.` "

To achieve this:

a) create a function called `changeValues` following the rules below:

- `R1`: The function return is void.
- `R2`: The function may have any parameters, of any type.

b) Write a call to `changeValues` to be inserted on line 3.

Expected Answer:

```
1. void changeValues (int* a) {
2.   *a = 20;
3. }
4.
5. void main ( ) {
6.   int a = 10;
7.
8.   // Insert a call to changeValues here
9.   changeValues (&a);
10.  printf ("The value of a is %d", a);
11. }
```

Discussion:

This question was created to verify if the students understand how pointers work in C. Answers can help to identify new misunderstandings related to this subject.

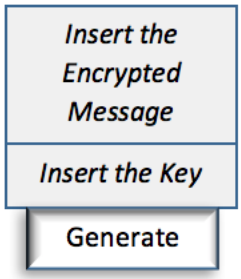
3.6.2 Question F.Q2 (AN-OE)

A spy works in a post office, and his mission is to intercept and decode 3 letters (`L1`, `L2`, and `L3`), identifying a secret message `(SM)` and a secret address `(SA)` from them. Once identified, he will create a secret letter `(SL)`, containing the `(SM)` and addressed to `(SA)`. He will personally deliver the `(SL)` - he does not trust the post office to send letters.

Each letter is composed of an address and a message. The spy knows the letters' messages will be:

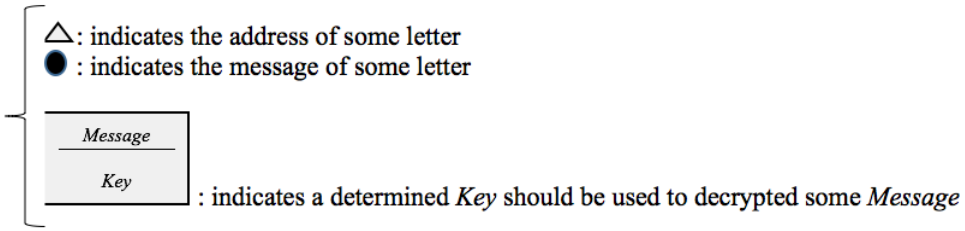
- `L1`: `SA` (encrypted).
- `L2`: `SM` (encrypted).
- `L3`: a key that can be used to decrypt `SA` or `SM` (the spy does not know which).

The spy has a special app to decrypt messages. The app layout is as follows:

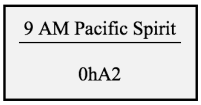


For security reasons, the app only works once for a specific key. If a key is used twice, the app stops working and is deactivated.

The letters will arrive today. To help the spy, you have to send him a note explaining how to decrypt `SA` and `SM`. You have combined the following codes:

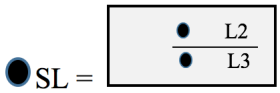


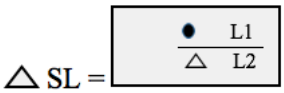
For example, to refer to `L3` message you should use ●`L3`; to refer to `L1` address, you should use ▲`L1` and to indicate for example the key `0hA2` can be used to decrypt the message "`9 AM Pacific Spirit`" you should use a statement like the figure below:



Using the combined codes, write a message to the spy explaining:
a) The `SL` message is the `L2` message decrypted by `L3` message
b) The `SL` address is the `L1` message decrypted by `L2` address.

Expected Answer:

a) 

b) 

Discussion:
This question was created to verify if the students understand the concepts related to pointers and to identify possible new misconceptions related to it.

3.6.3 Question F.Q3 (LD-MC)

Consider the following function:

```
1. int sumTwoValues (int* a,int*b) {
2.     int c;
3.
4.     // Add code here
5.
6.     return c;
7. }
```

Which code should be inserted on line 4 to allow the function to return the sum of two int values correctly?

a) &c = *a + *b;
b) c = a + b;
c) &c = a + b;
d) c = *a + *b;
e) I don't know

Discussion:

Choice a) relates to assigning value to address. This choice relates to *Misconception F.1.*

Choice b) relates to assigning an address to a value. This choice relates to *Misconception F.2.*

Choice c) relates to assigning to a pointer an invalid address. This choice relates to *Misconception F.3.*

Choice d) is correct.

3.7 Topic G – Boolean Expressions

3.7.1 Question G.Q1 (LD-OE)

John wants to go on vacation to Hawaii. Unfortunately, this decision does not depend only on his will. He listed some circumstances that can determine if the trip will happen or not:

- `C1`: John's savings must be at least $ 10000.
- `C2`: Hawaii's local weather must be hot and sunny.
- `C3`: Kids must be approved on school.
- `C4`: His favorite resort be vacant.

John decided he will go on vacation if circumstances `C1` and `C4` are true, or if any 3 or all 4 circumstances are true.

The following program reads the 4 circumstances as `int` numbers. If a circumstance is true, the number read will be 1, otherwise it will be 0. For example, if John's current savings are less than $10000, `c1` will be 0, otherwise `c1` will be 1.

```
1. void main ( ) {
2.     int c1, c2, c3, c4;
3.
4.     scanf ("%d", &c1);
5.     scanf ("%d", &c2);
6.     scanf ("%d", &c3);
7.     scanf ("%d", &c4);
8.
9.     // Complete here
10.
11.
12. }
```

Write the code that must be inserted on line 9 and subsequent lines to make the program print if John will be, or will not be, going on vacation to Hawaii this year.

Expected Answer:

```
9.    if (C1 && C4)         || (C1 && C2 && C3) ||
10.      (C1 && C2 && C4) || (C1 && C3 && C4) ||
11.      (C2 && C3 && C4) {
12.      printf("John is going on vacation");
13.   }
14.   else {
15.      printf("John is NOT going on vacation");
16.   }
```

Discussion:
The focus of this question is to check if the students understand the boolean logic. An alternative solution could consider the sum of variables, as following:

```
9. if ( (c1 && c4) || (c1+c2+c3+c4 >= 3) )
10.      printf("John will go to vacation");
11. else
12.      printf("John will not go to vacation");
```

Although correct, this solution is focused on arithmetic instead of boolean expression (*Misconception G.3).*

3.7.2 Question G.Q2 (AN-OE)

Imagine that APSC 160 opened a selective process for TAs. To be selected as TA, a student must have a grade greater than or equal to 9.0 on the **Selection Exam**; or have a undergraduation average grade of A- or greater and a undergraduation average attendance of 75% or greater.

Another option is when the instructor directly selects the student. In this case, the Selection Exam grade and undergraduation grade and undergraduation attendance are not considered.

a) Complete the statements indicating the unique criteria used to select a student as TA:

A Selection Exam grade >= 9.0

B Average undergraduation grade >= A-

C

D

b) Using the operators `AND` and `OR`, and the parenthesis (and) as delimiters, construct a Boolean sentence that represents how a student can be selected as TA. For example, if to be selected as TA the statements A and B should be true, the sentence would be `A AND B`. If the statements `A` or `C` should be true, the sentence would be `A OR C`.

c) Based on the Boolean sentence you created on item b, complete the following table, indicating for each statement possibility if the student was selected or not to be a TA.

NOTE: The columns indicate if the individual statement is TRUE (T) or FALSE (F). For example, a student with Selection Exam grade of 9.5 would have a T on the A column and a student with average graduation grade of 5.0 would have an F on the B column.

A	B	C	D	Was selected?
F	F	F	F	No
F	F	F	T	
F	F	T	F	
F	F	T	T	
F	T	F	F	
F	T	F	T	
F	T	T	F	
F	T	T	T	
T	F	F	F	
T	F	F	T	
T	F	T	F	
T	F	T	T	
T	T	F	F	
T	T	F	T	
T	T	T	F	
T	T	T	T	Yes

Expected Answer:

a)

A: Selection Exam grade >= 9.0

B: Average undergraduation grade >= A-

C: Average graduation attendance >= 75%

D: Directly selected by instructor

b) (A) || (B && C) || (D)

c)

A	B	C	D	Was selected?
F	F	F	F	No
F	F	F	T	Yes
F	F	T	F	No
F	F	T	T	Yes
F	T	F	F	No
F	T	F	T	Yes
F	T	T	F	Yes
F	T	T	T	Yes
T	F	F	F	Yes
T	F	F	T	Yes
T	F	T	F	Yes
T	F	T	T	Yes
T	T	F	F	Yes
T	T	F	T	Yes
T	T	T	F	Yes
T	T	T	T	Yes

Discussion:
This question relates to the misconception that students are not able to translate English expressions to Boolean expressions (*Choice a)* and *Choice b*)).

Choice c) verifies if the students can evaluate a Boolean expression through a *truth table*.

3.7.3 Question G.Q3 (LD-MC)

The Revised Julian calendar adds an extra day to February in years that are multiples of four, except for years that are multiples of 100 that do not leave a remainder of 200 or 600 when divided by 900. A year with an extra day added to February is called a leap year.

Consider `y` to be an `int` value representing a year. The best clause which identifies if `y` is a leap year is:

```
a) (y%4 == 0)    && !((y%100 == 0) &&
   !(y%900 == 200 ||   y%900 == 600))
b) (y%4 == 0)      ||  (y%100 == 0) ||
   (y%900 == 200) ||  (y%900 == 600)
c) (y%4 == 0)      &&  (y%100 == 0) &&
   (y%900 == 200) &&  (y%900 == 600)
d) (y%4 == 0)      && ((y%100 == 0) &&
   (y%900 == 200   ||   y%900 == 600))
e) I don't know
```

Discussion:

Choice a) is correct.

Choice d) is the same of *Choice a)*, except the NOT (!) is not present. This choice relates to *Misconception G.1.*

Choice b) and *Choice c)* presents all clauses in a line, without the parentheses that indicate precedence order. These choices relate to *Misconception G.1.*

4 Acknowledgements

This research is supported by grant #2014/07502-4, São Paulo Research Foundation (FAPESP), grant #2015/08668-6, São Paulo Research Foundation (FAPESP), Brazilian Federal Agency for Support and Evaluation of Graduate Education (CAPES), Brazilian National Council for Scientific and Technological Development (CNPQ), University of Campinas (Unicamp) and by the Natural Science and Engineering Research Council of Canada (NSERC) under grant RGPIN 116412-11.

References

[1] Ricardo Caceffo, Steve Wolfman, Kellogg S. Booth, and Rodolfo Azevedo. 2016. Developing a Computer Science Concept Inventory for Introductory Programming. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16). ACM, New York, NY, USA, 364-369. DOI: https://doi.org/10.1145/2839509.2844559