**Instrucciones:**

- ◇ Fecha de publicación: 22 de marzo de 2018 a las 12:00.
- ◇ Fecha de entrega: 29 de marzo de 2018 hasta las 12:00.
- ◇ Medio de entrega: `https://e-aulas.urosario.edu.co` (no se reciben entregas por correo electrónico u otros medios).
- ◇ La actividad **debe** realizarse **en parejas**, excepto por **un grupo de tres**.
- ◇ Formato de entrega: implementación, interface y driver en C++14.
- ◇ Nombre archivos: definidos más abajo.
- ◇ Importante: no use acentos ni deje espacios en los nombres de los archivos que cree.

**Protocolo para la evaluación:**

Los siguientes lineamientos serán seguidos de forma estricta y sin excepción.

1. Los grupos pueden consultar entre sí *las ideas básicas*; sin embargo, la solución y detalles del ejercicio debe realizarse **individualmente**. Cualquier tipo de fraude o plagio es causa de anulación directa de la evaluación y correspondiente proceso disciplinario.
2. El grupo de trabajo debe indicar en su entrega de la solución a la actividad cualquier asistencia que haya recibido.
3. El grupo no debe consultar ninguna solución de la solución a la actividad que no sea la suya.
4. El grupo no debe intentar ocultar ningún código que no sea propio en la solución a la actividad.
5. Todas las entregas están sujetas a herramientas automatizadas de detección de plagio en códigos.
6. E-aulas se cerrará a la hora en punto acordada para el final de la evaluación. La solución de la actividad debe ser subida antes de esta hora. El material entregado a través de e-aulas será calificado tal como está. Si ningún tipo de material es entregado por este medio, la nota de la evaluación será 0.0.

No habrán excepciones a estas reglas.

**Enunciado:**

Resuelva el siguiente ejercicio sobre punteros, manejo dinámico de memoria, clases y vectores. Utilice el estándar C++14 en la solución de sus problemas. No olvide compilar con los *flags* apropiados para detectar *warnings* y errores.

Escriba su código a partir de los archivos: implementación (`simple_vector.cpp`), interface (`simple_vector.hpp`) y driver (`main.cpp`). Asegúrese de seguir cuidadosamente las indicaciones del ejercicio.

1. [*Simple vector.*] Implemente una versión muy simple del contenedor `std::vector`. Defina una clase `simple_vector` y suponga que el contenedor implementado almacena elementos de punto flotante tipo `double`. Su código debe manejar correctamente la memoria reservada dinámicamente; es decir, no debe haber *memory leaks*. Esto implica que constructores, destructor y otros métodos, que modifican el estado del contenedor, deben reservar y liberar memoria del *heap*.

   El tamaño de un `simple_vector` puede cambiar dinámicamente. Típicamente, cuando se crea un objeto de esta clase, el arreglo dinámico que contiene los datos es más grande que los elementos que se han agregado, o insertado, hasta el momento. De esta forma el contenedor tiene algunos espacios extra para agregar o insertar más elementos posteriormente. Para diferenciar entre el tamaño del arreglo y el número de elementos, defina dos atributos llamados `capacity` y `length`, respectivamente.

   La interface de esta estructura de datos es la siguiente.

```
class simple_vector {
public:
    simple_vector();
    simple_vector(const simple_vector &rhs);
    ~simple_vector();

    simple_vector & operator=(const simple_vector &rhs);

    void push(double value);
    void erase(int index);
    void insert(int index, double value);
    void modify(int index, double value);
    double retrieve(int index) const;

    void clear();       // deletes contents of array
    bool empty() const; // checks if container is empty
    int size() const;   // returns number of elements

private:
    double *array;
    int capacity, length;
};
```

   Su implementación debe satisfacer los siguiente requisitos:

*a*) El constructor por defecto debe ser el sintetizado por el compilador.

*b*) El constructor con un parámetro de tipo `int` debe ser borrado.

*c*) Los elementos almacenados se deben mantener en un arreglo dinámico.

*d*) La sobrecarga del operador de asignación y definición del constructor copia deben realizar una copiar elemento por elemento del contenido del contenedor.

*e*) Los métodos que modifican el contenedor, siempre deben revisar la capacidad del arreglo tal que siempre hayan lugares disponibles para nuevos elementos.

*f*) Los valores de inicialización de los atributos deben ser un arreglo dinámico de 10 elementos, 10 y 0 para `array`, `capacity` y `length`, respectivamente.

Demuestre el correcto funcionamiento de su implementación invocando los métodos y operadores sobrecargados desde la función principal. La descripción de cada método se muestra a continuación.

*a*) `push(double value)`: agreaga `value` al final de `array`

*b*) `erase(int index)`: elimina elemento en posición `index`

*c*) `insert(int index, double value)`: inserta `value` en posición `index`

*d*) `modify(int index, double value)`: modifica el elemento en `index` por `value`

*e*) `retrieve(int index)`: retorna elemento en posición `index`

*f*) `array`: puntero al arreglo dinámico que contiene los elementos

*g*) `capacity`: tamaño real de `array`

*h*) `length`: número de elementos en el contenedor

Ayuda: En el manejo dinámico de la memoria, puede ser útil implementar un método que se encargue de expandir la capacidad de `array` (al doble de la actual), si está lleno, copiar los elementos al nuevo arreglo y luego liberar la memoria que ya no es necesaria.

Importante: Su implementación debe poder ejecutarse con las instrucciones que contiene el driver. Cuando entregue su Tarea, este archivo no debe estar modificado.

### Invariante de representación

Proponga un invariante de representación apropiado para la estructura de datos particular que está implementando en el ejercicio. Una vez haya escogido el invariante más *relevante y completo* para la desarrollo de la clase, implemente una función

```
1 void checkrep() const;
```

que revisa si el invariante de representación se mantiene después de invocar métodos de la clase que implementa la estructura de datos. En la implementación de esta función puede ser útil la función `assert(...)` definida en la librería `<cassert>`.

Como regla general, es buena idea chequear el invariante de representación a la salida de constructores y en la entrada y salida de métodos modificadores (*mutators*).