EXTRA EXERCISES: WEEK 1

EXERCISE 1

The objective of this exercise is to "manually execute" an algorithm. If you have minimum practice working with the classic programming structures you may skip this exercise, it won't provide you any kind of new knowledge.

Taking into account the following pseudocode, write down the value of the variable "a" once the algorithm has been executed.

```
a <= 25;
For i in 0 to 4 loop
    For j in 0 to 4 loop
        If (i=j) then a <= a + 1;
            elsif (j = 4) then a <= a + 2;
            end if;
        end if;
    end loop;
end loop;
a <= a + 10;
```

To clarify the following explanations, let's numerate the pseudocode instructions:

```
1:  a <= 25;
2:  For i in 0 to 4 loop
3:      For j in 0 to 4 loop
4:          If (i=j) then a <= a + 1;
5:              elsif (j = 4) then a <= a + 2;
6:              end if;
7:          end if;
8:      end loop;
9:  end loop;
10: a <= a + 10;
```

Initially "a" is set to 25 and both the indexes i and j are set to 0.

a	i	j	Comments
25	0	0	

The condition inside the "if" statement at line 4 was true (i=j) and as a result "a" is incremented by 1. After that the "for" loop from the lines 3 to 8 is executed again, incrementing j by 1.

©Elena Valderrama, Jean-Pierre Deschamps, Mercè Rullán

a	i	j	Comments
25	0	0	
26	0	0	If (i=j) then a <= a + 1;
26	0	1	end loop (line 8)

The "if" from the line 4 is executed again but this time i≠j and j≠4, so this time no action is performed with "a". As in the previous case, j increments by 1 and the "for" loop from line 3 is executed again.

a	i	j	Comments
25	0	0	
26	0	0	If (i=j) then a <= a + 1;
26	0	1	end loop (line 8)

26	0	2	"a" does not change value because it does not fulfil any of the "if" conditions:
			If (i=j) then a <= a + 1;
			elsif (j = 4) then a <= a + 2;
			and "j" increments by 1 on executing once again the loop of lines 3-8.

This situation happens again for j=3, but once j=4 the "elsif" condition from line 5 complies and "a" is incremented by 2:

a	i	j	Comments
25	0	0	
26	0	0	If (i=j) then a <= a + 1;
26	0	1	end loop (line 8)

26	0	2	The value of "a" isn't changed because none of the condition from the if statement is complied:
			If (i=j) then a <= a + 1;
			elsif (j = 4) then a <= a + 2;
			In addition, "j" is incremented by 1 once the loop from lines 3-8 is executed again.

26	0	3	
28	0	4	elsif (j = 4) then a <= a + 2;

At this point the loop "for" from lines 2 to 9 is executed again incrementing i by 1 and the "for" loop from line 4 is initialized again (j=0):

a	i	j	Comments
25	0	0	
26	0	0	If (i=j) then a <= a + 1;
26	0	1	end loop (line 8)
26	0	2	…
26	0	3	
28	0	4	elsif (j = 4) then a <= a + 2;
28	1	0	

None of the "if" conditions comply so "a" doesn't change its value and j increments by 1.

a	i	j	Comments
25	0	0	
26	0	0	If (i=j) then a <= a + 1;
26	0	1	end loop (line 8)

©Elena Valderrama, Jean-Pierre Deschamps, Mercè Rullán

26	0	2	…
26	0	3	
28	0	4	elsif (j = 4) then a <= a + 2;
28	1	0	
28	1	1	

Now i=j, so “a” and “j” increment by 1. While j=2 or j=3, the value of “a” is not modified.

a	i	j	Comments
25	0	0	
26	0	0	If (i=j) then a <= a + 1;
26	0	1	end loop (line 8)
26	0	2	…
26	0	3	
28	0	4	elsif (j = 4) then a <= a + 2;
28	1	0	
28	1	1	
29	1	2	
29	1	4	

Once j = 4 the “elsif” condition from line 5 complies and “a” increments by 2:

a	i	j	Comments
25	0	0	
26	0	0	If (i=j) then a <= a + 1;
26	0	1	end loop (line 8)
26	0	2	…
26	0	3	
28	0	4	elsif (j = 4) then a <= a + 2;
28	1	0	
28	1	1	
29	1	2	
31	1	4	

We can continue with the same reasoning:

a	i	j	Comments
25	0	0	
…	…	…	…
31	1	4	
31	2	0	
31	2	1	
32	2	2	
32	2	3	
34	2	4	
34	3	0	
34	3	1	
34	3	2	
35	3	3	
37	3	4	
37	4	0	
37	4	1	
37	4	2	
37	4	3	
38	4	4	This time the “if” condition from line 4 (i=j) complies, so “a” increments by 1.

©Elena Valderrama, Jean-Pierre Deschamps, Mercè Rullán

Finally both loops are finished and "a" increments by 10 (line 10). Therefore, the final value of "a" is 48.

a	i	j	Comments
25	0	0	
…	…	…	…
38	4	4	This time the "if" condition in line 4 (i=j) complies, so "a" increments by 1.
48	4	4	a <= a + 10;

[Solution: 48]

EXERCISE 2

Given two numbers X and $Y = y_3 \cdot 10^3 + y_2 \cdot 10^2 + y_1 \cdot 10 + y_0$, and P being the result of $X \cdot Y$ ($P = X \cdot Y$), *we can express P* as $P = (((y_3 \cdot X) \cdot 10 + y_2 \cdot X) \cdot 10 + y_1 \cdot X) \cdot 10 + y_0 \cdot X$. *Which of the following algorithms compute P?*

1) P <=0; for i in 0 to 3 loop P <=(P*10) + (y(i)*X); end loop;	2) P <=0; for i in 2 downto 0 loop P <=(P*10) + (y(i)*X); end loop;
3) P <=0; for i in 3 downto 0 loop P <=(P*10) + (y(i)*X); end loop;	4) P <=y(3)*X; for i in 2 downto 0 loop P <=(P*10) + (y(i)*X); end loop;

Let's analyze, for example, how algorithm 3 works:

Initially,

i	P	Comments
	0	(P <= 0)

In the first loop iteration,

i	P	Comments
	0	(P <= 0)
3	y_3.X	Iteration1

And in the following ones,

i	P	Comments
	0	(P <= 0)
3	y_3.X	Iteration 1
2	(y_3.X).10 + y_2.X	Iteration 2
1	((y_3.X).10 + y_2.X).10 + y_1.X	Iteration 3
0	(((y_3.X).10 + y_2.X).10 + y_1.X).10 + y_0.X	Iteration 4

©Elena Valderrama, Jean-Pierre Deschamps, Mercè Rullán

At the end of the loop P has the right value.

The rest of the algorithms can be analyzed in a similar way. Algorithm 1 computes P using a similar process, but "i" is comprehended between 0 and 3. As a consequence, the function generated is $(((y_0.X).10 + y_1.X).10 + y_2.X).10 + y_3.X$.

Algorithm 2 "forgets" the term y(3)*X in the function, therefore concluding in the generation of the function $((y_2.X).10 + y_1.X).10 + y_0.X$.

Finally, algorithm 4 **is also correct** and even more efficient then algorithm 3 because the value of P is initialized to $y_3.X$ and that implies that the "for" loop is executed one time less.

[Solution: 3 and 4]

EXERCISE 3

We want to design a circuit that controls the opening of a parking gate when the car driver presses a button (REQ = 1). The system to be designed receives 4 different input signals:

a. REQ: This signal comes from a pushbutton. REQ is set to 1 while the button is pressed and 0 otherwise.

b. UPPER: This signal comes from a sensor placed in the parking gate. UPPER is equal to 1 when the gate is completely open.

c. LOWER: This signal comes from a sensor placed in the parking gate. LOWER is equal to 1 when the gate is completely closed.

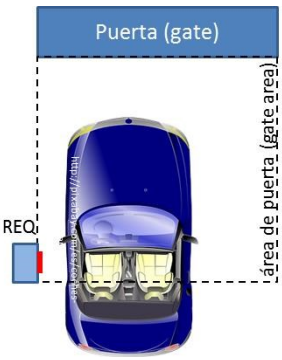
d. SENSORS: This signal comes from a set of sensors placed in the "gate area" (see picture). SENSORS=0 when there are no vehicles inside the gate-area and 1 otherwise.

And generates two output signals to control the gate engine:

a. ON/OFF: ON/OFF=0 when the engine is OFF, and 1 when the engine is ON.

b. UP/DOWN: This signal indicates the direction of movement of the gate.

- To open the door UP/DOWN = 0 (and ON/OFF =1)
- To close the door UP/DOWN=1 (y ON/OFF=1)



Write an algorithm to perform the following sequence of operations:

1. Wait until an open-gate order is received (REQ = 1)
2. Open the gate

©Elena Valderrama, Jean-Pierre Deschamps, Mercè Rullán

3. Wait until the gate is completely open
4. Wait until the gate-area is free of vehicles and start closing the gate
5. When the gate is completely closed go back to the initial state and wait there until a new open-gate order is received.

No new open-gate order should be attended until the current one is completed.

Let's see a possible solution (but not the only one): The algorithm should indefinitely repeat steps 1 to 5, so we would include them inside a loop:

```
loop
        (actions 1 to 5)
end loop;
```

Action 1 says "wait until REQ = 1". This may be coded in different ways. For instance we could use the pseudo-instruction "wait until (*condition1*)", or we could say "don't do anything until …" using a *while loop* with a no-operation inside: "while (*condition2*) loop noop;" where noop means no-operation. Note that both conditions are different: condition1 should be "REQ=1" (wait until an open-gate request is detected), and condition2 should be "REQ=0" (do not do anything while there is not an open-gate request).

We will use the first option:

```
loop
        wait until REQ=1;
        (actions 2 to 5)
end loop;
```

Actions 2 and 3, "open the door completely", could be described saying (i) open the gate (ON/OFF = 1 and UP/DOWN = 0), (ii) wait until the gate is completely open (UPPER=1), and (iii) stop the engine (ON/OFF=0) once the gate is completely open:

```
loop
        wait until REQ=1;
        ON/OFF <= 1 ; UP/DOWN <= 0 ;
        wait until UPPER = 1;
        ON/OFF <= 0;
        (actions 4 and 5)
end loop;
```

Action 4, "wait until the gate-area is free and close the gate" can be write as "wait until SENSORS=0; and "ON/OFF<=1, UP/DOWN<=1"

```
loop
        wait until REQ=1;
        ON/OFF <= 1; UP/DOWN <= 0;
        wait until UPPER = 1;
        ON/OFF <= 0;
        wait until SENSORS = 0;
```

©Elena Valderrama, Jean-Pierre Deschamps, Mercè Rullán

```
            ON/OFF <= 1; UP/DOWN <= 1;
            (action 5)
        end loop;
```

Finally, action 5, “wait until the door is completely closed and return to the initial state” could be written as (i) wait until the door is completely closed (wait until LOWER = 1), (ii) stop the gate engine (ON/OFF=0) and (iii) finish the current loop:

```
        Loop
            wait until REQ=1;
            ON/OFF <= 1; UP/DOWN <= 0;
            wait until UPPER = 1;
            ON/OFF <= 0;
            wait until SENSORS = 0;
            ON/OFF <= 1; UP/DOWN <= 1;
            wait until LOWER = 1;
            ON/OFF <= 0;
        end loop;
```

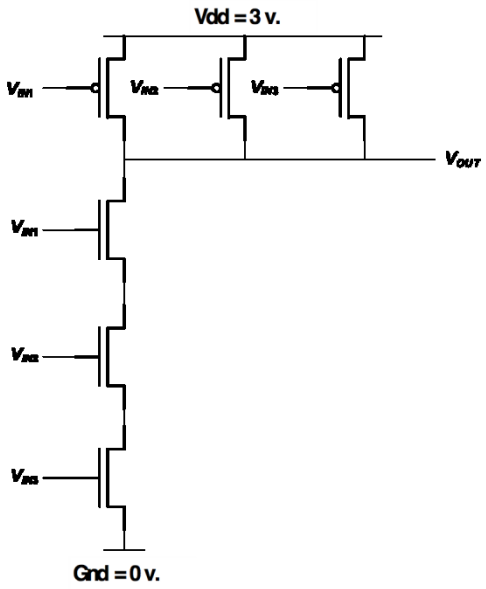
[Solution:]

```
        Loop
            wait until REQ=1;
            ON/OFF <= 1; UP/DOWN <= 0;
            wait until UPPER = 1;
            ON/OFF <= 0;
            wait until SENSORS = 0;
            ON/OFF <= 1; UP/DOWN <= 1;
            wait until LOWER = 1;
            ON/OFF <= 0;
        end loop;
```

©Elena Valderrama, Jean-Pierre Deschamps, Mercè Rullán

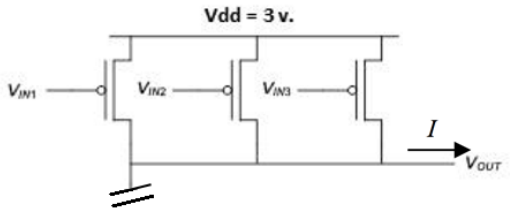
EXERCISE 4

Write the input-output table of the following circuit:



Let's analyze what happens when anyone of the inputs V_{IN1}, V_{IN2} or V_{IN3} has a value of 0 volts:

1) Let's assume that initially V_{OUT} = 0v. As V_{IN1}, V_{IN2} or V_{IN3} is at 0 volts, at least one of the P transistors conducts, and at least one of the N transistors is switched off. Therefore, the path from V_{OUT} to Gnd is cut, and a path from V_{dd} to V_{OUT} is established (see figure). The voltage difference between V_{dd} and V_{OUT} (= 0v) causes a current to pass from V_{dd} to V_{OUT} which increases the voltage of node V_{OUT} up to a value close to V_{dd}. This voltage is identified as a logical 1.



The path to Gnd is cut because at least one N transistor is switched of.

2) If initially $V_{OUT} \approx$ 3v, in spite of some of the P transistor conducts, the voltage of node V_{OUT} would remain at 3v (a logical 1) because $V_{dd} = V_{OUT}$ =3v.

So, up to now, we can describe the operating of the circuit by the next table:

©Elena Valderrama, Jean-Pierre Deschamps, Mercè Rullán

V_{IN1}	V_{IN2}	V_{IN3}	V_{OUT}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1

3) When V_{IN1}=V_{IN2}=V_{IN3} ≈ 3 volts, the situation is just the reverse: The three P transistors are switched off and the three transistors N conduct. If V_{OUT} has the logical value 1 (that is, V_{OUT} ≈ 3v.) a current *I* flows from V_{OUT} to *Gnd*, carrying V_{OUT} to a voltage close to *Gnd* (0v). As in the previous case, if initially V_{OUT} ≈ 0v, V_{OUT} does not change it is because the voltage difference between V_{OUT} and *Gnd* is 0.

V_{IN1}	V_{IN2}	V_{IN3}	V_{OUT}
1	1	1	0

The following table summarizes all cases. As you can see, the circuit works as a 3-input logic NAND gate.

[Solution:]

V_{IN1}	V_{IN2}	V_{IN3}	V_{OUT}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

©Elena Valderrama, Jean-Pierre Deschamps, Mercè Rullán