

Instrucciones:

- ◇ Fecha de publicación: 30 de septiembre de 2019.
- ◇ Fecha de entrega: 08 de octubre de 2019 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 parejas**. Si sobra una persona, habrá **un solo grupo de tres**.
- ◇ Formato de entrega: implementación, interface 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.

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 excepción 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.cpp`), interface (`simple_vector.hpp`) y driver (`main.cpp`). Asegúrese de seguir cuidadosamente las indicaciones del ejercicio.

1. [*Simple vector.*] Se tiene una clase `SimpleVect` que implementa una versión muy simple del contenedor `std::vector` para valores `double`. Como métodos básicos la clase incluye el método constructor por defecto, un constructor por copia, destructor y sobrecarga el operador de asignación. Como métodos adicionales del ADT 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, `clear`, que borra todos los elementos, `empty`, que retorna `true` si el vector está vacío o `false` en caso contrario y, finalmente, `size`, que retorna la cantidad de elementos del vector (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 de elemento a elemento en una estructura. 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), utilizando los operadores `++` y `--`. Igualmente el iterador bidireccional permite el acceso a los elementos de la estructura a travez del operador de derreferencia (`*`).

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

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

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

```
1 | class BidirIterator {
2 | private:
3 |     double *ptr;
4 |
5 | public:
6 |     //Default constructor
7 |     BidirIterator() {}
8 |     //Constructor
9 |     BidirIterator(double *beg);
```

```
10 //Copy constructor
11 BidirIterator(const BidirIterator& it);
12 //Destructor
13 ~BidirIterator() {}
14
15 //Derreference operator (An alias to the value stored
16 //    at ptr)
17 double& operator*();
18
19 //prefix/postfix ++ operators
20 BidirIterator& operator++(); //++it
21 BidirIterator operator++(int); //it++
22 //prefix/postfix -- operators
23 BidirIterator& operator--(); //--it
24 BidirIterator operator--(int); //it--
25
26 //Assignment operator
27 void operator=(const BidirIterator it);
28
29 //Comparison operators
30 bool operator==(const BidirIterator it);
31 bool operator!=(const BidirIterator it);
32 };
```

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

Finalmente, es necesario definir el iterador como miembro público de la clase `SimpleVec` 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.