1) Implemente un programa que a partir del uso de la función **`rand()`** obtenga un histograma con N intervalos de clase, en donde cada intervalo contenga la cantidad de veces que la función **`rand()`** dio un elemento en él (en el primer intervalo contendrá la cantidad de elementos con valores entre **`0`** y **`RAND_MAX/N - 1`**, el segundo la cantidad de elementos con valores entre **`RAND_MAX/N`** y **`2 * RAND_MAX/N - 1`**, etc.) Impleméntelo utilizando arreglos nativos, **`std::array`** y **`std::vector`**. ¿Nota alguna diferencia en la forma de uso?

2) Implementar una función que reciba un arreglo nativo unidimensional y su dimensión y lo imprima

   Prototipo: **`void ImprimeVector(double Vector[], int NumElementos);`**

3)
   a. Implementar una función como la anterior para ingresar los elementos del vector.
   b. Si ahora en lugar de utilizar arreglos nativos utiliza **`std::array`** o **`std::vector`**, ¿Qué comportamiento observa en la variable originalmente pasada a esta función?

4) Implementar una función que reciba un array nativo de enteros y su dimensión y los ordene en sentido creciente utilizando algún algoritmo como los que vimos a principio del curso.

   Prototipo: **`void OrdenaArrayNativo(int V[], int n);`**

5) Arme un .h con los prototipos de las funciones de los problemas 2, 3.a y 4 y copie las definiciones de esas funciones en un archivo separado llamado misfuncs.cpp. Escriba un programa que utilice todas esas funciones y compílelo con:

   ```
   g++ miprog.cpp misfuncs.cpp -o miprog
   ```

   O integrándolo dentro de un proyecto en su entorno de desarrollo.

6) Correr repetidas veces el programa realizado en el problema 5 para distintos tamaños del vector (con las componentes inicializas a través de la función rand) e ir haciendo manualmente una tabla (tamaño, tiempo) para encontrar alguna correlación entre estas dos variables. Analizar el algoritmo utilizado para corroborarlo. Realice la misma tabla para ordenar un **`std::array`** utilizando la función **`std::sort`**. ¿A que conclusión llega?

7) Implementar una función que reciba un array nativo de enteros y su dimensión e invierta a todos sus elementos (El primer elemento pasa a ser el último, el segundo pasa a penúltimo, etc.)
   Prototipo: **`void InvierteArrayNativo(int V[], int n);`**

8) Implemente las siguientes funciones estadísticas:

```
double promedio(double datos[], int n);
double dstandard(double datos[], int n);
```

que calculan el promedio y la desviación estándar de un conjunto de 'n' valores:

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

9) Escriba un programa que utilizando la siguiente estructura para representar puntos en 2D:

```
struct Punto2D {
    double x;
    double y;
};

struct Triangulo {
    Punto2D vertices[3];
    enum ClaseLado   { EQUILATERO, ISOCELES, ESCALENO };
    enum ClaseAngulo { ACUTANGULO, RECTANGULO, OBTUSANGULO };

    ClaseLado clasificaPorLado();
    ClaseAngulo clasificaPorAngulo();
};
```

a. Implemente los métodos **`clasificaPorLado()`** y **`clasificaPorAngulo()`**.
b. Realice un programa que pida al usuario los puntos correspondientes a los 3 vértices de un triángulo y pruebe los métodos anteriores.

10) El problema de cumpleaños está relacionado con cuál es la probabilidad de que en un grupo de personas haya al menos dos que cumplan años el mismo día. Se sabe que si en una misma sala hay 23 personas reunidas, la probabilidad es del 50,7%. Para 57 o más personas la probabilidad es mayor del 99%. Si bien esto se puede deducir de manera analítica, es posible hacer una comprobación empírica. Para eso, considere 1000 grupos de un tamaño fijo de personas, tomando sus fechas de nacimiento al azar y analice en cuántos casos al menos dos personas cumplen el mismo día. Los tamaños de los grupos deben tomarse entre 10 y 60, aumentando de a 10. Hacer una tabla que indique la probabilidad en función del tamaño de grupo. Válgase de funciones para descomponer el problema en sub-problemas.

11) **Ejercicio progresivo**[1]**.** En el archivo coincidencias-1.cpp encontrará el esqueleto de un programa que debe solicitar que se ingresen 2 caracteres por teclado, para luego compararlos. Complete lo marcado con TODO para que el programa funcione correctamente.

12) **Ejercicio progresivo.** (Continuación del Ejercicio 11 de la Práctica 3, **Parcial 1 2018, Problema 3,** puntos b y c).

a) Clasificación de un polígono como convexo. Se dice que un polígono es convexo si todos los segmentos orientados que lo definen cumplen la condición de que todos los puntos de los restantes segmentos están en la misma dirección (o todos están a la derecha, o todos están a la izquierda). Teniendo en cuenta esto, se solicita implementar la función (respetar el prototipo):

```
// Retorna 1 si el polígono recibido es convexo, 0 en caso contrario
// Recibe un polígono como un vector de vértices
int isConvexPolygon( std::vector<P2D_t> polygon );
```

b) **Punto interior a un polígono.** Un punto interior a un polígono orientado cumple la propiedad que si se toma a éste como centro y se recorren los segmentos orientados en orden entonces el ángulo total barrido es $\pm 2pi$[2]. Por el contrario, si el punto es externo, el ángulo barrido es 0. Implementar la función:

```
// Retorna 1 si el punto es interior, 0 en caso contrario
// Recibe un punto y el polígono como vector de vértices
int isInside( P2D_t point, std::vector<P2D_t> polygon );
```

[1] *Los ejercicios marcados de esta forma irán cobrando mayor complejidad con cada práctica, hasta llegar a tener la correspondiente a un ejercicio de parcial*

[2] El ángulo barrido por un segmento orientado se toma en [-pi, pi]. Utilice la relación entre el coseno del ángulo y el producto escalar de dos vectores