Recursión y punteros

1. ¿Qué es la recursión?

Un tipo de dato recursivo se define como una estructura que puede contener componentes del mismo tipo. La recursión es un mecanismo de estructuración para definir datos agrupados cuyo tamaño puede crecer arbitrariamente y cuya estructura puede ser arbitrariamente compleja.
Para implementarla, los lenguajes de programación convencionales utilizan el concepto de los punteros, sin embargo, los lenguajes funcionales proveen mecanismos más abstractos que enmascaran a los punteros.

2. ¿Qué es un puntero?

Los punteros son variables que tienen como contenido "direcciones" a variables anónimas a las que la única forma de llegar es a través del puntero. Las variables dinámicas son las que apunta el puntero.
Características:

- **Relaciones múltiples entre los items**: varias estructuras sin necesidad de duplicarlo.
- **Acceso a bajo nivel**: los punteros están cerca de la máquina en su implementación.

Sin embargo, los punteros son estructuras inseguras por varias razones, entre ellas, una de sus características principales: el hecho de poder acceder a bajo nivel. La cierto es que pueden obscurecer o hacer inseguros a los programas que los usan.
Algunas inseguridades:

- Violación de tipos
- Referencias sueltas - referencias dangling
- Liberación de memoria: objetos
- Liberación de memoria: objetos perdidos
- Punteros no inicializados
- Punteros y uniones discriminadas
- Alias

3. Liberación de memoria: objetos perdidos

Las variables puntero se alocan como cualquier otra variable en la pila de registros de activación. Los objetos apuntados que se alocan a través de la primitiva new son alocados en la heap. La memoria disponible(heap) podría rápidamente agotarse a menos que de alguna forma se devuelva el almacenamiento alocado liberado. Hay varias situaciones en las que podría liberarse la memoria sin afectar el programa, por ejemplo, si los objetos en la heap dejan de ser accesibles. Sin embargo la cuestión es esa: ¿Cuáles son esas situaciones en que se debe liberar memoria? Lo cierto es que no hay una respuesta exacta: depende del programador que debe notificar al sistema cuando un objeto ya no se usa. Sin embargo, esto último podría provocar punteros sueltos si hubiera más de un puntero apuntando a tal dirección. Este error es dificil de detectar y costoso, por lo que la mayoría de los lenguajes no lo implementan. Es en este contexto en que aparecen dos formas de liberar la memoria:

- **Explícita**: en C, por ejemplo, existe una función free() que libera la memoria reservada de forma dinámica, sin embargo puede generar referencias sueltas. Para evitarlo se necesitaría una verificación dinámica que garantize el uso correcto.
- **Implícita**: en este caso, el sistema, durante la ejecución, tomará la decisión de descubrir la basura por medio de un algoritmo de recolección de basura denominado garbage collector. Para lenguajes que hacen uso de variables dinámicas esta funcionalidad es mucho muy importante.
  El garbage collector es un algoritmo que se ejecuta durante el procesamiento de las aplicaciones. Hay varias formas de implementarlo. Una de ellas es la de *reference counting* que supone que cada objeto

en la heap tiene un campo descriptor extra que indica la cantidad de variables que lo referencian. Si es 0 es porque es basura.

4. TAD(Tipos Abstractos de datos)

Los nuevos tipos de datos definidos por el usuario se llaman tipos abstractos de datos. Los tipos de datos son abstracciones y el proceso de construir nuevos tipos se llama abstracción de datos. Un TAD está compuesto por una forma en que los datos son representados y operaciones inherentes a ese dato. La clave para el desarrollo de estos tipos de dato reside en la abstracción. Abstraer es representar algo descubriendo sus características esenciales y suprimiendo las que no lo son. Un TAD satisface varias características:

- **Encapsulamiento**: la representación del tipo y las operaciones permitidas para los objetos del tipo se describen en una única unidad sintáctica.
- **Ocultamiento de la información**: la representación de los objetos y la implementación del tipo permanecen ocultos. Cada lenguaje llama de distinto modo a los TAD: en ADA son *paquetes*, en JAVA son *clases*, en Modula son *módulos*. Para definir un TAD se utiliza la especificación formal, la cual proporciona un conjunto de axiomas que describen el comportamiento de todas las operaciones. También debe de incluir una parte de sintaxis y semántica. Hay operaciones definidas por sí mismas que se consideran constructores del TAD. Normalmente solo inicializan.