Tarea 1

Algoritmos y Estructuras de Datos III (TR/TQ) - 2022 - 1er. Periodo

Total Puntos: 65 pts.

Cada archivo fuente deberá tener la identificación del grupo (g##) y los integrantes del grupo con la indicación del número de tarea y ejercicio . Por cada integrante colocar: apellido, nombre, CIC y sección.

1. Diseño de TAD en Lenguaje C. (25 pts)

Diseñar el TAD (Tipo Abstracto de Dato) que representa el tipo de dato Lista del lenguaje *Scheme*. Implementar la estructura de datos y las posibles operaciones (crear, destruir, agregar, borrar, listar, etc) (al menos 6 operaciones). Las operaciones deben estar correctamente definidas y documentadas.

Implementar las operaciones de *crear*, *agregar, listar y destruir*.

Scheme es un lenguaje cuya estructura de datos básica es la lista enlazada. Estas listas son heterogéneas, es decir no se necesita que los elementos sean del mismo tipo.

Considerar los tipos **NRO** (float), **STR** (char *), **LISTA**, **NIL (**vacío**)**. *Cada uno de estos tipos deben ser definidos como TAD.* Una cadena siempre empieza y termina con la doble comilla (“) y un número algún caracter 0-9.

Algunos ejemplos de listas en Scheme (fijarse que pueden existir listas de listas).

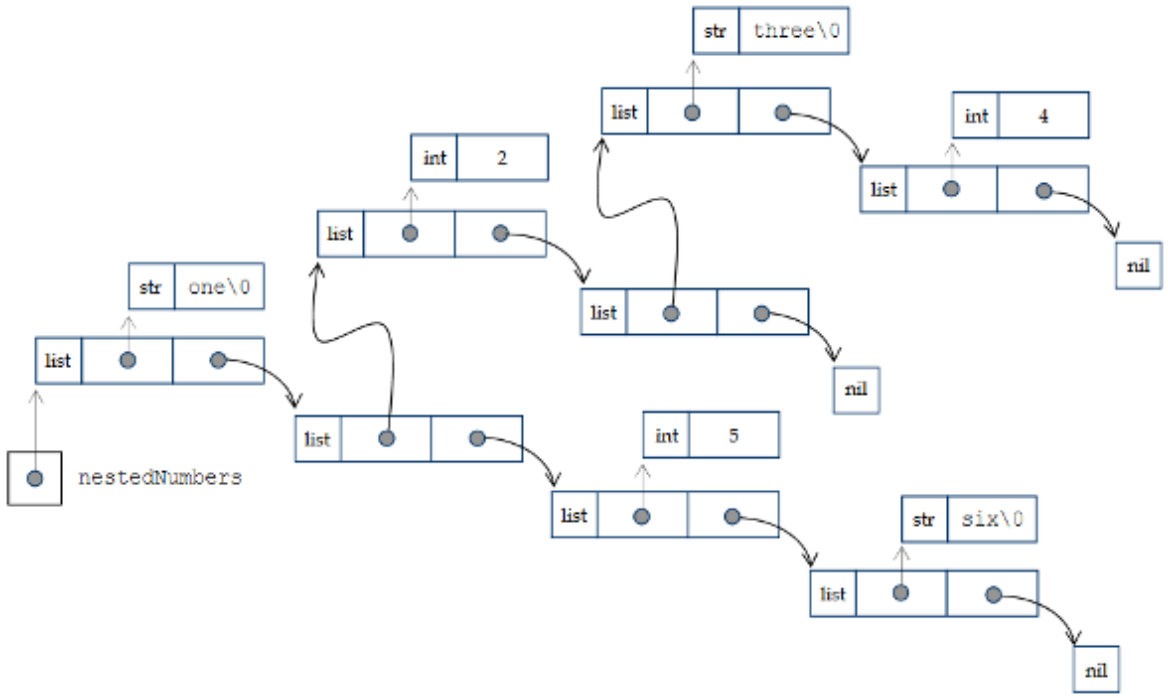
```
( 2 3 4 7 )

( “Hola” “como” “estan” )

( “luque” 2 “cerro” 2 )

((“hola que”) ( 1 3 (4 5) ))
```

Una representación gráfica (en memoria) de la lista `("one" ( 2 ( "three" 4 ) ) 5 "six" )` donde el nombre de la variable es *nestedNumbers* sería



Un posible ejemplo de uso de la implementación para crear la lista : `("hola" 10 ())`

```
LISTA * lista1;
LISTA * lista2;

crearLista(lista1);
crearLista(lista2);

STR * s  = crearSTR("Hola"); // sin restricción de longitud
NUMBER * n  = crearNRO(10);  // solo usar enteros.

agregarLista(lista1, STR);   // el orden de adición define
agregarLista(lista1, NRO);   // lo que se quiere.
agregarLista(lista1,lista2); // lista de lista (ojo con la
                             // circularidad)


listarLista(lista1); // ( "hola" 10 ( ) )


destruirLista(lista1);
destruirLista(lista2);

destruirSTR(s);
destruirNRO(n);
```

Observaciones

- La implementación puede tener varios archivos *.h* y *.c*. Su correcta modularización es parte de la evaluación.
- Debe alzar en el sitio indicado en la Plataforma utilizando la tarea VPL (Virtual Programming).
- Su implementación debe compilarse en VPL (debe poder ejecutarse y probarse de acuerdo a sus guías).
- Debe incluir un archivo en formato texto (extensión .txt) con las indicaciones de uso del TAD.
- Utilizar lenguaje C para la implementación. Solo se pueden utilizar las funciones estándar del lenguaje.
- Se considerará el orden (espaciado, indentación, comentarios, etc), estilo (nombre de variables y funciones, comentario en las funciones, etc) y datos (lo requerido).
- Evitar en lo posible uso de variables GLOBALES, sólo debe utilizarse cuando sea estrictamente necesario y justificar.
- Incluir un archivo fuente de prueba de funcionamiento que contenga la función main (debe ser el único fuente que contenga el punto de entrada), por ejemplo *Test.c*

Rúbrica	**Puntaje**
Diseño de la ED	3
Interface de las operaciones (al menos 6)	3
Implementación de TAD NRO	2
Implementación de TAD STR	2
Implementación de la operación crear	1
Implementación de la operación agregar	1
Implementación de la operación listar	1
Implementación de la operación destruir	1
Documentación de la implementación	2
Modularización	2
Casos de prueba para testing	2
Orden, estilo, comentarios y datos	5

2. Implementaciones en Java (25 pts)

Ejercicio 2.1 - TAD Lista Scheme

Usando la misma descripción del ejercicio 1 de lenguaje C, implemente ahora en Java la misma idea con las mismas consideraciones (obviamente *destruir* ya no hará falta).

Se debe incluir el uso de interfaces (al menos una) y excepciones. También debe hacer que su implementación sea iterable de modo que puede utilizarse el loop mejorado sobre colecciones o arreglos.

No debe utilizar ninguna clase del API de estructura de datos (*Collections*) de Java (ArrayList, Vector, HashTable, etc).

Rúbrica	**Puntaje**
Diseño de la ED	2
Interface de las operaciones (al menos 6)	2
Implementación de TAD NRO	2
Implementación de TAD STR	2
Implementación de la operación crear	1
Implementación de la operación agregar	1
Implementación de la operación listar	1
Documentación de la implementación	2
Modularización	2
Casos de prueba para testing	2
Orden, estilo, comentarios y datos	2
Uso de interfaces	2
Uso de excepciones	2
Es iterable	2

Ejercicio 2.2 - TAD Polinomial (15 puntos)

Los polinomios de la forma $ax^{n} + bx^{n-1} + cx^{n-2} + \ldots + dx^{2} + ex^{1} + fx^{0}$ donde $\{n > 0;\ a, b, c \ldots, d, e, f \geq 0\}$son muy **útiles** en el **álgebra**. Los **científicos** del Laboratorio de **Computación** necesitan una clase Java capaz de representar a este tipo de estructuras algebraicas. Específicamente, necesitan de los siguientes atributos y **Método**s:

a. **Clase**: Se debe definir una clase para representar a un término del polinomio. La clase debe tener como atributos el coeficiente y el exponente del término del polinomial.

b. **Constructor**: Método para creación de una clase a partir de una cadena cuya estructura debe ser la siguiente: `ax^n + bx^n-1 + ... + cx^1 + dx^0`. En caso de que la estructura sea incorrecta, o los valores sean inválidos se debe lanzar una excepción.

c. **Constructor**: Método para creación de una clase a partir de una lista de objetos término. El constructor debe lanzar una excepción en caso que se envíe una lista vacía, o términos con valores no válidos.

d. **Imprimir**: Método que debe imprimir la representación del polinomial en formato de cadena legible para el usuario.

e. **Sumar**: Método que debe ser capaz de realizar la suma entre dos objetos de la clase polinomial. Debe retornar otro objeto de la clase polinomial.

f. **Multiplicar**: Método que debe poder multiplicar un polinomio por un término. Debe retornar otro objeto de la clase polinomial.

g. **Derivar**: Método que debe poder obtener el polinomio que representa a la derivada. Debe retornar otro objeto de la clase polinomial.

Observación: Las clases implementadas deben ser genéricas, de manera a soportar los tipos de datos numéricos más importantes en Java (Integer, Double, Float y otros que puedan usarse).

Rúbrica	**Puntaje**
Implementación correcta de clase y constructores	5
Implementación correcta de **Imprimir, Sumar**.	3
Implementación correcta de **Multiplicar, Derivar**.	3
Comentarios aclaratorios	2
Datos de prueba para los constructores y las operaciones que se realizan sobre polinomiales	2