# SMART POINTERS

NICOLÁS ROJAS GUTIÉRREZ
EDWIN ALEJANDRO FORERO GÓMEZ

UNIVERSIDAD DEL ROSARIO
ALGORITMOS Y ESTRUCTURAS DE DATOS
2018

# SMART POINTERS

NICOLÁS ROJAS GUTIÉRREZ
EDWIN ALEJANDRO FORERO GÓMEZ

Trabajo presentado al docente:
JULIÁN RINCÓN

UNIVERSIDAD DEL ROSARIO
ALGORITMOS Y ESTRUCTURAS DE DATOS
2018

# SMART POINTERS

Un puntero es una variable que relaciona una región de memoria, permiten acceder y modificar objetos por medio de una referencia. En la mayoría de las aplicaciones informáticas que son desarrolladas hoy en día, son muy útiles al momento de optimizar algoritmos que requieren frecuentemente acceso a grandes cantidades de datos, pero, el problema de los punteros tradicionales es que no le otrogan al usuario un control más completo del manejo de la memoria a sus programas.

Por esta razón, decidimos implementar los punteros inteligentes de tipo único y compartido que tengan la capacidad de que al ser impelmentados ofrezcan eficiencia y permitan al usuario tener mayor manejo de la memoria.

Nuetros objetivos específicos son:

- Diseñar e implementar la clase s*mart_ptr*.
- Diseñar e imlementar el puntero de tipo único, *unique_ptr*.
- Diseñar e implementar el puntero de tipo compartido, *shared_ptr*.
- Buscar que estos tipos de punteros funcionen para cualquier tipo de dato.
- Realizar pruebas para corregir errores .

**Marco Teórico:**

Un *Smart Pointer*(puntero inteligente), es un tipo abstracto de dato que simula el comportamiento de un puntero corriente pero añade ciertas características adicionales que tiene como objetivo reducir errores a causa del mal uso del mismo y ayuda a mantener la eficiencia, como por ejemplo, cuando lleva registro de los objetos a los que apunta con el fin de gestionar la memoria.

Existen varios tipos de punteros, entre ellos, los punteros *unique_ptr* y *shared_ptr.*

*Unique_ptr,* el sustituto del puntero *auto_ptr*, es un puntero que mantiene la propiedad exclusiva de un objeto a través de un puntero y cuando este se destruye, se elimina el objeto. Este puntero es muy útil si se desea proporcionar seguridad a excepción clases y funciones que manejan los objetos con vida dinámico, garantizando la eliminación tanto de salida normal y salir por excepción .

En contraste, *Shared_ptr*, es un puntero que permite que más de un propietario administre la duración de un objeto en memoria. Tiene la caracteristica de que cuando el último que quede apuntando al objeto se destruya o reestablesca, el objeto se tendrá que destruir.

**Implementación:**

Duante la elaboración del proyecto se intentó diseñar e implementar directamente las clases *unique_ptr* y *shared_ptr*, pero esto no era una buena idea, por lo cual, se inició diseñando una clase s*mart_ptr* que nos facilitaría entender el manejo a estos punteros inteligentes. Esto con el fin de poder usar los métodos de esta clase para los punteros *unique_ptr* y *shared_ptr* por medio de la herencia.

Los métodos de la clase s*mart_ptr* son los siguientes*:*

- Los constructores**:** *smart_pointer(); smart_pointer(Datatype*); smart_pointer(const smart_pointer& s); smart_pointer(smart_pointer&& s)*
- El destructor**:** *~smart_pointer()*
- La sobrecarga operador de asignación**:** *smart_pointer& operator=(const smart_pointer&); smart_pointer& operator=( smart_pointer&&)*
- La sobrecarga del operador de desreferencia/acceso**:** *Datatype& operator*(); Datatype*operator->()*
- La sobrecarga de operadores de comparación**:** *bool smart_ptr<Datatype>::operator==(const smart_ptr<Datatype>& s); bool smart_ptr<Datatype>::operator!=(const smart_ptr<Datatype>& s); bool smart_ptr<Datatype>::operator==(const Datatype*& p); bool smart_ptr<Datatype>::operator!=(const Datatype*& p); bool smart_ptr<Datatype>::operator!()*
- Copia**:** *void copy(const smart_ptr<Datatype>& s);* que hace parte de la sección protegida de la clase.

Teniendo esta clase se implementaron los métodos correspondientes a los punteros *unique_ptr* y *shared_ptr*.

Los métodos para la clase *unique_ptr* son*:*

- Los constructores**:** *unique_ptr(); unique_ptr(Datatype * p); unique_ptr(const unique_ptr<Datatype> & s); unique_ptr(unique_ptr<Datatype>&& s)*
- El destructor**:** *~unique_ptr()*
- La sobrecarga operador de asignación**:** *unique_ptr & operator=(const unique_ptr<Datatype> & s); unique_ptr & operator=(unique_ptr<Datatype>&& s)*

Los métodos para la clase *shared_ptr* son:

- Los constructores**:** *shared_ptr(); shared_ptr(Datatype * p); shared_ptr(const shared_ptr<Datatype> & s); shared_ptr(shared_ptr<Datatype>&& s)*

- El destructor**:** *~shared_ptr()*

- La sobrecarga operador de asignación**:** *shared_ptr & operator=(const shared_ptr<Datatype> & s); shared_ptr & operator=(shared_ptr<Datatype>&& s)*

- Copia**:** *void copy(const shared_ptr<Datatype>& s)*

Cabe resaltar que este último método forma parte de la sección privada de la clase *shared_ptr* y no se impementa en *unique_ptr* porque este puntero no tiene la capacidad de ser copiado, solo de ser movido.

**Conclusión:**

Por último, se alcanzaron los objetivos propuestos y se logró que la herramienta al ser implementada ofrezca eficiencia. Se espera que a futuro se pueda seguir expandiendo la cantidad de punteros inteligentes para proporcionar al usuario variedad y permitan un mayor control de la memoria, con el fin de optimizar los programas que desarrolle.

*i*

---

*i*

*https://github.com/EdwinForero29/Algorithms-Project*