Instrucciones:

- ◇ Fecha de publicación: 14 de abril de 2020.
- ◇ Fecha de entrega: 21 de abril de 2020 hasta las 23:55.
- ◇ Medio de entrega: `https://e-aulas.urosario.edu.co` (no se reciben entregas por correo electrónico u otros medios).
- ◇ La actividad **debe** realizarse **en grupos de tres** estudiantes.
- ◇ Formato de entrega: implementación, interfaz y driver (main) en C++14.
- ◇ Nombre archivos: definidos más abajo.
- ◇ Importante: no use acentos ni deje espacios en los nombres de los archivos que cree.
- ◇ Solamente un miembro del grupo debe realizar la entrega en formato `zip`. Liste los miembros del mismo como un comentario en el encabezado de la implementación.

Protocolo para la evaluación:

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

1. Los grupos pueden consultar sus ideas con los profesores para recibir orientación; sin embargo, la solución y detalles del ejercicio debe realizarlos **los integrantes de cada grupo**. 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 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 (a exepción del que se encuentra en las plantillas).
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, iteradores, manejo dinámico de memoria y clases. Utilice el estándar C++14 en la solución de sus problemas. No olvide compilar con los *flags* apropiados para detectar *warnings* y errores: `-Wall -Wextra -Werror`.

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

1. [*Simple vector iterator.*] Se tiene una clase `simple_vector` que implementa una versión muy simple del contenedor `std::vector` para valores tipo `double`. Como métodos básicos la clase incluye: un constructor por defecto, un constructor copia, destructor y la sobrecarga del operador de asignación. Como métodos adicionales tiene `push`, que añade un elemento al final del vector, `erase`, que borra el elemento en la posición indicada, `insert` que inserta un elemento en la posición indicada, `modify`, que cambia el valor del elemento indicado, `retrieve` que retorna el valor del elemento indicado. (La implementación de esta clase se proporciona en los archivos `simple_vector.hpp` y `simple_vector.cpp`).

   [*Iterator.*] Un iterador es un objeto que permite navegar, elemento a elemento, en una estructura de datos. En particular, un iterador bidireccional permite pasar de un elemento al siguiente o al anterior (en donde el orden de los elementos en la estructura está definido). Esto se logra utilizando los operadores de incremento `++` y decremento `--`. Igualmente, el iterador bidireccional permite el acceso a los elementos de la estructura a través del operador de derreferencia (`*`).

   Implemente un iterador bidireccional (`BidirIterator`) que permita recorrer la estructura `simple_vector`. Note que para poder realizar un ciclo del tipo

   ```
   for (it = vec.begin(); it != vec.end(); it++)
       cout << *it << endl;
   ```

   es importante sobrecargar, adicionalmente a los operadores `++`, `--` y `*`, el operador asignación (`=`) y los operadores de comparación (`==` y `!=`). La interfaz de la clase `BidirIterator` se presenta a continuación

   ```
   class BidirIterator {
   private:
       double *ptr;

   public:
       BidirIterator() { ptr = nullptr; }        // def. ctor
       BidirIterator(double *beg);               // par. ctor
       BidirIterator(const BidirIterator & it); // copy ctor
       ~BidirIterator() {}                      // destructor

       double & operator*();         // dereference operator
       BidirIterator & operator=(const BidirIterator & it);

   ```

```
14      // prefix/postfix ++ operators
15      BidirIterator & operator++();  // ++it
16      BidirIterator operator++(int); // it++
17      // prefix/postfix -- operators
18      BidirIterator & operator--();  // --it
19      BidirIterator operator--(int); // it--
20
21      // relational operators
22      bool operator==(const BidirIterator it);
23      bool operator!=(const BidirIterator it);
24  };
```

Note que esta clase es un *wrapper* de un puntero a un `double`, cuya interfaz restringe el tipo de operaciones que puede realizar para cumplir con la especificación de iterador bidireccional.

Finalmente, es necesario declarar el iterador como miembro público de la clase `simple_vector` e implementar los métodos `begin` y `end`, que permiten localizar el inicio y fin del vector:

```
1  // iterator and related methods
2  typedef BidirIterator iterator;
3  iterator begin();
4  iterator end();
```

Demuestre el correcto funcionamiento de su implementación invocando los métodos y operadores sobrecargados desde la función principal.

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