2. (6 points) **Suma de números Binarios**
Tema a evaluar: Funciones (nivel básico) y recursividad
Crear un programa que realice la conversión de números decimales a **números Binarios**. El programa debe solicitar un número repetidamente y mostrar la conversión del número a binario hasta que el usuario ingrese el valor cero (0). Finalmente debe mostrar los números binarios ingresado y la suma de estos números.
Debe implementar al menos 2 funciones realizando el pase por valor de las variables:

   1. Realizar la conversión de un número decimal a binario de forma recursiva.
   2. Imprimir los números binarios y la suma de estos. Para realizar la suma, se sugiere sumar los números decimales y convertir el número a binario.

   Además, para convertir el número a binario debe implementar una función recursiva, de la siguiente forma:

$$BINARIO(A) = \begin{cases} String(A), & \text{si } A \leq 2 \\ BINARIO(A/2) + String(A\%2), & \text{si } A \geq 2 \end{cases} \quad (1)$$

   IMPORTANTE: En este ejercicio solo se permite el uso de la biblioteca String.

   A continuación se muestra algunos ejemplos de la ejecución correcta del código:

Listing 5: Ejemplo de resultado 1

```
----
NUMEROS BINARIOS
----
INGRESE NUMERO:12
El numero 12 en binario es: 1100
----
NUMEROS BINARIOS
----
INGRESE NUMERO:25
El numero 25 en binario es: 11001
----
NUMEROS BINARIOS
----
INGRESE NUMERO:-6
Solo se realiza el calculo para numeros positivos.
----
NUMEROS BINARIOS
----
INGRESE NUMERO:0
```

2. (6 points) **Sumatoria de números al exponente n de forma recursiva.**
Tenemos las siguientes sumatorias de potencias:

$$\sum_{i=1}^{m} \frac{1}{x_i} = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + ... + \frac{1}{m}$$

$$\sum_{i=1}^{m} \frac{1}{x_i^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + ... + \frac{1}{m^2}$$

$$\sum_{i=1}^{m} \frac{1}{x_i^3} = \frac{1}{1^3} + \frac{1}{2^3} + \frac{1}{3^3} + \frac{1}{4^3} + ... + \frac{1}{m^3}$$

$$...$$

$$\sum_{i=1}^{m} \frac{1}{x_i^n} = \frac{1}{1^n} + \frac{1}{2^n} + \frac{1}{3^n} + \frac{1}{4^n} + ... + \frac{1}{m^n}$$

Se solicita crear un programa que calcule la sumatoria de la potencia n para el número m. Por ejemplo si la potencia n = 3 y el número m = 6 se tendría la siguiente sumatoria.

$$\sum_{i=1}^{m} \frac{1}{x_i^3} = \frac{1}{1^3} + \frac{1}{2^3} + \frac{1}{3^3} + \frac{1}{4^3} + \frac{1}{5^3} + \frac{1}{6^3} = 1.19029$$

Consideraciones:

- Crear una función recursiva que permita calcular la sumatoria de potencias inversas.
- El rango de valores que puede ingresar para la variable **m** es entre 1 y 20. El programa debe validar el valor del dato, en caso sea un número fuera del rango se debe volver a solicitar su ingreso
- El rango de valores que puede ingresar para la variable **n** es entre 1 y 5. El programa debe validar el valor del dato, en caso sea un número fuera del rango se debe volver a solicitar su ingreso

Algunos ejemplos en consola de este programa serían:

Listing 5: Ejemplo 1

```
m = 5
n = 2
Sumatoria de potencia a la 2 hasta el numero (5) = 1.46361
```

Listing 6: Ejemplo 2

```
m = 0
m = 21
m = 6
```



```
n = 3
```

2. (6 points) **Cálculo de exponenciales**
Desarrollar la función **exponencial** que permita calcular el exponencial de un número utilizando la siguiente ecuación:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + ... \tag{1}$$

Continuar la ejecución de la sumatoria mientras: $termino_n > 0$.

$$termino_n = \frac{x^n}{n!} \tag{2}$$

Declaración de función:

```
long double exponential(double x);
```

NOTA: NO SERA CONSIDERADO respuesta válida si se usa de la función `exp` de la libreria `<cmath>`.

Casos de uso #1:

```
cout << fixed << setprecision(5);
cout << exponential(0) << endl;
```

Output:

```
1
```

Casos de uso #2:

```
cout << fixed << setprecision(5);
cout << exponential(1) << endl;
```

Output:

```
2.71828
```

Casos de uso #3:

```
cout << fixed << setprecision(5);
cout << exponential(14) << endl;
```

Output:

```
1202604.28416
```

Casos de uso #2:

```
cout << fixed << setprecision(5);
cout << exponential(23) << endl;
```

Output:

```
9744803446.24890
```

2. (6 points) **Sombrero y suela de zapato:**
Elaborar dos funciones: una función llamada sombrero y otra función llamada suelaDeZapato. Ambas funciones recibirán un valor double n y retornarán un entero, y funcionarán de la siguiente manera:

La función sombrero retornará el entero inmediato mayor o igual n, y la función suelaDeZapato retornará el entero inmediato menor o igual a n.

Desde el el programa principal *main* se deberá leer una variable de tipo double x, y utilizar las funciones sombrero y suelaDeZapato para imprimir al entero inmediato mayor o igual a x y al entero inmediato menor o igual a x.

Asuma que x siempre es positivo. Es decir, no debe validar que x sea positivo, ni tampoco debe considerar el valor sombrero y suela de zapato para números negativos.

Recuerde que puede obtener la parte entera de un valor real escribiendo (int) delante de una variable de tipo double. Por ejemplo, la sentencia a = (int)x; almacena la parte entera de x en la variable a. Por lo tanto, si x tuviera el valor de 2.7, entonces dicha sentencia almacenaría 2 en a.

Ejemplo de ejecución 1:

```
Ingrese x: 2.1
el entero inmediato mayor o igual a x es: 3
el entero inmediato menor o igual a x es: 2
```

Ejemplo de ejecución 2:

```
Ingrese x: 2
el entero inmediato mayor o igual a x es: 2
el entero inmediato menor o igual a x es: 2
```

Ejemplo de ejecución 3:

```
Ingrese x: 5.99
el entero inmediato mayor o igual a x es: 6
el entero inmediato menor o igual a x es: 5
```

Ejemplo de ejecución 4:

```
Ingrese x: 10.001
el entero inmediato mayor o igual a x es: 11
```

3. (7 points) **Calculadora de quebrados**

Programe una calculadora de quebrados que realice las siguientes operaciones binarias:

- Suma de quebrados: $\frac{a}{b}+\frac{c}{d}=\frac{a*d+b*c}{bd}$
- Producto de quebrados: $\frac{a}{b} \cdot \frac{c}{d}=\frac{ac}{bd}$
- Maximo de dos quebrados

Los resultados deben estar en forma de quebrados ($\frac{numerador}{denominador}$) y simplificados, es decir, ser factores mínimos.

Nota: para simplificar los quebrados puede hacer uso del calculo del minimo comun multiplo link: lcm y/o maximo comun divisor link: gcd en C++

En caso los use, no olvide incluir #include <numeric>

Ejemplos: (la solucion no tiene que seguir el formato del ejemplo)

```
Ingrese el primer quebrado
a: 5
b: 8
Ingrese el segundo quebrado
c: 3
d: 12
Ingrese la operacion deseada: (suma, producto, maximo)
suma
Resultado:
7/8
```

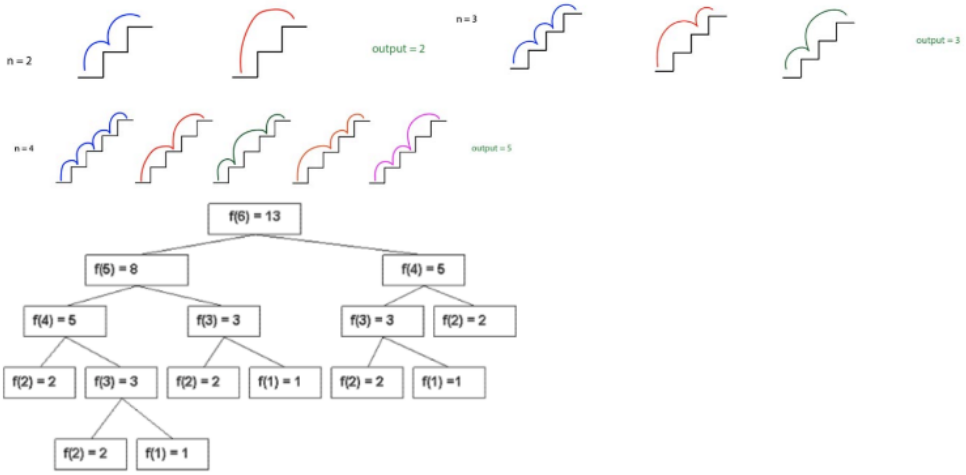
```
Ingrese el primer quebrado
a: 5
b: 8
Ingrese el segundo quebrado
c: 3
d: 12
Ingrese la operacion deseada: (suma, producto, maximo)
maximo
Resultado:
5/8
```

La rúbrica para esta pregunta es:

2. (7 points) **Subir escaleras**
Implemente la solución del problema de encontrar las posibles formas de subir una escalera de n peldaños, si se pueden solo subir 1 o 2 peldaños a la vez.

Aquí la fórmula:

Se puede demostrar que la solución de la cantidad de formas posibles de subir n peldaños es igual a la suma de la cantidad de formas de subir (n-1) peldaños + la cantidad de formas de subir (n-2) peldaños. Las figuras muestran las formas de subir escaleras de n=2,3,4 peldaños, y el diagrama, describe la forma de cálculo para n=6



Programe la solución en forma **recursiva** o **no-recursiva** (a elección del estudiante) para calcular las formas de subir escaleras. Pruebe el algoritmo para n= 10, 25, y 45 peldaños. ¿Encuentra algún problema al calcular un número de peldaños mayor que 50? ¿En ese caso, qué solución propone? (no necesita programarla, solo comentarla)

Ejemplos:

```
cantidad de peldaños: 10
Tiene 89 posibilidades de subir una escalera con 10 peldaños
```


Extras

- Recursive palindrome
- Recursive max
- Recursive Grid

2. (6 points) **Funciones**

Algoritmo raro

Considere un algoritmo que reciba como dato de entrada un número entero positivo n. Si n es par, el algoritmo lo divide entre dos, y si n es impar, el algoritmo lo multiplica por tres y le agrega 1. El algoritmo se repetirá hasta que eventualmente n tome el valor de 1. Por ejemplo, la secuencia para $n = 3$ será la siguiente:

$$3 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$$

Se le pide construir una función *secuencia* que reciba como mínimo un parámetro : el valor de n, e imprima la lista de la secuencia de todos los valores que toma n durante la ejecución del algoritmo mencionado.

```
ingrese n: 1
secuencia:
1
```

```
ingrese n: 3
secuencia:
3 10 5 16 8 4 2 1
```

```
ingrese n: 9
secuencia:
9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```

```
ingrese n: 24
secuencia:
24 12 6 3 10 5 16 8 4 2 1
```

```
ingrese n: 27
secuencia:
27 82 41 124 62 31 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137
412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445
1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438
719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154
3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184
92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
```

```
ingrese n: 2022
```