1. Investigue cual es el tamaño del stack en su sistema (sistema operativo + entorno de desarrollo). Utilice un arreglo nativo de tamaño constante dentro del main y modifique esa constante hasta que el programa “reviente”. Vuelva a intentarlo utilizando en lugar del arreglo nativo un `std::array`. ¿Qué pasa si utiliza un `std::vector`, haciendo un resize del mismo tamaño que con los anteriores? ¿Y si en lugar de dentro del main declara el arreglo afuera, como variable global?

2. Implemente una función para calcular el factorial de un número, que **recuerde** el último valor calculado y de esta manera pueda evitar parte del cálculo en subsiguientes llamadas. Por ejemplo, si primero se llama a factorial(7) lo calcula desde el principio y lo guarda, pero si después se llama a factorial de 10, entonces utiliza el resultado guardado del factorial de 7 para seguir a partir de ahí. Si se invoca pidiendo el factorial de un número menor que el guardado, no queda más remedio que resetear y calcularlo desde el principio.

3. Cree una clase A con un constructor por defecto que se anuncia a sí mismo. Ahora cree una nueva clase B, ponga un objeto de A como miembro de B y haga que el constructor de B también se anuncie. Cree un arreglo de objetos B y vea qué ocurre.
4. Agregue en el ejercicio anterior destructores que también se anuncien. ¿Qué puede observar con respecto al orden de construcción y destrucción de objetos?

5. Cree una clase sin constructor y muestre que se pueden crear objetos utilizando el constructor por defecto. Ahora cree un constructor que reciba un argumento y reintente el mismo main que usó para mostrar lo anterior.

6. Comprobar la existencia de constructores de copia implícitos.

7. Comprobar la activación de constructores de copia cuando:

    a. Se crean objetos como copia de otros del mismo tipo.

    b. Se pasan argumentos por valor a una función.

    c. Se retorna un objeto de un UDT.

8. Comprobar que cuando una función recibe objetos por referencia, el “copy constructor” no es activado.

9. Hacer un programa que agregue a un `vector<T> v;` 10 elementos de tipo `T` (donde `T` sea una clase cuyos constructores y destructores se anuncien). Redimensione el vector a través del método `vector<T>::resize` para llevarlo a 3 elementos. Verifique que los destructores de los 7 elementos eliminados son activados. ¿Qué pasa si ahora lo redimensiona nuevamente a 10 elementos?

10. Cree una clase con un constructor de copia que se anuncie a sí mismo (en `cout`). Haga una segunda clase conteniendo un objeto miembro de la clase anterior pero no cree un constructor de copia. Muestre que el constructor sintetizado en la segunda clase llama automáticamente al constructor de copia de la primera clase.

11. **Ejercicio progresivo.** En el archivo coincidencias-2.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.** En el archivo coincidencias-3.cpp encontrará el esqueleto de un programa que debe solicitar que se ingresen 2 string de 8 caracteres por teclado, para luego compararlos y ver si al menos un carácter coincide. Complete lo marcado con TODO para que el programa funcione correctamente.

13. **Ejercicio progresivo.** En el archivo coincidencias-4.cpp encontrará el esqueleto de un programa que lee 55 cartas (cada carta se representa como un string) de un mazo guardado como un archivo de texto (mazo.txt).Luego de leer todas las cartas, se imprimen en pantalla. Complete lo marcado con TODO para que el programa funcione correctamente.