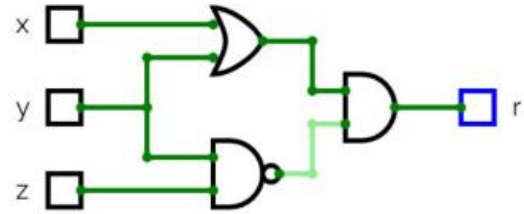


- Sea **D** el circuito digital a continuación:



- 1) Construya la tabla de verdad de **D**
- Muestre los valores de entrada (x, y, z) y el resultado (r) correspondiente en la tabla
 - Cubra todos los casos, siguiendo el orden descendiente que usamos en las láminas del curso

x	y	z	r
1	1	1	0
1	1	0	1
1	0	1	1
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	0

- 2) Implemente una función isomorfa al circuito **D** en **Zilly**
- Sugerencia: primero implemente las funciones correspondientes a las compuertas de **D**
 - Prueben la función en **Zilly**, comprobando que es consistente con la tabla de verdad de **D**

Primero definimos las funciones not, or, and y nand

```
Z => Z not := fn(Z x) -> if(x, 0, 1);
```

```
Z => Z => Z or := fn(Z x) -> fn(Z y) -> if(x, 1, if(y, 1, 0));
```

```
Z => Z => Z and := fn(Z x) -> fn(Z y) -> if(x, if(y, 1, 0), 0);
```

```
Z => Z => Z nand := fn(Z x) -> fn(Z y) -> not(and(x)(y));
```

```
Z => Z => Z => Z D := fn(Z x) -> fn(Z y) -> fn(Z z) -> and(or(x)(y))(nand(y)(z))
```

Comprobación de la tabla de verdad

```
• Q: Z => Z => Z => Z D := fn(Z x) -> fn(Z y) -> fn(Z z) -> and(or(x)(y))(nand(y)(z))
• R: ACK: Z => Z => Z => Z D := λ(Z x) -> λ(Z y) -> λ(Z z) -> and(or(x)(y))(nand(y)(z));
• Q: D(1)(1)(1)
• R: OK: D(1)(1)(1) ==> 0
• Q: D(1)(1)(0)
• R: OK: D(1)(1)(0) ==> 1
• Q: D(1)(0)(1)
• R: OK: D(1)(0)(1) ==> 1
• Q: D(1)(0)(0)
• R: OK: D(1)(0)(0) ==> 1
• Q: D(0)(1)(1)
• R: OK: D(0)(1)(1) ==> 0
• Q: D(0)(1)(0)
• R: OK: D(0)(1)(0) ==> 1
• Q: D(0)(0)(1)
• R: OK: D(0)(0)(1) ==> 0
• Q: D(0)(0)(0)
• R: OK: D(0)(0)(0) ==> 0
```

► 3) Implemente una función isomorfa al circuito **D** en *C/C++*

- Sugerencia: primero implemente las funciones correspondientes a las compuertas de **D**
- Prueben la función en *C/C++*, comprobando que es consistente con la tabla de verdad de **D**

Compuertas lógicas primero

```
#include <iostream>

#include <functional>

auto not_ = [](int x) -> int {
    return x ? 0 : 1;
};

auto and_ = [](int x) {
    return [x](int y) -> int {
        return (x && y) ? 1 : 0;
    };
};

auto or_ = [](int x) {
    return [x](int y) -> int {
        return (x || y) ? 1 : 0;
    };
};
```

```

auto nand = [](int x) {
    return [x](int y) -> int {
        return not_(and_(x)(y));
    };
};

auto D = [](int x) {
    return [x](int y) {
        return [x, y](int z) -> int {
            return and_(or_(x)(y))(nand(y)(z));
        };
    };
};

```

Comprobación de la tabla de verdad

<pre> ~/workspace\$./main 1 1 1 D(1)(1)(1) = 0 Desglose: or_(1)(1) = 1 nand(1)(1) = 0 and_(1)(0) = 0 ~/workspace\$./main 1 1 0 D(1)(1)(0) = 1 Desglose: or_(1)(1) = 1 nand(1)(0) = 1 and_(1)(1) = 1 ~/workspace\$./main 1 0 1 D(1)(0)(1) = 1 Desglose: or_(1)(0) = 1 nand(0)(1) = 1 and_(1)(1) = 1 ~/workspace\$./main 1 0 0 D(1)(0)(0) = 1 Desglose: or_(1)(0) = 1 nand(0)(0) = 1 and_(1)(1) = 1 </pre>	<pre> ~/workspace\$./main 0 1 1 D(0)(1)(1) = 0 Desglose: or_(0)(1) = 1 nand(1)(1) = 0 and_(1)(0) = 0 ~/workspace\$./main 0 1 0 D(0)(1)(0) = 1 Desglose: or_(0)(1) = 1 nand(1)(0) = 1 and_(1)(1) = 1 ~/workspace\$./main 0 0 1 D(0)(0)(1) = 0 Desglose: or_(0)(0) = 0 nand(0)(1) = 1 and_(0)(1) = 0 ~/workspace\$./main 0 0 0 D(0)(0)(0) = 0 Desglose: or_(0)(0) = 0 nand(0)(0) = 1 and_(0)(1) = 0 </pre>
--	--