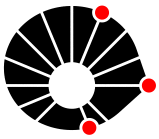

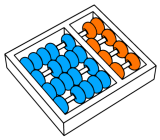

A Concept Inventory for CS1 Introductory Programming Courses in C

Ricardo Caceffo *Guilherme Gama* *Raysa Benatti*
Tales Aparecida *Tania Caldas* *Rodolfo Azevedo*

Technical Report - IC-18-06 - Relatório Técnico
March - 2018 - Março

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

A Concept Inventory for CS1 Introductory Programming Courses in C

Ricardo Caceffo, Guilherme Gama,
Raysa Benatti, Tales Aparecida, Tania Caldas, Rodolfo Azevedo

Instituto de Computação Universidade Estadual de Campinas (UNICAMP), Caixa Postal 6176
13083-970 Campinas-SP, Brasil
caceffo@ic.unicamp.br; guilherme.gama@students.ic.unicamp.br;
raysa.benatti@students.ic.unicamp.br; tales.aparecida@gmail.com; unicamp2010@ig.com.br; rodolfo@ic.unicamp.br

Abstract. This work is a report related to the development and assessment of a Concept Inventory to Introductory Programming (CS1) Courses. A Concept Inventory (CI) is a set of multiple-choice questions addressing specific misunderstandings and misconceptions of the students. In previous works, through instructor interviews, exam analysis, online pilot test and interviews with students, we have identified a list of 33 misconceptions related to 7 programming topics in C language. On this report, we present a CI composed of 27 multiple-choice questions in C language. Each possible answer, besides the right one, was mapped to a previously documented misconception. Future work involves the CI submission to CS1 students and the analysis of its internal consistency and educational impact.

Keywords: Concept Inventory, CS0, CS1, C, Misconceptions, Active Learning, Programming

1. Introduction

This report presents an ongoing work related to the development and assessment of a Concept Inventory (CI) to CS1 Introductory Programming Courses. A Concept Inventory is a set of multiple-choice questions addressing specific misunderstandings and misconceptions of the students [2]. Specifically, this report presents a CI composed of 27 questions in C language (English and Portuguese versions). These questions were created upon previous works that mapped the student's misconceptions in the C language [2, 4, 5, 6].

In previous work [2] we analyzed two CS1 course exams (based on C programming language), classifying the students' errors according to misunderstandings (misconceptions) that might explain incorrect answers. We also conducted semi-structured interviews [5] with five instructors, asking them about the main misconceptions they believed to be common for students. Through this process, we identified seven topics and their misconceptions for a CI: (A) Function Parameter Use and Scope (B) Variables, Identifiers, and Scope; (C) Recursion; (D) Iteration; (E) Structures; (F) Pointers; and (G) Boolean Expressions. A total of 19 misconceptions were identified, two each for Topics G and E, and three each for the others. Figure 1 illustrates the methodology used in the study [2]:

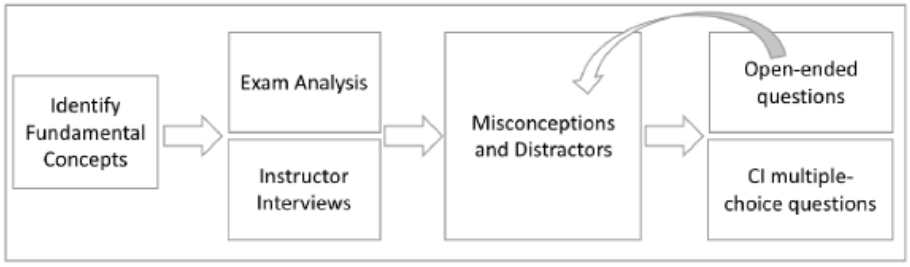

Figure 1. Methodology used in previous work [2] to map student's misconceptions

We have since developed a pilot test for the CI with three goals: (a) to validate the proposed misconceptions; (b) to identify additional misconceptions; and (c) to test the hypothesis that is possible to use analogy questions, not dependent on any particular programming language, to assess the concepts related to the introductory topics we identified.

Three questions were created for each topic, producing a pilot test for the CI with 21 questions. The first question for each topic was an open-ended language dependent question

to identify new misconceptions. The second question was either a multiple-choice or an open-ended (for all other topics) analogy not specific to any programming language. The third question (Q3) was multiple-choice and language-dependent.

The work [4] presents the pilot test for the CI, as well as the data collected after a study conducted at the University of British Columbia (UBC). In that study, we found another 12 misconceptions, thus totalizing a list of 33 misconceptions (21 from the study [2] and 12 from the study [4]) related to CS1 programming in C.

Then, **in our previous work [5]**, we described each one of these 33 misconceptions, following the Antipattern [1] methodology. This methodology describes how to achieve a good solution from a fallacious solution and how to avoid the specious solution in the future. The goal was to document all misconceptions found, also describing how to avoid them to happen in the future.

In parallel, based on the misconceptions described in the work [5], we started the development of a Computer Science Peer Instruction (CSPI) methodology. The Peer Instruction approach, created by Eric Mazur [3], is an Active Learning technique that proposes the instructor to submit Concept Questions to students in the class. If the number of right answers is less than a pre-defined threshold (usually 75%) the instructor stops the lecture, asking the students to discuss and then resubmit their answers.

The **work [6]** presents an Experience Report describing the comparison of 3 classes: a traditional (instructional) class; an active learning PBL (problem-based learning) and a first CSPI version class.

Finally, also based on the misconceptions described in the work [5], **we present in this report** a CI composed of 27 questions. Each possible answer, besides the right one, was mapped to a misconception described in [5]. The details of how the CI was designed are described in Section 2 (Methodology). The CI English version is presented in Section 3, and the Portuguese version in Section 4.

Figure 2 shows how previous publications related to this research [2, 4, 5, 6] are linked:

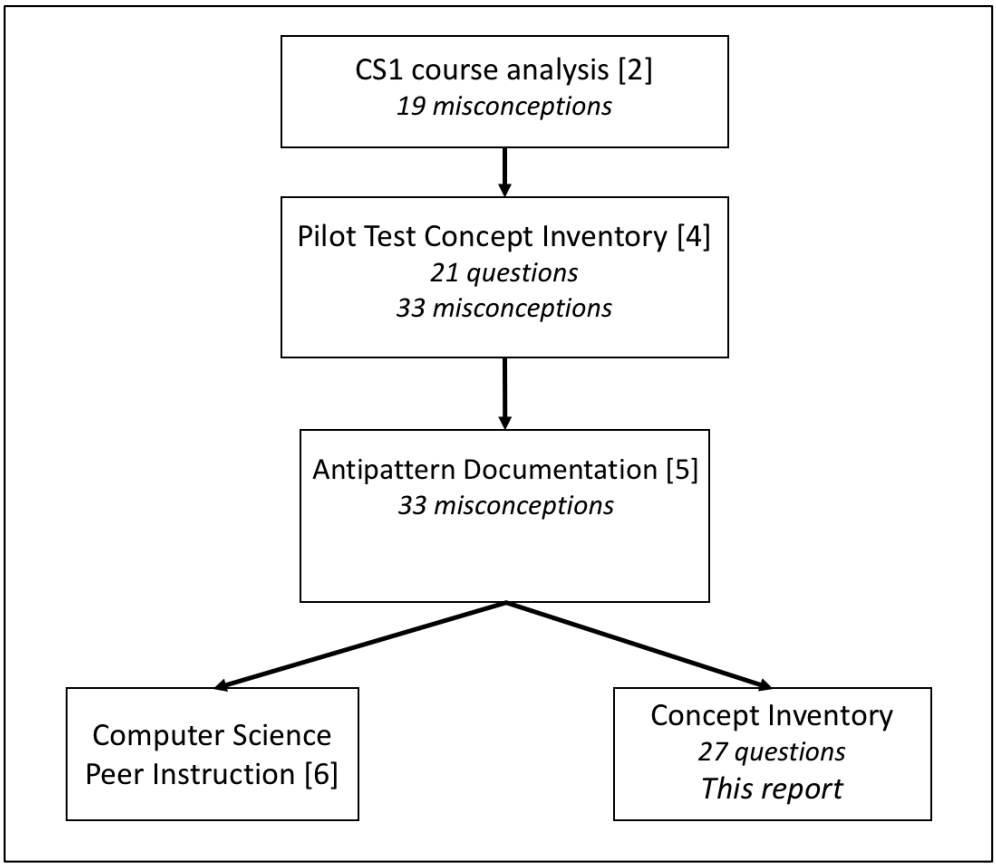

Figure 2. Tree of Concept Inventory research publications [2, 4, 5, 6].

2. Methodology

In previous work [5] we presented the 33 mapped misconceptions, divided into 7 topics:

- A) Function Parameter Use and Scope (6 misconceptions – from A.1 to A.6).
- B) Variables, Identifiers, and Scope (4 misconceptions – from B.1 to B.4)
- C) Recursion (3 misconceptions – from C.1 to C.3)
- D) Iteration (6 misconceptions – from D.1 to D.6)
- E) Structures (5 misconceptions – from E.1 to E.5)
- F) Pointers (5 misconceptions – from F.1 to F.5)
- G) Boolean Expressions (4 misconceptions – from G.1 to G.4)

Considering the future requirement to prove the CI has internal consistency [7] (*i.e.* the student repeatedly demonstrates the same misconception in multiple questions) it was defined that each misconception should be addressed **exactly** 3 times.

This decision led to: a) a CI composed of a different number of questions in each category – as the number of misconceptions in each category is also different and; b) a different number of wrong choices in each question (3 or 4).

Table 1 shows how these numbers were calculated:

Topic	**Number of Misconceptions in [5]**	**Number of Questions**	**Wrong Choices**	**Total available "slots" for misconceptions**	**Each misconception is addressed:**
A	6	5	3 questions with 4 wrong choices	18	3
			2 questions with 3 wrong choices		
B	4	3	3 questions with 4 wrong choices	12	3
C	3	3	3 questions with 3 wrong choices	9	3
D	6	5	3 questions with 4 wrong choices	18	3
			2 questions with 3 wrong choices		
E	5	4	3 questions with 4 wrong choices	15	3
			1 question with 3 wrong choices		

F	5	4	3 questions with 4 wrong choices 1 question with 3 wrong choices	15	3
G	4	3	3 questions with 4 wrong choices	12	3

Table 1. Variables and constraints considered to create the CI

For example, in Topic A we have mapped 6 misconceptions [5]. In order to follow the constraint to have each misconception addressed 3 times, it was needed 18 (*6 x 3 = 18*) slots (wrong question answers) to map these misconceptions. This led to the development of 5 questions: 3 questions with 4 wrong choices and 2 questions with 3 wrong choices (*3 x 4 + 2 x 3 = 18*).

3. Concept Inventory (English Version)

The 27 CI questions were numbered from 1 to 27. Also, each question received an unique ID, in the format Q*TX:* T representing the topic (A..G) as described in Section 2; and X representing the question number in that topic. For example, question QA1 is the first question in topic A, and question QC2 is the second question in topic C.

Also, each question has a "Mapping" table, mapping each possible answer (choices) to a misconception (as described in the previous work [5]) or to a "Right Answer" tag. Figure 2 shows an example for the QA1 question:

Mapping:

Choice	Misconception
a	A.1
b	A.2
c	Right Answer
d	A.3

Figure 2. QA1 mapping. The A.1, A.2 and A.3 ids indicate the misconceptions described in the previous work [5]. The "Right Answer" tag indicates the choice *c)* is the right one.

1. QA1

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

```
1.    int addFiveToNumber (int n) {
2.       int c = 0;
3.       // Insert code 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 in line 3 is:

a) `scanf ("%d", &n);`

b) `n = n + 5;`

c) `c = n + 5;`

d) `c = x + 5;`

Mapping:

Choice	Misconception
a	A.1
b	A.2
c	Right Answer
d	A.3

2. QA2

The following code includes a function that performs the subtraction of two numbers. It is called on `x` in the `main` function:

```
1.   int subtractNumbers (int b, int a) {
2.      int c = 0;
3.      c = b - a;
4.      return c;
5.   }
6.
7.   int main () {
8.      int a = 20;
9.      int b = 10;
10.     // Insert code here
11.     printf ("20 - 10 = %d", c);
12.     return 0;
13.  }
```

The correct code to be inserted in line 10 is:

```
a) int c = subtractNumbers (a, b);

b) int c = subtractNumbers (b, a);

c) subtractNumbers (a, b);

d) c = subtractNumbers (a, b);

e) int c = subtractNumbers (b-a, a);
```

Mapping:

Choice	Misconception
a	Right Answer
b	A.4
c	A.5
d	A.3
e	A.6

3. QA3

Consider the following `main` function, that prints the result of `2a - b` **if** `a >= b`. Otherwise, it prints the result of `b - a`. It uses the `subHelper` function to process the subtractions.

```
1.  int subHelper(int d, int e){
2.    // Insert code here
3.    return d - e;
4.  }
5.
6.  int main(){
7.     int a = 5;
8.     int b = 7;
9.     int r;
10.    if (a >= b){
11.         // Insert code here
12.    }
13.    else {
14.      r = subHelper(b, a);
15.    }
16.    printf("%d ", r);
17.    return 0;
18.  }
```

The correct code to be inserted would be:

a) Line 2: //No code.
   Line 11: r = subHelper(a, 2*b);

b) Line 2: //No code.
   Line 11: subHelper(2*a, b);

c) Line 2: scanf("%d %d", &d, &e);
   Line 11: r = subHelper(2*a, b);

d) Line 2: d = 2 * a;
   Line 11: r = subHelper (a, b);

e) Line 2: //No code.
   Line 11: r = subHelper(2*a, b);

Mapping:

Choice	Misconception
a	A.6
b	A.5
c	A.1
d	A.3
e	Right Answer

4. QA4

Consider the following code:

```
1.   int main(){
2.     int x = 10;
3.     printf("x is %d ", x);
4.     x = addTen(x);
5.     printf("x plus 10 is %d ", x);
6.     return 0;
7. }
```

A correct implementation of the `addTen` function is:

```
a) int addTen (int n) {
      scanf ("%d", &n);
      return n + 10;
   }

b) int addTen (int n) {
      n = n + 10;
      return n;
   }

c) void addTen (int n) {
      n = n + 10;
   }

d) int addTen (int n) {
      int nPlusTen = n;
      return nPlusTen;

   }
```

Mapping:

Choice	Misconception
a	A.1
b	Right Answer
c	A.2
d	A.4

5. QA5

An example of program that reads two values (`a` and `b`) and prints the result of `a - b` is:

a)

```
1.   void subtraction (int a, int b, int r) {
2.      r = a - b;
3.   }
4.
5.   int main(){
6.     int a, b, r;
7.     scanf ("%d %d", &a, &b);
8.     subtraction (a, b, r);
9.     printf("a - b = %d", r);
10.    return 0;
11.  }
```

b)

```
1.  int subtraction (int a, int b) {
2.     return a - b;
3.   }
4.
5.   int main(){
6.     int a, b, r;
7.     scanf ("%d %d", &a, &b);
8.     r = subtraction (b, a);
9.     printf("a - b = %d", r);
10.    return 0;
11.  }
```

c)

```
1.   int subtraction (int n1, int n2) {
2.     return n1 - n2;
3.   }
4.
5.   int main(){
6.     int a, b, r;
7.     scanf ("%d %d", &a, &b);
8.     r = subtraction (a, b);
9.     printf("a - b = %d", r);
10.    return 0;
11.  }
```

d)

```
1.   int subtraction (int a, int b) {
2.     return a - b;
3.   }
4.
5.   int main(){
6.     int a, b, r;
7.     scanf ("%d %d", &a, &b);
8.     subtraction (a, b);
9.     printf("a – b = %d", r);
10.    return 0;
11.  }
```

e)

```
1.   int subtraction (int a, int b) {
2.     return a - b;
3.   }
4.
5.   int main(){
6.     int a, b, r;
7.     scanf ("%d %d", &a, &b);
8.     r = subtraction (a - b, b);
9.     printf("a – b = %d", r);
10.    return 0;
11.  }
```

Mapping:

Choice	Misconception
a	A.2
b	A.4
c	Right Answer
d	A.5
e	A.6

6. QB1

Consider the following code:

```
1.   #include<stdio.h>
2.
3.   void printNumber (int a, int c);
4.
5.   int   g = 6;
6.
7.   int main(){
8.     g = 10;
9.     int a = 20;
10.    int b = 30;
11.    int c = 50;
12.    int d = 100;
13.    printNumber (a,b);
14.  }
15.
16.   void printNumber (int a, int c) {
17.     int result = a + c;
18.     result = result + g;
19.     printf ("Value = %d", result);
20.   }
```

The value that will be printed is:

a) 60

b) 56

c) The program will not compile. Error in lines 8 and 18: variable g not declared.

d) 80

e) If the line 18 were replaced to result = result + d; the value printed would be 150.

Mapping:

Choice	Misconception
a	Right Answer
b	B.2
c	B.4
d	B.3
e	B.1

7. QB2

In the following programs, assume the existence of a function `int existVariable(int var)`.

The function returns 1 if `var` exists in the current scope (the scope the function `existVariable` is called), and 0 otherwise. This is a special function that will always compile, even if the `var` variable is not identified.

The code that prints the phrase: "`r = 22`" is:

a)

```
1.   int func(int a, int b){
2.     return a + b + c;
3.   }
4.
5.   int main(){
6.     int a = 5, b = 7, c = 10;
7.     int r = func (a, b);
8.     printf ("r = %d ", r);
9.     return 0;
10.  }
```

b)

```
1.  int func(int a, int b, int c){
2.     return a + b + c;
3.   }
4.
5.   int main(){
6.     int a = 5, b = 7, c = 10;
7.     int r = func (a, b, c);
8.     printf ("r = %d ", r);
9.     return 0;
10.  }
```

c)

```
1.  int func(int a, int b, int c){
2.     return a + b + c;
3.   }
4.
5.   int main (){
6.     int a = 7, b = 13, c = 2;
7.     int r = func (a, b, b);
8.     printf ("r = %d ", r);
9.     return 0;
10.   }
```

d)

```
1.  int z = 0;
2.  int func(int a, int b, int c){
3.     z = 1;
4.     return a + b + c;
5.   }
6.
7.   int main (){
8.     int a = 7, b = 11, c = 4;
9.     int r = func (a, b, c);
10.     r = r + z;
11.     printf ("r = %d ", r);
12.     return 0;
13.   }
```

e)

```
1.  int z = 0;
2.  int func(){
3.    if (existVariable(z)) {
4.      // z exists in this scope
5.      return -1;
6.    }
7.    else {
8.      // z does not exist in this scope
9.      return 22;
10.     }
11.
12.  int main (){
13.    int r = func ();
14.    printf ("r = %d ", r);
15.    return 0;
16.  }
```

Mapping:

Choice	Misconception
a	B.1
b	Right Answer
c	B.3
d	B.2
e	B.4

8. QB3

In the following program, assume the existence of a function `int existVariable(int var.`

The function returns 1 if `var` exists in the current scope (the scope the function `existVariable` is called), and 0 otherwise. This is a special function that will always compile, even if the `var` variable is not identified.

What will happen when one attempts to compile and execute this program ?

```
1.   int max = 15;
2.   void compare(int a){
3.     if (existVariable(d)) {
4.          a = 100;
5.     }
6.     if (a > max) {
7.        printf("%d is greater than %d ", a, max);
8.     }
9.     else {
10.         printf("%d is not greater than %d ", a, max);
11.    }
12.  }
13.  int main () {
14.    int d = 10;
15.    int a = 20;
16.    max = 25;
17.    compare (d);
18.  }
```

a) Compilation error in line 6: `Variable` *`max`* `not declared.`

b) Program will print: `20 is not greater than 25.`

c) Program will print: `10 is not greater than 25.`

d) Program will print: `10 is not greater than 15.`

e) Program will print: `100 is greater than 25.`

Mapping:

Choice	Misconception
a	B.4
b	B.3
c	Right Answer
d	B.2
e	B.1

9. QC1

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 code here
16.     }
17.     // Insert code here
18.  }
```

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

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));`

Mapping:

Choice	Misconception
a	C.1
b	C.2
c	C.3
d	Right Answer

10. QC2

The following function `fibo` returns the `Nth` number in the Fibonacci sequence (n >=1). The Fibonacci sequence is defined by the fact that every number after the first two (defined as 1) is the sum of the two preceding ones.

Therefore, the first 10 numbers in the Fibonacci sequence are: 1, 1, 2, 3, 5, 8, 13, 21, 34 and 55.

Consider the following program:

```
1.   #include<stdio.h>
2.
3.   int fibo (int n) {
4.     if ((n == 1) || (n==2)){
5.       // Insert code here
6.     }
7.     // Insert code here
8. }
9.   int main(){
10.        int n, result;
11.        printf("\n Enter a positive number: ");
12.        scanf("%d",&n);
13.        result = fibo(n);
14.        printf("\n The %dth Fibonacci number is: %d",result);
15.        return 0;
16.  }
```

The correct code to be inserted on lines 5, and 7:

a) LINE 5: `return 1;`
   LINE 7: `return (fibo (n-1) + fibo (n-2));`

b) LINE 5: `return 1;`
   LINE 7: `return ((n-1) + (n-2));`

c) LINE 5: `// blank`
   LINE 7: `return (fibo (n-1) + fibo (n-2));`

d) LINE 5: `return 1;`
   LINE 7: `return (fibo (n-1));`

Mapping:

Choice	Misconception
a	Right Answer
b	C.2
c	C.3
d	C.1

11. QC3

Consider the following code:

```
1.  #include<stdio.h>
2.
3.   int sumNumbers (int n){
4.     if(n == 0) {
5.       // Insert code here
6.     }
7.     // Insert code here
8.
9.   }
10.
11.  int main(){
12.    int num, sum;
13.    printf("\n Enter a number: ");
14.    scanf("%d",&num);
15.    sum = sumNumbers(num);
16.    printf("\n The sum of all numbers from 0 to %d
17.                is: ", num, sum);
18.    return 0;
19.  }
```

Which of the following insertions to the code will generate a program that prints the sum of all integers from 0 to `n`?

a) LINE 5: `// blank`
   LINE 7: `return(n + sumNumbers (n-1));`

b) LINE 5: `return n;`
   LINE 7: `return(n + sumNumbers (n-1));`

c) LINE 5: `return n;`
   LINE 7: `return(n + (n-1));`

d) LINE 5: `return 1;`
   LINE 7: `return(sumNumbers (n-1));`

Mapping:

Choice	Misconception
a	C.3
b	Right Answer
c	C.2
d	C.1

12. **QD1**

Which one of the following codes correctly prints the sum of all integers 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);
```

d)

```
1.  int sum = 0;
2.  int array[9] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
3.  int i = 1;
4.  while (i <= 10)
5.    sum = sum + array[i];
6.    i++;
7.  }
8.  printf ("The sum is %d", sum);
```

e)

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

Mapping:

Choice	Misconception
a	D.1
b	D.2
c	Right Answer
d	D.3
e	D.4

13. QD2

By definition, a number `n` is prime if it has only 2 distinct divisors: 1 and `n`. Therefore, the number 1 is not prime, but 2, 3 and 5 are. The simplest (but not the most efficient) algorithm to detect whether a number `n` is prime is to check all numbers between 2 (inclusive) and `n -1` (inclusive) looking for some number that divides `n`. In C language, a number `i` divides `n` if the remainder of the division of `n` by `i` is 0, *i.e*, `if n%i == 0`.

The following function `isPrime` was designed to return 1 if the number `n` is prime and 0 otherwise.

```
1. int isPrime (int n) {
2.
3.   int prime = 1, i;
4.
5.   if (n < 2)  return 0;
6.   if (n == 2) return 1;
7.
8.   // Insert code here
9.     if (n%i == 0) {
10.      prime = 0;
11.      }
12.    }
13.     return prime;
14. }
```

The correct code to be inserted in line 8 is:

```
a) if (n > 2){

b) for (i = 2; i < n; i++){

c) for (i = 2; i < n; i++){
     if (isPrime == 1)
       printf ("Number %d is prime");
     else
       printf ("Number %d is not prime");

d) for (i = 2; i <= n; i++){

e) for (i = 1; i < n-1; i++){
```

Mapping:

Choice	Misconception
a	D.5
b	Right Answer
c	D.2
d	D.4
e	D.3

14. QD3

By definition, a number `n` is prime if it has only 2 distinct divisors: 1 and `n`. Therefore, the number 1 is not prime, but 2, 3 and 5 are. The simplest (but not most efficient) algorithm to detect if a number `n` is prime is to check all numbers between 2 (inclusive) and `n -1` (inclusive) looking for some number that divides `n`.

In C language, a number `i` divides `n` if the remainder of the division of `n` by `i` is 0, *i.e.*, `if n%i == 0`.

The following function `int isDivisible(int a, int b)` was designed to return 1 if the number `a` is divisible by `b`.

```
1. int isDivisible(int a, int b) {
2.   if (a % b == 0) return 1;
3.   else return 0;
4. }
```

The program below reads a number `n` and prints if `n` is prime or not.

```
1. int main() {
2.    int n;
3.    int i = 2;
4.    // Insert code here
5.    scanf ("%d", &n);
6.    // Insert code here
7.       foundDivisible = isDivisible(n, i);
8.       i++;
9.    }
10.  if (!foundDivisible)
11.     printf("Number %d is prime", n);
12.   else
13.     printf("Number %d is not prime, n);
14.  return 0;
15.  }
```

Consider the number `n` will always be equal or greater than 2.
The correct code to be inserted is:

```
a) Line 4: int foundDivisible = 1;

   Line 6: while (!foundDivisible && i < n) {
```

b)
```
Line 4: int foundDivisible = 0;
Line 6: if (!foundDivisible && i < n) {
```

c)
```
Line 4: int foundDivisible = 0;
Line 6: for (i = 1; i < n-1; i++) {
            if (foundDivisible) break;
```

d)
```
Line 4: int foundDivisible = 0;
Line 6: while (!foundDivisible && i < n) {
```

Mapping:

Choice	Misconception
a	D.6
b	D.5
c	D.3
d	Right Answer

15. QD4

The following program was designed to read two integers, `a` and `b`, and print all numbers between them. For example, if `a` = 5 and `b` = 10, the program would print `6,7,8,9`, from least to greatest. In turn, if a=10 and b=5, the program would still print `6,7,8,9`.

```
1. int main() {
2.    int a, b, i, small, large;
3.    scanf("%d %d", &a,&b);
4.    if (a < b) {
5.      small = a;
6.      large = b;
7.    }
8.    else {
9.      small = b;
10.     large = a;
11.   }
12.  // Insert code here
13.  // Insert code here
14.    printf("%d", i);
15.    // Insert code here
16.    }
17. }
```

The correct code to be inserted is:

a)
```
Line 12: i = small + 1;
Line 13: while (i < large) {
Line 15: i++;
```

b)
```
Line 12: i = small + 1;
Line 13: while (i < large) {
Line 15: i = a + 1;
```

c)
```
Line 12: i = a;
Line 13: while (a < b) {
Line 15: i = i + 1;
```

d) Line 12: i = small;

Line 13: while (i < large) {

Line 15: i++;

Mapping:

Choice	Misconception
a	Right Answer
b	D.1
c	D.6
d	D.4

16. QD5

A program was designed to calculate the sum of all numbers from 1 (inclusive) to 10 (inclusive) and print the result. A correct implementation of that program would be:

a)

```
1. int main ( ){
2.   int i = 1;
3.   int sum = 0;
4.   while (i <= 10)
5.     sum = sum + i;
6.     i = 1;
7.   }
8.   printf ("The sum is: %d", sum);
9. }
```

b)

```
1. int main ( ){
2.   int i = 1;
3.   int sum = 0;
4.   while (i <= 10)
5.     sum = sum + i;
6.     printf ("The sum is: %d", sum);
7.     i = i + 1;
8.   }
9.   printf ("The sum is: %d", sum);
10.   }
```

c)

```
1. int main ( ){
2.   int i = 1;
3.   int sum = 0;
4.   while (i < 11)
5.     sum = sum + i;
6.     i = i + 1;
7.   }
8.   printf ("The sum is: %d", sum);
9. }
```

d)

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

e)

```
1. int main ( ){
2.   int i;
3.   int sum = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10;
4.   for (i = 1; i <= 10; i++)
5.     sum = sum + i;
6.   }
7.   printf ("The sum is: %d", sum);
8. }
```

Mapping:

Choice	Misconception
a	D.1
b	D.2
c	Right Answer
d	D.5
e	D.6

17. QE1

Consider the `struct`:

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

Which is the correct `if` statement that verifies if two variables of type `Date` (`dateA` and `dateB`) have exactly the 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->day && dateB->month == dateA->day
&& dateB->year == dateA->year)
```

e)
```
if (dateB[day] == dateA[day] && dateB[month] == dateA[day]
&& dateB[year] == dateA[year])
```

Mapping:

Choice	Misconception
a	E.1
b	Right Answer
c	E.2
d	E.3
e	E.4

18. QE2

Consider the following `struct`:

```
1.  struct student {
2.    int bornDay;
3.    int bornMonth;
4.    int bornYear;
5.  };
6.  typedef struct student Student;
```

Consider the following variable declaration statements:

```
Student sA;
Student sB;
```

Which is the correct `if` statement that verifies if two students were born in the same year?

a) `if (Student sA.bornYear == Student sB.bornYear)`

b) `if (sA[bornYear] == sB[bornYear])`

c) `if (sA->bornYear == sB->bornYear)`

d) `if (sA.bornYear == sB)`

e) `if (sA.bornYear == sB.bornYear)`

Mapping:

Choice	Misconception
a	E.5
b	E.4
c	E.3
d	E.2
e	Right Answer

19. QE3

Consider the following code:

```
1.  struct student {
2.    int bornDay;
3.    int bornMonth;
4.    int bornYear;
5.  };
6.  typedef struct student Student;
7.
8.  struct Date {
9.    int day;
10.   int month;
11.   int year;
12.   };
13. typedef struct date Date;
14.
15. Student sA;
16. Date d1;
```

Which is the correct *if* statement that verifies if the student `sA` was born in the year contained in the variable `d1`?

```
a) if (d1[year] == sA [year])
b) if (d1 == sA)
c) if (sA.bornYear == d1)
d) if (Student sA.bornYear == Date d1.year)
e) if (sA.bornYear == d1.year)
```

Mapping:

Choice	Misconception
a	E.4
b	E.1
c	E.2
d	E.5
e	Right Answer

20. QE4

Consider the following code:

```
1.  struct date {
2.    int day;
3.    int month;
4.    int year;
5.  };
6.  typedef struct date Date;
7.
8.  Date d1;
```

Which is the correct `if` statement that checks if the year stored in the variable `d1` is greater than 2000?

```
a) if (d1 > 2000)
b) if (d1 -> year > 2000)
c) if (struct d1.year > 2000)
d) if (d1.year > 2000)
```

Mapping:

Choice	Misconception
a	E.1
b	E.3
c	E.5
d	Right Answer

21. QF1

Consider the following program:

```
1. int main () {
2.     int a = 25, b = 10, result = 0;
3.     int* p;
4.     p = &result;
5.     subtraction (a, b, p, result);
6.     printf ("25 - 10 = %d", result);
7.   }
```

A correct implementation of the `subtraction` function, which will cause the program above to print `25 - 10 = 15` would be:

a)

```
1.   void subtraction (int a, int b, int* p, int result) {
2.     return a - b;
3.   }
```

b)

```
1.   void subtraction (int a, int b, int* p, int result) {
2.     *p = a - b;
3.   }
```

c)

```
1.   void subtraction (int a, int b, int* p, int result) {
2.     result = a - b;
3.   }
```

d)

```
1.   void subtraction (int a, int b, int* p, int result) {
2.     p = a - b;
3.   }
```

e)

```
4.   void subtraction (int a, int b, int* p, int result) {
5.     &p = a - b;
6.   }
```

Mapping:

Choice	Misconception
a	F.4
b	*Right Answer*
c	F.5
d	F.2
e	F.1

22. QF2

The program that prints: `a + b = 30` on the screen is:

a)
```
1. void sumValues (int a, int b) {
2.   return a + b;
3. }
4. int main () {
5.   int a = 10, b = 20, sum = 0;
6.   sum = sumValues (a, b);
7.   printf ("a + b = %d", sum);
8. }
```

b)
```
1.   void sumValues (int a, int b, int sum) {
2.     sum = a + b;
3.   }
4.   int main () {
5.     int a = 10, b = 20, sum = 0;
6.     sumValues (a, b, sum);
7.     printf ("a + b = %d", sum);
8.   }
```

c)
```
1.   void sumValues (int a, int b, int* sum) {
2.     sum = a + b;
3.   }
4.   int main () {
5.     int a = 10, b = 20, sum = 0;
6.     sumValues (a, b, &sum);
7.     printf ("a + b = %d", sum);
8.   }
```

d)
```
1.   void sumValues (int a, int b, int* sum) {
2.     &sum = a + b;
3.   }
4.   int main () {
5.     int a = 10, b = 20, sum = 0;
6.     sumValues (a, b, &sum);
7.     printf ("a + b = %d", sum);
8.   }
```

e)

```
9.   void sumValues (int a, int b, int* sum) {
10.      *sum = a + b;
11.    }
12.    int main () {
13.      int a = 10, b = 20, sum = 0;
14.      sumValues (a, b, &sum);
15.      printf ("a + b = %d", sum);
16.    }
```

Mapping:

Choice	Misconception
a	F.4
b	F.5
c	F.3
d	F.1
e	Right Answer

23. QF3

Consider the following program:

```
1. int main () {
2.   int a = 10;
3.   int* p;
4.
5.   // Add code here
6.
7.   printf (" a = %d", a);
8.  }
```

The correct code that should be inserted in line 5 to make the program print `a = 20` is:

```
a) p = &a;
   *p = a + 10;
```

```
b) p = &a;
   &p = a + 10;
```

```
c) p = &a + 10;
```

```
d) p = &a;
    *p = p + 10;
```

Mapping:

Choice	Misconception
a	Right Answer
b	F.1
c	F.3
d	F.2

24. QF4

The program that prints `result = 110` is:

a)

```
1.   void addTen (int* p) {
2.     *p = p + 10;  }
3.
4.   int main () {
5.     int a = 100;
6.     addTen (&a);
7.     printf (" result = %d", a);
8.   }
```

b)

```
1.   void addTen (int* p) {
2.     *p = *p + 10;
3.   }
4.
5.   int main () {
6.     int a = 100;
7.     addTen (&a);
8.     printf (" result = %d", a);
9.   }
```

c)

```
1.   void addTen (int* p) {
2.     p = *p + 10;
3.   }
4.
5.   int main () {
6.     int a = 100;
7.     addTen (&a);
8.     printf (" result = %d", a);
9.   }
```

d)

```
1.   void addTen (int* p) {
2.     return *p + 10;
3.   }
4.
5.   int main () {
6.     int a = 100;
7.     a = addTen (&a);
8.     printf (" result = %d", a);
9.   }
```

e)

```
1.   void addTen (int p) {
2.     p = p + 10;
3.   }
4.
5.   int main () {
6.     int a = 100;
7.     addTen (a);
8.     printf (" result = %d", a);
9.   }
```

Mapping:

Choice	Misconception
a	F.2
b	Right Answer
c	F.3
d	F.4
e	F.5

25. QG1

The Revised Julian calendar adds an extra day to February in years that are divisible by four, except for years that are divisible by 100 **and** 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 **single** boolean expression 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)
```
int isLeapYear = false;
if (y%4==0) {
   isLeapYear = true;
   // All years divided by 4 are leap years, except:
   if (y%100 == 0) {
     if (y%900 != 200 || y%900 != 600) {
       // A year that is divisible by 100 and do not
       // leave a remainder of 200 or 600 when divided by
       // 900 is not a leap year.
       isLeapYear = false;
     }
   }
}
```

d)
```
isLeapYear = false;
while (y%4 == 0) {
   if (y%100 == 0) && (y%900 != 200 || y%900 != 600) {
     isLeapYear = true;
   }
}
```

e)
```
(y+4+100-200-600)%900 == 0
```

Mapping:

Choice	Misconception
a	Right Answer
b	G.1
c	G.2
d	G.4
e	G.3

26. QG2

Consider the statements `S1`, `S2` and `S3`.
The variables `v1`, `v2` and `v3` are `int` values (0 or 1) that represents if the statements (respectively `S1`, `S2` and `S3`) are true or false. The value 0 indicates the statement is false, and the number 1 indicates the statement is true.

For example, if the statement `S1` is true, the value of `v1` is 1; if the statement `S2` is false, the value of `v2` is 0; and if the statement `S3` is true, the value of `v3` is 1.

The **single** boolean expression which identifies whether *at least* two statements are true, without the use of any arithmetic operator, is:

```
a) (v1 == 1 && v2 == 1) ||
   (v1 == 1 && v3 == 1) ||
   (v2 == 1 && v3 == 1)
b) (v1 + v2 + v3 >= 2)
c)
int atLeast2AreTrue = 0;
if (v1 == 1 && v2 == 1) {
  atLeast2AreTrue = 1;
}
else
if (v1 == 1 && v3 == 1) {
  atLeast2AreTrue = 1;
}
else
if (v2 == 1 && v3 == 1) {
  atLeast2AreTrue = 1;
}
```

d)
```
v1 == 1 && v2 == 1 ||
v1 == 1 && v3 == 1 ||
v2 == 1 && v3 == 1
```

e)
```
int atLeast2AreTrue = 0;
while (v1 == 1 || v2 == 1 || v3 == 1) {
  atLeast2AreTrue = 1;
}
```

Mapping:

Choice	Misconception
a	Right Answer
b	G.3
c	G.2
d	G.1
e	G.4

27. QG3

Consider the following definition to leap year: *"A leap year is any year that is divisible by 4. This rule applies to all years, except for century years (years ending in 00). A century year is only a leap year if it is divisible by 400."*

Consider `year` to be an `int` value representing a year. The **single** boolean expression which identifies if `year` is a leap year is:

a)
```
year % 400 == 0 || year%4 == 0 && year%100 != 0
```

b)
```
int isLeapYear = 0;
if (year % 400 == 0) {
  isLeapYear = 1;
}
else
if (year%4 == 0) {
  if (year%100 != 0){
    isLeapYear = 1;
  }
}
```

c)
```
(year + 400 + 4) % 100 != 0
```

d)
```
(year%4 == 0 && year%100 != 0) || year % 400 == 0
```

e)
```
int isLeapYear = 0;
while (year%400 == 0) {
  if (year%4 == 0 && year%100 != 0) {
    isLeapYear = 1;
  }
}
```

Mapping:

Choice	Misconception
a	G.1
b	G.2
c	G.3
d	Right Answer
e	G.4

4. Concept Inventory (Portuguese Version)

As 27 questões do CI foram numeradas de 1 a 27. Cada questão recebeu um ID único, no formato QTX: T indicando o tópico (A...G), como descrito na Seção 2 deste documento, e X representando o número da questão naquele tópico. Por exemplo, a questão QA1 é a primeira questão do tópico A, enquanto a questão QC2 é a segunda questão do tópico C.

Ainda, cada questão possui uma tabela de mapeamento, indicando o problema de compreensão (*misconception*) associado a cada possível resposta. O trabalho anterior [5] apresenta os detalhes de cada um dos *misconceptions.*

A Figura 3 mostra um exemplo de mapeamento para a questão QA1:

Mapeamento:

Choice	Misconception
a	A.1
b	A.2
c	Resposta Correta
d	A.3

Figura 3. Mapeamento das respostas da questão QA1. Os identificadores A.1, A.2 e A.3 indicam os *misconceptions* associados às alternativas a), b) e d), respectivamente. O trabalho anterior [5] apresenta a descrição em detalhes de cada um desses *misconceptions*.

1. QA1

O código a seguir apresenta uma função que adiciona cinco a um número. Essa função é chamada pelo `main`:

```
1.    int somaCincoAoNum (int n) {
2.       int c = 0;
3.       // Insira o código aqui
4.       return c;
5.    }
6.
7.    int main () {
8.       int x = 0;
9.       x = somaCincoAoNum (x);
10.      return 0;
11.   }
```

O código correto a ser inserido na linha 3 é:

a) `scanf ("%d", &n);`

b) `n = n + 5;`

c) `c = n + 5;`

d) `c = x + 5;`

Mapeamento:

Choice	Misconception
a	A.1
b	A.2
c	Resposta Correta
d	A.3

2. QA2

O código a seguir apresenta a função `subtrairNumeros`, que realiza a subtração de dois números inteiros. Essa função é chamada pelo `main`:

```
1.   int subtrairNumeros (int b, int a) {
2.      int c = 0;
3.      c = b - a;
4.      return c;
5.   }
6.
7.   int main () {
8.      int a = 20;
9.      int b = 10;
10.     // Insira o código aqui
11.     printf ("20 - 10 = %d", c);
12.     return 0;
13.  }
```

O código correto a ser inserido na linha 10 é:

a) `int c = subtrairNumeros (a, b);`

b) `int c = subtrairNumeros (b, a);`

c) `subtrairNumeros (a, b);`

d) `c = subtrairNumeros (a, b);`

e) `int c = subtrairNumeros (b-a, a);`

Mapeamento:

Choice	Misconception
a	Resposta Correta
b	A.4
c	A.5
d	A.3
e	A.6

3. QA3

Considere a seguinte função `main`, que exibe o resultado de `2a - b` **se** `a >= b` e exibe `b - a` caso contrário. A função `func` é utilizada para gerenciar as subtrações.

```
1.  int func(int d, int e){
2.    // Insira o código aqui
3.    return d - e;
4.  }
5.
6.  int main(){
7.     int a = 5;
8.     int b = 7;
9.     int r;
10.    if (a >= b){
11.         // Insira o código aqui
12.    }
13.    else {
14.      r = func(b, a);
15.    }
16.    printf("%d ", r);
17.    return 0;
18.  }
```

O código correto a ser inserido é:

a) Linha 2: //Sem código.
   Linha 11: r = func(a, 2*b);

b) Linha 2: //Sem código
   Linha 11: func(2*a, b);

c) Linha 2: scanf("%d %d", &d, &e);
   Linha 11: r = func(2*a, b);

d) Linha 2: d = 2 * a;
   Linha 11: r = func(a, b);

e) Linha 2: //Sem código
   Linha 11: r = func(2*a, b);

Mapeamento:

Choice	Misconception
a	A.6
b	A.5
c	A.1
d	A.3
e	Resposta Correta

4. QA4

Considere o seguinte código:

```
1.   int main(){
2.     int x = 10;
3.     printf("x é %d ", x);
4.     x = somaDez(x);
5.     printf("x mais 10 é %d ", x);
6.     return 0;
7. }
```

Uma implementação correta da função `somaDez` seria:

```
a) int somaDez (int n) {
      scanf ("%d", &n);
      return n + 10;
   }

b) int somaDez (int n) {
      n = n + 10;
      return n;
   }

c) void somaDez (int n) {
      n = n + 10;
   }

d) int somaDez (int n) {
      int nMaisDez = n;
      return nMaisDez;
   }
```

Mapeamento:

Choice	Misconception
a	A.1
b	Resposta Correta
c	A.2
d	A.4

5. QA5

Um exemplo de programa que lê dois valores (`a` e `b`) e exibe na tela o resultado da subtração `a - b` é:

a)

```
1.   void subtracao (int a, int b, int r) {
2.      r = a - b;
3.   }
4.
5.   int main(){
6.     int a, b, r;
7.     scanf("%d %d", &a, &b);
8.     subtracao(a, b, r);
9.     printf("a - b = %d", r);
10.    return 0;
11.  }
```

b)

```
1.  int subtracao (int a, int b) {
2.    return a - b;
3.  }
4.
5.  int main(){
6.    int a, b, r;
7.    scanf ("%d %d", &a, &b);
8.    r = subtracao(b, a);
9.    printf("a - b = %d", r);
10.   return 0;
11. }
```

c)

```
1.   int subtracao (int n1, int n2) {
2.     return n1 - n2;
3.   }
4.
5.   int main(){
6.     int a, b, r;
7.     scanf("%d %d", &a, &b);
8.     r = subtracao(a, b);
9.     printf("a – b = %d", r);
10.    return 0;
11.  }
```

d)

```
1.   int subtracao (int a, int b) {
2.     return a - b;
3.   }
4.
5.   int main(){
6.     int a, b, r;
7.     scanf("%d %d", &a, &b);
8.     subtracao(a, b);
9.     printf("a – b = %d", r);
10.    return 0;
11.  }
```

e)

```
1.   int subtracao (int a, int b) {
2.     return a - b;
3.   }
4.
5.   int main(){
6.     int a, b, r;
7.     scanf("%d %d", &a, &b);
8.     r = subtracao(a - b, b);
9.     printf("a – b = %d", r);
10.    return 0;
11.  }
```

Mapeamento:

Choice	Misconception
a	A.2
b	A.4
c	Resposta Correta
d	A.5
e	A.6

6. QB1

Considere o seguinte código:

```
1.    #include <stdio.h>
2.
3.    void exibeNumero (int a, int c);
4.
5.    int g = 6;
6.
7.    int main(){
8.      g = 10;
9.      int a = 20;
10.      int b = 30;
11.      int c = 50;
12.      int d = 100;
13.      exibeNumero(a,b);
14.   }
15.
16.   void exibeNumero (int a, int c) {
17.       int resultado = a + c;
18.       resultado = resultado + g;
19.       printf("Valor = %d", resultado);
20.    }
```

O valor que será exibido na tela é:

a) 60

b) 56

c) O programa não irá compilar. Erro nas linhas 8 e 18: variável *g* não foi declarada.

d) 80

e) Se na linha 18 tivéssemos resultado = resultado + d; o valor exibido seria 150.

Mapeamento:

Choice	Misconception
a	Resposta Correta
b	B.2
c	B.4
d	B.3
e	B.1

7. QB2

Nos seguintes programas, assuma a existência da função `int existeVariavel(int var)`.

Essa função retorna 1 se `var` existe no escopo atual (o escopo em que a função `existeVariavel` é chamada), e 0 caso contrário. Essa é uma função especial que **sempre irá compilar**, mesmo que a variável `var` não seja identificada.

O código que imprime a frase: "`r = 22`" é:

a)

```
1.   int func(int a, int b){
2.     return a + b + c;
3.   }
4.
5.   int main(){
6.     int a = 5, b = 7, c = 10;
7.     int r = func (a, b);
8.     printf ("r = %d ", r);
9.     return 0;
10.  }
```

b)

```
1.  int func(int a, int b, int c){
2.     return a + b + c;
3.   }
4.
5.   int main(){
6.     int a = 5, b = 7, c = 10;
7.     int r = func (a, b, c);
8.     printf ("r = %d ", r);
9.     return 0;
10.  }
```

c)

```
1.  int func(int a, int b, int c){
2.     return a + b + c;
3.   }
4.
5.   int main (){
6.     int a = 7, b = 13, c = 2;
7.     int r = func (a, b, b);
8.     printf ("r = %d ", r);
9.     return 0;
10.  }
```

d)

```
1.  int z = 0;
2.  int func(int a, int b, int c){
3.     z = 1;
4.     return a + b + c;
5.   }
6.
7.   int main (){
8.     int a = 7, b = 11, c = 4;
9.     int r = func (a, b, c);
10.     r = r + z;
11.     printf ("r = %d ", r);
12.     return 0;
13.   }
```

e)

```
1.  int z = 0;
2.  int func(){
3.    if (existeVariavel(z)) {
4.      // z existe neste escopo
5.      return -1;
6.    }
7.    else {
8.      // z não existe neste escopo
9.      return 22;
10.     }
11.
12.  int main (){
13.     int r = func ();
14.     printf ("r = %d ", r);
15.     return 0;
16.  }
```

Mapeamento:

Choice	Misconception
a	B.1
b	Resposta Correta
c	B.3
d	B.2
e	B.4

8. QB3

No seguinte programa, assuma a existência da função `int existeVariavel(int var)`.

Essa função retorna 1 se `var` existe no escopo atual (o escopo em que a função `existeVariavel` é chamada), e 0 caso contrário. Essa é uma função especial que **sempre irá compilar**, mesmo que a variável `var` não seja identificada.

O que ocorrerá quando alguém tentar compilar e executar o programa?

```
1.   int max = 15;
2.   void compare(int a){
3.     if (existeVariavel(d)) {
4.          a = 100;
5.     }
6.     if (a > max) {
7.        printf("%d é maior que %d ", a, max);
8.     }
9.     else {
10.         printf("%d não é maior que %d ", a, max);
11.    }
12.  }
13.  int main () {
14.    int d = 10;
15.    int a = 20;
16.    max = 25;
17.    compare (d);
18.  }
```

a) Erro de compilação na linha 6: `Variável` *`max`* `não foi declarada.`

b) Programa exibe: `20 não é maior que 25.`

c) Programa exibe: `10 não é maior que 25.`

d) Programa exibe: `10 não é maior que 15.`

e) Programa exibe: `100 é maior que 25.`

Mapeamento:

Choice	Misconception
a	B.4
b	B.3
c	Resposta Correta
d	B.2
e	B.1

9. QC1

Considere o seguinte código:

```
1.   #include<stdio.h>
2.
3.   int fat(int);
4.
5.   int main(){
6.       int num,f;
7.       printf("\n Digite um número: ");
8.       scanf("%d",&num);
9.       f=fat(num);
10.       printf("\n Fatorial de %d é: %d",num,f);
11.       return 0;
12.  }
13.  int fat(int n){
14.     if(n <= 1) {
15.         // Insira o código aqui
16.     }
17.     // Insira o código aqui
18.  }
```

O código correto a ser inserido nas linhas 15 e 17 é:

a) LINHA 15: `return 1;`
   LINHA 17: `return(n*fat(n+1));`

b) LINHA 15: *`return 1;`*
   LINHA 17: `// em branco`

c) LINHA 15: *`// em branco`*
   LINHA 17: `return(n*fat(n-1));`

d) LINHA 15: `return 1;`
   LINHA 17: `return(n*fat(n-1));`

Mapeamento:

Choice	Misconception
a	C.1
b	C.2
c	C.3
d	Resposta Correta

10. QC2

A seguinte função `fibo` retorna o *n*-ésimo número da sequência de Fibonacci (n >=1). A sequência de Fibonacci é definida pelo fato de que cada número depois dos dois primeiros (definidos como 1) são a soma dos dois números precedentes.

Deste modo, os primeiros 10 números da sequência de Fibonacci são: 1, 1, 2, 3, 5, 8, 13, 21, 34 e 55.

Considere o seguinte programa:

```
1.   #include <stdio.h>
2.
3.   int fibo (int n) {
4.      if ((n == 1) || (n==2)){
5.         // Insira o código aqui
6.      }
7.      // Insira o código aqui
8.   }
9.   int main(){
10.     int n, resultado;
11.     printf("\n Digite um numero positivo: ");
12.     scanf("%d",&n);
13.     resultado = fibo(n);
14.     printf("O %d-esimo numero de Fibonacci eh:
        %d",resultado);
15.     return 0;
16.  }
```

O código correto a ser inserido nas linhas 5 e 7 é:

a) LINHA 5: `return 1;`
   LINHA 7: `return (fibo (n-1) + fibo (n-2));`

b) LINHA 5: `return 1;`
   LINHA 7: `return ((n-1) + (n-2));`

c) LINHA 5: `// em branco`
   LINHA 7: `return (fibo (n-1) + fibo (n-2));`

d) LINHA 5: `return 1;`
   LINHA 7: `return (fibo (n-1));`

Mapeamento:

Choice	Misconception
a	Resposta Correta
b	C.2
c	C.3
d	C.1

11. QC3

Considere o seguinte código:

```
1.  #include<stdio.h>
2.
3.   int somaNumeros (int n){
4.     if(n == 0) {
5.       // Insira o código aqui
6.     }
7.     // Insira o código aqui
8.
9.   }
10.
11.  int main(){
12.    int num, soma;
13.    printf("\n Digite um numero: ");
14.    scanf("%d",&num);
15.    soma = somaNumeros(num);
16.    printf("\n A soma de todos os numeros de 0 a %d
17.                eh: ", num, soma);
18.    return 0;
19.  }
```

Qual das seguintes inserções de código irá gerar um programa que imprime a soma de todos os inteiros de 0 a n?

a) LINHA 5: `// em branco`
   LINHA 7: `return(n + somaNumeros (n-1));`

b) LINHA 5: `return n;`
   LINHA 7: `return(n + somaNumeros (n-1));`

c) LINHA 5: `return n;`
   LINHA 7: `return(n + (n-1));`

d) LINHA 5: `return 1;`
   LINHA 7: `return(somaNumeros (n-1));`

Mapeamento:

Choice	Misconception
a	C.3
b	Resposta Correta
c	C.2
d	C.1

12. QD1

Qual dos seguintes códigos exibe corretamente a soma de todos os inteiros de 0 a 9?

a)

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

b)

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

c)

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

d)

```
1.  int soma = 0;
2.  int vetor[9] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
3.  int i = 1;
4.  while (i <= 10)
5.    soma = soma + vetor[i];
6.    i++;
7.  }
8.  printf ("A soma eh %d", soma);
```

e)

```
1.  int soma = 0;
2.  for (int i = 0; i <= 10; i++) {
3.    soma = soma + i;
4.  }
5.  printf ("A soma eh %d", soma);
```

Mapeamento:

Choice	Misconception
a	D.1
b	D.2
c	Resposta Correta
d	D.3
e	D.4

13. QD2

Por definição, um número `n` é primo se ele possui apenas 2 divisores: 1 e `n`. Deste modo, por exemplo, o número 1 não é primo (possui apenas um divisor), mas 2, 3 e 5 são primos. Uma das formas mais simples (mas não mais eficiente) de verificar se um número `n` é primo é percorrer todos os números entre 2 (inclusive) e `n-1` (inclusive), procurando por algum número que divide `n`.

Na linguagem C, um número `i` divide `n` se o resto da divisão de `n` por `i` é 0, *i.e.,* `if n%i == 0`.

A função a seguir `ehPrimo` foi criada para retornar 1 se o número `n` passado como parâmetro é primo, e 0 caso contrário.

```
1. int ehPrimo (int n) {
2.
3.   int primo = 1, i;
4.
5.   if (n < 2)  return 0;
6.   if (n == 2) return 1;
7.
8.   // Insira o código aqui
9.     if (n%i == 0) {
10.      primo = 0;
11.      }
12.    }
13.     return primo;
14. }
```

O código correto a ser inserido na linha 8 é:

```
a) if (n > 2){

b) for (i = 2; i < n; i++){

c) for (i = 2; i < n; i++){
      if (ehPrimo == 1)
          printf ("Numero %d eh primo");
      else
          printf ("Numero %d nao eh primo");

d) for (i = 2; i <= n; i++){

e) for (i = 1; i < n-1; i++){
```

Mapeamento:

Choice	Misconception
a	D.5
b	Resposta Correta
c	D.2
d	D.4
e	D.3

14. QD3

Por definição, um número `n` é primo se ele possui apenas 2 divisores: 1 e `n`. Deste modo, por exemplo, o número 1 não é primo (possui apenas um divisor), mas 2, 3 e 5 são primos. Uma das formas mais simples (mas não mais eficiente) de verificar se um número `n` é primo é percorrer todos os números entre 2 (inclusive) e `n-1` (inclusive), procurando por algum número que divide `n`.

Na linguagem C, um número `i` divide `n` se o resto da divisão de `n` por `i` é 0, *i.e.,* `if n%i == 0`.

A seguinte função `ehDivisivelPor(int a, int b)` foi criada para retornar 1 se o número `a` é divisível por `b`.

```
1. int ehDivisivelPor (int a, int b) {
2.    if (a % b == 0)
3.      return 1;
4.    else
5.      return 0;
6. }
```

O programa abaixo lê um número `n` e exibe se `n` é ou não primo:

```
1. int main() {
2.    int n;
3.    int i = 2;
4.    // Insira o código aqui
5.    scanf ("%d", &n);
6.    // Insira o código aqui
7.       encontreiDivisivel = ehDivisivelPor(n, i);
8.       i++;
9.    }
10.  if (!encontreiDivisivel)
11.     printf("Numero %d eh primo", n);
12.   else
13.     printf("Numero %d nao eh primo", n);
14.  return 0;
15.  }
```

Considere que o número `n` será sempre igual ou maior do que 2. O código correto a ser inserido nas linhas 4 e 6 é:

a) LINHA 4: `int encontreiDivisivel = 1;`

   LINHA 6: `while (!encontreiDivisivel && i < n) {`

b) LINHA 4: `int encontreiDivisivel = 0;`

LINHA 6: `if (!encontreiDivisivel && i < n) {`

c) LINHA 4: `int encontreiDivisivel = 0;`

LINHA 6: `for (i = 1; i < n-1; i++) {`

`if (encontreiDivisivel) break;`

d) LINHA 4: `int encontreiDivisivel = 0;`

LINHA 6: `while (!encontreiDivisivel && i < n) {`

Mapeamento:

Choice	Misconception
a	D.6
b	D.5
c	D.3
d	Resposta Correta

15. QD4

O programa a seguir foi elaborado para ler dois inteiros, `a` e `b`, e exibir todos os números entre eles. Por exemplo, se `a`=5 e `b`=10, o programa exibiria `6,7,8,9`, do menor ao maior. De modo similar, se `a`=10 e `b` = 5, o programa exibiria a mesma sequência `6,7,8,9`.

```
1. int main() {
2.    int a, b, i, pequeno, grande;
3.    scanf("%d %d", &a,&b);
4.    if (a < b) {
5.      pequeno = a;
6.      grande = b;
7.    }
8.    else {
9.      pequeno = b;
10.      grande = a;
11.   }
12.  // Insira o código aqui
13.  // Insira o código aqui
14.    printf("%d", i);
15.    // Insira o código aqui
16.  }
17. }
```

O código correto a ser inserido nas linhas 12, 13 e 15 é:

a) LINHA 12: `i = pequeno + 1;`

   LINHA 13: `while (i < grande) {`

   LINHA 15: `i++;`

b) LINHA 12: `i = pequeno + 1;`

   LINHA 13: `while (i < grande) {`

   LINHA 15: `i = a + 1;`

c) LINHA 12: `i = a;`

LINHA 13: `while (a < b) {`

LINHA 15: `i = i + 1;`

d) LINHA 12: `i = pequeno;`

LINHA 13: `while (i < grande) {`

LINHA 15: `i++;`

Mapeamento:

Choice	Misconception
a	Resposta Correta
b	D.1
c	D.6
d	D.4

16. QD5

Um programa foi elaborado para calcular a soma de todos os números entre 1 (inclusive) e 10 (inclusive) e exibir o resultado. Uma correta implementação deste programa é:

```
1. int main (){
2.   int i = 1;
3.   int soma = 0;
4.   while (i <= 10){
5.     soma = soma + i;
6.     i = 1;
7.   }
8.   printf ("A soma eh: %d", soma);
9. }
```

b)

```
1.   int main (){
2.      int i = 1;
3.      int soma = 0;
4.      while (i <= 10){
5.        soma = soma + i;
6.        printf ("A soma eh: %d", soma);
7.        i = i + 1;
8.      }
9.      printf ("A soma eh: %d", soma);
10.  }
```

c)

```
1. int main (){
2.   int i = 1;
3.   int soma = 0;
4.   while (i < 11){
5.     soma = soma + i;
6.     i = i + 1;
7.   }
8.   printf ("A soma eh: %d", soma);
9. }
```

d)

```
1. int main (){
2.   int i = 1;
3.   int soma = 0;
4.   soma = soma + i;
5.   i = i + 1;
6.   printf ("A soma eh: %d", soma);
7. }
```

e)

```
1. int main (){
2.   int i;
3.   int soma = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10;
4.   for (i = 1; i <= 10; i++){
5.     soma = soma + i;
6.   }
7.   printf ("A soma eh: %d", soma);
8. }
```

Mapeamento:

Choice	Misconception
a	D.1
b	D.2
c	Resposta Correta
d	D.5
e	D.6

17. QE1

Considere a seguinte `struct`:

```
1.   struct data {
2.     int dia;
3.     int mes;
4.     int ano;
5.   };
6.   typedef struct data Data;
```

Qual é o correto código `if` que verifica se duas variáveis do tipo `Data` (`dataA` e `dataB`) têm exatamente a mesma data?

a)
```
if (dataA == dataB)
```

b)
```
if (dataA.dia == dataB.dia && dataA.mes == dataB.mes &&
dataA.ano == dataB.ano)
```

c)
```
if (dataA.dia == dataB && dataA.mes == dataB && dataA.ano ==
dataB)
```

d)
```
if (dataB->dia == dataA->dia && dataB->mes == dataA->mes &&
dataB->ano == dataA->ano)
```

e)
```
if (dataB[dia] == dataA[dia] && dataB[mes] == dataA[mes] &&
dataB[ano] == dataA[ano])
```

Mapeamento:

Choice	Misconception
a	E.1
b	Resposta Correta
c	E.2
d	E.3
e	E.4

18. QE2

A struct `aluno` foi definida da seguinte forma:

```
1.   struct aluno {
2.     int diaNasc;
3.     int mesNasc;
4.     int anoNasc;
5.   };
6.   typedef struct aluno Aluno;
```

Considere as seguintes declarações de variáveis:

```
Aluno al_A;
Aluno al_B;
```

Qual é o correto código `if` que verifica se dois estudantes nasceram no mesmo ano?

```
a) if (Aluno al_A.anoNasc == Aluno al_B.anoNasc)

b) if (al_A[anoNasc] == al_B[anoNasc])

c) if (al_A->anoNasc == al_B->anoNasc)

d) if (al_A.anoNasc == al_B)

e) if (al_A.anoNasc == al_B.anoNasc)
```

Mapeamento:

Choice	Misconception
a	E.5
b	E.4
c	E.3
d	E.2
e	Resposta Correta

19. QE3

Considere o seguinte código:

```
1.   struct estudante {
2.      int nascimentoDia;
3.      int nascimentoMes;
4.      int nascimentoAno;
5.   };
6.   typedef struct estudante Estudante;
7.
8.   struct Data {
9.      int dia;
10.     int mes;
11.     int ano;
12.  };
13.  typedef struct data Data;
14.
15.  Estudante eA;
16.  Data d1;
```

Qual é o correto código `if` que verifica se o estudante `eA` nasceu no ano contido na variável `d1`?

a) `if (d1[ano] == eA[ano])`
b) `if (d1 == eA)`
c) `if (eA.nascimentoAno == d1)`
d) `if (Estudante eA.nascimentoAno == Data d1.ano)`
e) `if (eA.nascimentoAno == d1.ano)`

Mapeamento:

Choice	Misconception
a	E.4
b	E.1
c	E.2
d	E.5
e	Resposta Correta

20. QE4

Considere o seguinte código:

```
1.   struct data {
2.     int dia;
3.     int mes;
4.     int ano;
5.   };
6.   typedef struct data Data;
7.
8.   Data d1;
```

Qual é o correto código `if` que verifica se o ano armazenado na variável `d1` é maior do que 2000?

```
a) if (d1 > 2000)
b) if (d1->ano > 2000)
c) if (struct d1.ano > 2000)
d) if (d1.ano > 2000)
```

Mapeamento:

Choice	Misconception
a	E.1
b	E.3
c	E.5
d	Resposta Correta

21. QF1

Considere o seguinte programa:

```
1. int main () {
2.     int a = 25, b = 10, resultado = 0;
3.     int* p;
4.     p = &resultado;
5.     subtrai (a, b, p, resultado);
6.     printf ("25 – 10 = %d", resultado);
7. }
```

Uma implementação correta da função `subtrai`, que faria o programa acima exibir `25 – 10 = 15` é:

a)
```
1.   void subtrai (int a, int b, int* p, int resultado) {
2.     return a – b;
3.   }
```

b)
```
1.   void subtrai (int a, int b, int* p, int resultado) {
2.     *p = a - b;
3.   }
```

c)
```
1.   void subtrai (int a, int b, int* p, int resultado) {
2.     resultado = a - b;
3.   }
```

d)
```
1.   void subtrai (int a, int b, int* p, int resultado) {
2.     p = a - b;
3.   }
```

e)
```
1.   void subtrai (int a, int b, int* p, int resultado) {
2.     &p = a - b;
3.   }
```

Mapeamento:

Choice	Misconception
a	F.4
b	*Resposta Correta*
c	F.5
d	F.2
e	F.1

22. QF2

O programa que exibe `a + b = 30` na tela é:

a)

```
1.  void somaValores (int a, int b) {
2.    return a + b;
3.  }
4.
5.  int main () {
6.    int a = 10, b = 20, soma = 0;
7.    soma = somaValores (a, b);
8.    printf ("a + b = %d", soma);
9. }
```

b)

```
1.  void somaValores (int a, int b, int soma) {
2.    soma = a + b;
3.  }
4.
5.  int main () {
6.    int a = 10, b = 20, soma = 0;
7.    somaValores (a, b, soma);
8.    printf ("a + b = %d", soma);
9.  }
```

c)

```
1.  void somaValores (int a, int b, int* soma) {
2.    soma = a + b;
3.  }
4.
5.  int main () {
6.    int a = 10, b = 20, soma = 0;
7.    somaValores (a, b, &soma);
8.    printf ("a + b = %d", soma);
9.  }
```

d)

```
1.  void somaValores (int a, int b, int* soma) {
2.    &soma = a + b;
3.  }
4.
5.  int main () {
6.    int a = 10, b = 20, soma = 0;
7.    somaValores (a, b, &soma);
8.    printf ("a + b = %d", soma);
9.  }
```

e)

```
1.   void somaValores (int a, int b, int* soma) {
2.      *soma = a + b;
3.   }
4.
5.   int main () {
6.     int a = 10, b = 20, soma = 0;
7.     somaValores (a, b, &soma);
8.     printf ("a + b = %d", soma);
9.  }
```

Mapeamento:

Choice	Misconception
a	F.4
b	F.5
c	F.3
d	F.1
e	Resposta Correta

23. QF3

Considere o seguinte programa:

```
1. int main () {
2.   int a = 10;
3.   int* p;
4.
5.   // Insira o código aqui
6.
7.   printf (" a = %d", a);
8.  }
```

O código correto a ser inserido na linha 5, que faça com que o programa exiba na tela `a = 20` é:

```
a) p = &a;
   *p = a + 10;
```

```
b) p = &a;
   &p = a + 10;
```

```
c) p = &a + 10;
```

```
d) p = &a;
    *p = p + 10;
```

Mapeamento:

Choice	Misconception
a	Resposta Correta
b	F.1
c	F.3
d	F.2

24. QF4

O programa que exibe `resultado = 110` é:

a)

```
1.   void somaDez (int* p) {
2.     *p = p + 10;
3.   }
4.
5.   int main () {
6.     int a = 100;
7.     somaDez (&a);
8.     printf ("resultado = %d", a);
9.   }
```

b)

```
1.   void somaDez (int* p) {
2.     *p = *p + 10;
3.   }
4.
5.   int main () {
6.     int a = 100;
7.     somaDez (&a);
8.     printf ("resultado = %d", a);
9.   }
```

c)

```
1.   void somaDez (int* p) {
2.     p = *p + 10;
3.   }
4.
5.   int main () {
6.     int a = 100;
7.     somaDez (&a);
8.     printf ("resultado = %d", a);
9.   }
```

d)

```
1.   void somaDez (int* p) {
2.     return *p + 10;
3.   }
4.
5.   int main () {
6.     int a = 100;
7.     a = somaDez (&a);
8.     printf ("resultado = %d", a);
9.   }
```

e)

```
1.   void somaDez (int p) {
2.     p = p + 10;
3.   }
4.
5.   int main () {
6.     int a = 100;
7.     somaDez (a);
8.     printf ("resultado = %d", a);
9.   }
```

Mapeamento:

Choice	Misconception
a	F.2
b	Resposta Correta
c	F.3
d	F.4
e	F.5

25. QG1

O calendário Juliano Revisado adiciona um dia extra ao mês de fevereiro em todos os anos que são divisíveis por 4, exceto nos anos que são divisíveis por 100 **e** não deixam um resto de 200 **ou** 600 quando divididos por 900. Um ano com um dia extra adicionado em fevereiro é chamado de bissexto.

Considere `y` como um inteiro representando um ano. A expressão booleana **única** que identifica se `y` é um ano bissexto é:

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)
```
int ehBissexto = false;
if (y%4==0) {
   ehBissexto = true;
   // Todos os anos divisíveis por 4 são bissextos, exceto:
   if (y%100 == 0) {
     if (y%900 != 200 || y%900 != 600) {
        // Um ano que é divisível por 100 e não deixa resto
       // de 200 ou 600 quando dividido por 900 não é um
       // ano bissexto.
        ehBissexto = false;
     }
   }
}
```

d)
```
ehBissexto = false;
while (y%4 == 0) {
   if (y%100 == 0) && (y%900 != 200 || y%900 != 600) {
     ehBissexto = true;
   }
}
```

e)
```
(y+4+100-200-600)%900 == 0
```

Mapeamento:

Choice	Misconception
a	Resposta Correta
b	G.1
c	G.2
d	G.4
e	G.3

26. QG2

Considere que `S1`, `S2` e `S3` são afirmações.

As variáveis `v1`, `v2` e `v3` são inteiros (0 ou 1) que representam se as afirmações (respectivamente `S1`, `S2` e `S3`) são verdadeiras ou falsas. O valor 0 indica que uma afirmação é falsa, e o valor 1 que uma afirmação é verdadeira.

Por exemplo, se a afirmação `S1` é verdadeira, o valor de `v1` é 1; se a afirmação `S2` é falsa, o valor de `v2` é 0; e se a afirmação `S3` é verdadeira, o valor de `v3` é 1.

A expressão booleana **única** que verifica se *ao menos* duas afirmações são verdadeiras, sem o uso de nenhum operador aritmético, é:

a)
```
(v1 == 1 && v2 == 1) ||
(v1 == 1 && v3 == 1) ||
(v2 == 1 && v3 == 1)
```

b)
```
(v1 + v2 + v3 >= 2)
```

c)
```
int peloMenosDuasSaoVerdadeiras = 0;
if (v1 == 1 && v2 == 1) {
  peloMenosDuasSaoVerdadeiras = 1;
}
else
if (v1 == 1 && v3 == 1) {
  peloMenosDuasSaoVerdadeiras = 1;
}
else
if (v2 == 1 && v3 == 1) {
  peloMenosDuasSaoVerdadeiras = 1;
}
```

```
d)  v1 == 1 && v2 == 1 ||
    v1 == 1 && v3 == 1 ||
    v2 == 1 && v3 == 1
```

```
e)
int peloMenosDuasSaoVerdadeiras = 0;
while (v1 == 1 || v2 == 1 || v3 == 1) {
  peloMenosDuasSaoVerdadeiras = 1;
}
```

Mapeamento:

Choice	Misconception
a	Resposta Correta
b	G.3
c	G.2
d	G.1
e	G.4

27. QG3

Considere a seguinte definição de ano bissexto: *"Um ano é considerado bissexto se ele é divisível por 4. Esta regra aplica-se a todos os anos, exceto para séculos (anos que terminam em 00). Um século é um ano bissexto apenas se ele é divisível por 400."*

Considere `ano` como um inteiro que representa um ano. A expressão booleana **única** que identifica se `ano` é um ano bissexto é:

a)
```
ano % 400 == 0 || ano % 4 == 0 && ano % 100 != 0
```

b)
```
int ehBissexto = 0;
if (ano % 400 == 0) {
  ehBissexto = 1;
}
else
if (ano % 4 == 0) {
  if (ano % 100 != 0){
    ehBissexto = 1;
  }
}
```

c)
```
(ano + 400 + 4) % 100 != 0
```

d)
```
(ano % 4 == 0 && ano % 100 != 0) || ano % 400 == 0
```

e)
```
int ehBissexto = 0;
while (ano % 400 == 0) {
  if (ano % 4 == 0 && ano % 100 != 0) {
    ehBissexto = 1;
  }
}
```

Mapeamento:

Choice	Misconception
a	G.1
b	G.2
c	G.3
d	Resposta Correta
e	G.4

5. Acknowledgments

This research is supported by grant #2014/07502-4, São Paulo Research Foundation (FAPESP). Additional support was provided by the Brazilian Federal Agency for Support and Evaluation of Graduate Education (CAPES), the National Counsel of Technological and Scientific Development (CNPq) and the University of Campinas (Unicamp).

The opinions, hypotheses, conclusions or recommendations expressed in this material are the responsibility of the author(s) and do not necessarily reflect the views of FAPESP.

We also would like to thank the support of Dr. Marco Aurélio Gerosa, Dra. Christiane Neme Campos, Dr Breno de França, Dr. Kellogg S. Booth and Dr. Steven Wolfman in this research development.

6. References

[1] Mohamed El-Attar and James Miller. 2006. Matching Antipatterns to Improve the Quality of Use Case Models. In *Proceedings of the 14th IEEE International Requirements Engineering Conference* (RE '06). IEEE Computer Society, Washington, DC, USA, 96-105. DOI=http://dx.doi.org/10.1109/RE.2006.42

[2] CACEFFO, R.; WOLFMAN, S.; BOOTH, K. 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=http://dx.doi.org/10.1145/2839509.2844559

[3] MAZUR, E.; CROUCH, C. 2001. Peer Instruction: Ten Years of Experience and Results. In American Journal of Physics 69, 970 (2001); doi: 10.1119/1.1374249. American Association of Physics Teachers.

[4] CACEFFO, R.; WOLFMAN, S.; BOOTH, S.; AZEVEDO, R. A Pilot Test to Validate Misconceptions for Introductory Computer Programming. ***To be published.***

[5] CACEFFO, R. FRANÇA, B.; GAMA, G.; BENATTI, R.; APARECIDA, T.; CALDAS, T.; AZEVEDO, R. (2017) An Antipattern Documentation about Misconceptions related to an Introductory Programming Course in C. In Technical Report 17-15, Institute of Computing, University of Campinas, SP, Brasil. 13 pages. October, 2017

[6] CACEFFO, R.; GAMA, G.; AZEVEDO, R. 2018. Exploring Active Learning Approaches to Computer Science Classes. In Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18). ACM, New York, NY, USA, 922-927. DOI: https://doi.org/10.1145/3159450.3159585

[7] LASRY, N. 2011. The Puzzling Reliability of the Force Concept Inventory. In American Journal of Physics 79, 909 (2011). https://doi.org/10.1119/1.3602073