Sintaxis y Semántica de Lenguajes 2005.2

TP #2 – Sistema Planetario – Presentación

20051023

Por Jorge Muchnik & José María Sola

Objetivos

- Especificar TADs. Precondiciones, poscondiciones.
- Implementar TADs mediante la aplicación de diferentes herramientas provistas por el lenguaje ANSI C y por su Biblioteca estándar.

Introducción

El objetivo de este TP es diseñar las especificaciones e implementaciones de las abstracciones que permitan, en forma simple, modelar diferentes *Sistemas Planetarios*. Esta tarea facilitará el trabajo de investigadores y de programadores del área astronómica.

Las abstracciones son Sistema Planetario, Planeta y Número Astronómico. Por cada una de ellas, se diseñará un tipo de dato abstracto.

Los valores de estos tipos de datos serán manipulados en memoria, pero podrán ser persistidos mediante flujos de texto y flujos binarios, para luego ser recuperados. En otras palabras, podrán ser conservados en memoria externa por un tiempo más prolongado.

Existen varios **roles**, similares pero diferenciables, para los especialistas involucrados en el diseño y utilización de los TADs:

- Especificador del TAD.
- Programador implementador del TAD.
- Programador de las pruebas que validan las implementaciones del TAD.
- Programador de las aplicaciones que utilizan el TAD.
- Usuario de las aplicaciones.

Los programadores que hagan uso de estas abstracciones construirán aplicaciones que permitan tareas como: definir un sistema planetario, definir su estrella o estrellas, agregar planetas, consultar atributos de las entidades, persistir los modelos (a disco, por ejemplo), y recuperarlos entre otras.

Para eso, se proveerá al programador de aplicaciones astronómicas, de los tres tipos de datos abstractos implementados en ANSI C y encapsulados en tres bibliotecas. El programador recibirá la especificación, los tres archivos encabezados y las tres bibliotecas.

Breve Descripción de los TADs

Este documento presenta el TP en forma general, le seguirá un documento con guías y normativas para la especificación e implementación y tres documentos más detallando los requerimientos de cada TDA.

El TAD *SistemaPlanetario* permite operaciones como: creación, asignación de estrella, consulta de cantidad de planetas ó del planeta con mayor masa, agregar planetas, sacar planetas, etc. *Planeta* permite operaciones como: creación, definición del radio, masa, consultar su posición en el sistema, etc. *NumeroAstronomico* permite manejar enteros grandes y asignárselos a diferentes atributos de los TAD *SistemaPlanetario* y *Planeta*. Los tres TADs permiten almacenamiento en memoria externa.

Secuencia de Entregas

1. NumeroAstronomico.
2. Planeta.
3. SistemaPlanetario.—

Sintaxis y Semántica de Lenguajes 2005.2

TP #2 – Sistema Planetario – Normativas y Guías

20051023

Por Ing. José María Sola

Especificación

Guía para la estructuración de la especificación de los valores

Descripción, n-upla, descripción de cada miembro de la n-upla, tipo de dato al que pertenece cada miembro de la n-upla, restricciones.

Guía para la estructuración de la especificación de las operaciones

- $f : A \times B \times C \rightarrow D \times E$ (*Título de la operación*)
- Clasificación de la operación.
- Breve descripción.
- $f : M \times K \times K \rightarrow S \times M$ (*Refinación de los conjuntos*)
- $f(m_1, k_1, k_2) = (s, m_2)$ (*Identificación de los datos y resultados*)
- Precondiciones y Poscondiciones.
- Dominio e Imagen.
- Semántica descripta en lenguaje matemático con la ayuda de axiomas o de lenguaje natural.
- Ejemplo(s).

Normativas para las implementaciones

Archivos Encabezado

El contenido de los archivos encabezado debe estar "rodeado" por las siguientes directivas al preprocesador:

```
#ifndef _INCLUIR_TAD_
#define _INCLUIR_TAD_

/* Contenido del archivo encabezado. */

#endif
```

Esto permite la compilación condicional, evitando incluir más de una vez el contenido del archivo encabezado. Reemplazar *TAD* por un nombre del TAD que se está incluyendo.

Convención de Nomenclatura para los Identificadores

Estilos de "Capitalización"

PascalCase

Todas las palabras juntas, sin espacios. Todas las palabras comienzan con mayúsculas y siguen en minúsculas. Ejemplos:

```
EstaEsUnaFraseEnPascalCase
Pila
AgregarElemento
```

camelCase

Todas las palabras juntas, sin espacios. Todas las palabras comienzan con mayúsculas y siguen en minúsculas, excepto la primera que está en minúsculas. Ejemplos:

```
estaEsUnaFraseEnCamelCase
unaPila
laLongitud
```

UPPERCASE

Todas las palabras juntas, sin espacios. Todas las palabras en mayúsculas. En general se separan las palabras con undescores. Ejemplos:

```
ESTAESUNAFRASEENUPPERCASE
LIMITE_SUPERIOR
MAXIMO
```

Underscores (Guiones Bajos)

- No usar con `camelCase` ni con `PascalCase`,
- Sí usar con `UPPER_CASE`.
- No usar delante de identificadores.

Identificadores para diferentes elementos

Nombres de tipo (typedef)

En camelCase, sustantivo. `SistemaPlanetario`, `Planeta` y `NumeroAstronomico`.

Funciones

En PascalCase, deben comenzar con un verbo en infinitivo. Ejemplos: `OrdenarArreglo()`, `SepararUsuarioDeDominio()`, `Planeta_SetNombre()`, `NumeroAstronomico_EsOverflow()`.

Funciones públicas de cada TAD

Los identificadores de las funciones públicas que implementan las operaciones se prefijan con el nombre del TAD seguido de un *underscore* ("_"). `SistemaPlanetario_`, `Planeta_` y `NumeroAstronomico_`**.**

Variables Locales y Parámetros

En camelCase, sustantivo, en plural para arreglos.

Variables Globales.

En PascalCase, sustantivo, en plural para arreglos.

Enumeraciones.

Su typedef y sus elementos en PascalCase y en singular.

Constantes Simbólicas (#define).

En `UPPER_CASE_CON_UNDERSCORES`.

Cadenas

Implementadas sin tamaño máximo (malloc y free).

Funciones privadas

Definidas como static. No forman parte de la especificación.

Operaciones de Destrucción

No forman parte de la especificación. Permiten liberar los recursos tomados durante la creación.

Operaciones de Creación

Las operaciones de creación serán implementadas como funciones que retornan un puntero a un objeto del tipo del TAD. Deberán usar malloc para obtener memoria para ese objeto. Si no hay memoria disponible, retornarán NULL. Por ejemplo:

```
NumeroAstronomico *NumeroAstronomico_CrearDesdeCadena(
    const char *unaCedena
);
```

Valores de los TADs como parámetros

Un valor de un TAD será pasado como parámetro mediante un puntero a ese valor. Si el parámetro es del tipo entrada/salida no tendrá el calificador const, si es de entrada sí. Por ejemplo:

```
SistemaPlanetario* SistemaPlanetario_AgregarPlaneta(
    SistemaPlanetario* unSistemaPlanetario, /*inout*/
    const Planeta*     unPlaneta              /*in*/
);

int NumeroAstronomico_EsOverflow(
    const NumeroAstronomico* unNumeroAstronomico /*in*/
);
```

Operaciones de Modificación (Mutación, Setter)

Estas operaciones seguirán el modelo impuesto por funciones como strcat de ANSI C.

La función strcat recibe dos parámetros: Primero, la cadena que será modificada concatenándole al final otra cadena, y segundo, la cadena que será concatenada al final de la primera. La función retorna el puntero a la primera cadena.

```
char *strcat(char *s1, const char *s2);
```

Este modelo permite evitar construcciones del tipo:

```
char s1[4+1]="ab", *s2="cd";
strcat(s1, s2);
puts(s1);
```

al ser reemplazadas por:

```
char s1[4+1]="ab", *s2="cd";
puts( strcat(s1, s2) );
```

Análogamente

```
Planeta* Planeta_SetNombre(Planeta* unPlaneta, const char* unNombre);
```

permite reemplazar:

```
Planeta* unPlaneta=Planeta_Crear(…);
Planeta_SetNombre(unPlaneta, "Krypton");
puts( Planeta_GetNombre(unPlaneta) );
```

por:

```
Planeta* unPlaneta=Planeta_Crear(…);
puts( Planeta_GetNombre( Planeta_SetNombre(unPlaneta, "Krypton") ) );
```

Nombres de archivos

- Cada TAD se implementará en una biblioteca. El código fuente de la implementación estará en un archivo *TAD*.c, la declaración de la parte pública en el archivo encabezado *TAD*.h, y se generará la biblioteca *TAD*.lib.

- El código fuente del programa de aplicación que prueba el TAD será *TADAplicacion*.c, y se generará *TADAplicacion*.exe.

- Sólo se entregarán los códigos fuentes: *TAD*.c, *TAD*.h y *TADAplicacion*.c.

- No se aceptarán *TAD*.lib y *TADAplicacion*.exe.

Nombres de los archivos de los TADs de este TP:

- SistemaPlanetario.c, SistemaPlanetario.h, SistemaPlanetario.lib, SistemaPlanetarioAplicacion.c y SistemaPlanetarioAplicacion.exe

- Planeta.c, Planeta.h, Planeta.lib, PlanetaAplicacion.c y PlanetaAplicacion.exe.

- NumeroAstronomico.c, NumeroAstronomico.h, NumeroAstronomico.lib, NumeroAstronomicoAplicacion.c y NumeroAstronomicoAplicacion.exe.

Guía de secuencia de actividades para la generación de los TADs

A continuación se presenta una guía de una posible secuencia de actividades para la construcción de TADs.

En base a una *correcta especificación*, se diseña un *correcto programa de prueba*, luego se implementa el TAD. Si la implementación realizada pasa el programa de prueba, la *implementación es correcta*.

Se debe considerar que *el proceso es en general iterativo*, en el sentido de *la especificación siempre es la entrada para la implementación* y que debe estar *completamente definida*, pero que *hay veces que la implementación retroalimenta a la especificación para mejorarla* y volver a comenzar el proceso.

1. Comprensión del contexto y del problema que el TAD ayuda a solucionar.
2. Diseño de la Especificación.
3. Diseño de los casos de prueba a nivel especificación.
4. Implementación.
   - 4.1. Diseño de prototipos.
   - 4.2. Codificación de prototipos.
   - 4.3. Diseño programa de prueba (aplicación)
   - 4.4. Diseño y codificación de implementación de valores.
   - 4.5. Codificación de funciones públicas y privadas (static).
   - 4.6. Construcción de biblioteca.
   - 4.7. Ejecución de programa de prueba.
   - 4.8. ¿Hubo algún error? Entonces volver a 4.4

Presentación

Forma

- El trabajo debe presentarse en **hojas A4 abrochadas en la esquina superior izquierda**.

- En el **encabezado de cada hoja** debe figurar el **título del trabajo**, el **título de entrega**, el **código de curso** , **número de equipo** y los **apellidos de los integrantes del equipo**.

- Las hojas deben estar enumeradas en el pie de las mismas con el formato "**Hoja n de m**".

- El **código fuente de cada componente del TP** debe comenzar con un **comentario encabezado**, con todos los datos del equipo de trabajo: **curso**; **legajo**, **apellido y nombre de cada integrante del equipo** y **fecha de última modificación.**

- La **fuente** (estilo de caracteres) a utilizar en la **impresión** de los **códigos fuente** y de las **capturas de las salidas** debe ser una fuente de ***ancho fijo*** (e.g. `Courier New`, `Lucida Console`).

- El TP tiene **tres secciones imporatnes, una por cada TAD**, a su vez, **cada sección** tiene las siguientes **tres grandes sub-secciones**:

  Nombre del TAD.
  1. Especificación.
  Especificación completa, extensa y sin ambigüedades de los valores y de las operaciones del TAD.
  2. Implementación
  Biblioteca que implementa el TAD.
  2.1. **Listado** de código fuente del **archivo encabezado**, **parte pública**, *TAD*.h.
  2.2. **Listado** de código fuente de la **definición de la Biblioteca**, **parte privada**, *TAD*.c. Las funciones privadas deberán ser precedidas por su documentación, un comentario con la documentación de la función indicando, entre otras cosas, propósito de la función, semántica de los parámetros in, out e inout.
  1.2.3. **Salidas**. Captura impresa de la salida del **proceso de traducción** (BCC32 y TLIB).
  3. Aplicación de Prueba
  3.1. **Código Fuente**. Listado del código fuente de la aplicación de prueba, *TAD*Aplicacion.c.
  3.2. **Salidas**
  3.2.1. Captura impresa de la salida del **proceso de traducción** (BCC32).
  3.2.2. Captura impresa de las salidas de la **aplicación de prueba**.
  3.2.3. Impresión de **archivos de prueba de entrada** y de **archivos de salida generados durante la prueba**.

- El TP será acompañado por:

  A. Copia Digitalizada
  CD ó disquette (preferentemente CD) con copia de **solamente los 3 archivos de código fuente de cada TAD** (i.e.: *SistemaPlanetario.c, SistemaPlanetario.h, SistemaPlanetarioAplicacion.c, Planeta.c, Planeta.h, PlanetaAplicacion.c, NumeroAstronomico.c, NumeroAstronomico.h y NumeroAstronomicoAplicacion.c*) **No se debe entregar ningún otro archivo**.
  B. Formulario de Seguimiento de Equipo.

Tiempo

- La **entrega** del último TAD, SistemaPlanetario será la primera clase de la semana del **21/11.**

- **Primer recuperatorio**: Durante las fechas de final de **Diciembre**.

- **Segundo recuperatorio**: Durante las fechas de final de **Febrero-Marzo**.

- Luego de la aprobación del TP se **evaluará individualmente a cada integrante del equipo**, esta evaluación también posee dos recuperatorios.—

Sintaxis y Semántica de Lenguajes 2005.2

TP #2 – Sistema Planetario – TAD NumeroAstronomico

20051023

Por Jorge Muchnik & José María Sola

Introducción

El NumeroAstronomico permite a programadores del área astronómica manejar cifras muy grandes.

Las siguientes dos secciones son guías y restricciones para el diseño del TAD pero no son ni la especificación y ni la implementación.

Valores

Un valor NumeroAstronomico es un par ordenado formado por una secuencia de hasta 100 dígitos y un indicador de error.

En la implementación, un valor NumeroAstronomico es un par ordenado formado por una secuencia de hasta 100 dígitos y un segundo dato que representará o bien la longitud del número, o bien un código de error.

```
typedef struct {
     const char* entero;
     int         longitudError;
} NumeroAstronomico;
```

El campo entero es un puntero al comienzo de un arreglo que fue definido dinámicamente. Este arreglo contiene cada uno de los dígitos del NumeroAstronomico. Evaluar diferentes posibilidades: dígito más significativo al principio o al final? Guardar el valor numérico del dígito o el código asociado al símbolo? (i.e. el valor cero ó el 48, que es el código ASCII del carácter '0'). Dejar espacio para el *carry* y marcar el *overflow*? Cómo se puede almacenar los diferentes errores y longitud del número en un mismo campo?

Operaciones

Operaciones de Creación

Aplicar malloc para reservar espacio para la estructura y para el arreglo entero dentro de la estructura.

1. **CrearDesdeCadena** : Cadena → NumeroAstronomico
   Especificar con y sin precondiciones (pero implementar únicamente sin precondiciones).

2. **CrearDesdeCifraSeguidaDeCeros** : Cifra × CantidadDeCeros → NumeroAstronomico
   Especificar con y sin precondiciones (pero implementar únicamente sin precondiciones). Ejemplo: CrearDesdeCifraSeguidaDeCeros(25, 7) = (270000000, *Ninguno*)

3. **CrearAleatorio** : PróximoNúmeroDeLaSecuenciaAleatoria → NumeroAstronomico
   Esta operación tiene precondiciones? En la especificación o en la implementación? Aplicar rand. En la implementación, no habrá parámetro de entrada, ya que el próximo número de la secuencia está implícito luego de invocar a srand.

Operaciones de Manejo de Errores

4. **EsSecuenciaNula** : NumeroAstronomico → Boolean

5. **EsSecuenciaInvalida** : NumeroAstronomico → Boolean

6. **EsOverflow** : NumeroAstronomico → Boolean

7. **EsPunteroNulo**. Esta operación es propia de la implementación.

8. **GetTipoDeError** : NumeroAstronomico → TipoDeError
   TipoDeError = {Ninguno, CadenaNula, CadenaInvalida, Overflow(, PunteroNulo)}
   TipoDeError se implementa como un enum. PunteroNulo es propia de la implementación.

9. **EsError** : NumeroAstronomico → Boolean
   Esta operación equivale a: EsCadenaNula ∨ EsCadenaInvalida ∨ EsOverflow

Operaciones de Salida

10. **Mostrar** : NumeroAstronomico × GruposEnPrimeraLinea × Flujo → Flujo
    Ejemplo:
    Sea na = (800700600500400300200100, *Ninguno*) ∈ NumeroAstronomico entonces:

    Mostrar(na, 3, $stdout_1$) = $stdout_2$

    escribe en stdout las siguientes líneas:

```
\t\t800.700.600.\n
\t\t    500.400.\n
\t\t    300.200.\n
\t\t    100\n
```

    ¿Por qué $stdout_1$ es diferente a $stdout_2$? Notar que la primer línea tiene 3 grupos y las siguientes uno menos (como máximo). Analizar la necesidad de precondiciones.

Operaciones Aritméticas

11. **Sumar** : NumeroAstronomico × NumeroAstronomico → NumeroAstronomico

12. **SonIguales** : NumeroAstronomico × NumeroAstronomico → Boolean

13. **EsMenor** : NumeroAstronomico × NumeroAstronomico → Boolean

De Persistencia

Permiten guardar y recuperar valores NumeroAstronomico en memoria externa, en formatos de texto y binario.

Binario

Definir una representación medianamente eficiente del NumeroAstronomico en disco y en forma binaria.

14. **Read** : Flujo → NumeroAstronomico × Flujo

15. **Write** : NumeroAstronomico × Flujo → Flujo

Texto

El formato de un NumeroAstronómico en un archivo de texto es el siguiente:

secuenciadedigitos#

El dígito más significativo es el primero.

16. **Scan** : Flujo → NumeroAstronomico × Flujo

17. **Print** : NumeroAstronomico × Flujo → Flujo.—

Sintaxis y Semántica de Lenguajes 2005.2

TP #2 – Sistema Planetario – TAD Planeta

20051103

Por José María Sola
con colaboración de Facundo Merighi

Introducción

El TAD Planeta permite a programadores del área astronómica modelar el concepto de planeta.

Las siguientes dos secciones son ***guías*** y restricciones para el diseño del TAD, pero no son ni la especificación ni la implementación.

Valores

Un planeta tiene un nombre, un tipo, una lista de lunas, un radio, un diámetro, una masa relativa a la tierra, un radio promedio de órbita. El tipo pertenece al conjunto {MinorPlanet, Rocky, IcyGiant, GasGiant}, implementado con un enum.

```
typedef struct {
     const char*        elNombre;
     TipoDePlaneta      elTipo;
     NumeroAstronomico* elDiametroConElRadio;
     char*              lasLunas;
     float              laMasaRelativaAaLaTierra;
     NumeroAstronomico* elRadioPromedioDeLaOrbita;
} Planeta;
```

Operaciones

Prestar atención a la definición de las precondiciones y la poscondiciones de todas las operaciones.

1. **Crear** : Nombre × Tipo × ListaDeLunas → Planeta

Operaciones sobre la lista de lunas

La representación de una *luna* será implementada con un string; y la *lista de lunas* será implementada como un string donde cada luna estará separada por punto y coma (**;**). Las operaciones que permitan el manejo de los elementos de esta lista se implementarán haciendo uso de las funciones strtok, malloc (para crear un buffer), free, strcpy, strcat y strcmp; se busca lograr un entendimiento más profundo y una aplicación práctica de estas funciones.

Se construirá una función privada (static), que será invocada por las funciones públicas que manipulan la lista de lunas, la cual abstraerá la construcción de una nueva lista de lunas a partir de una lista existente. La nueva lista actuará como buffer y la función ocultará la información en relación a los detalles de cálculo de espacio para separadores y carácter nulo que forman la nueva lista:

```
static char* CrearListaDeLunas(
     const char* unaListaDeLunas,
     int         unaDiferenciaDeEspacio
);
```

Ejemplos:

```
/* Crear una un buffer con espacio igual a la lista */
char* unBuffer = CrearListaDeLunas(unPlaneta->lasLunas, 0);

/* Crear una un buffer con espacio para una luna más */
char* unBuffer = CrearListaDeLunas (
     unPlaneta->lasLunas,
     strlen("Primer satélite natural")
);
```

```
/* Crear una un buffer con espacio para una luna menos */
char* unBuffer = CrearListaDeLunas(
      unPlaneta->lasLunas,
      -strlen("Tercer satélite natural")
);
```

1. **GetLuna** : Planeta × Posición → Luna.
   Obtiene una luna por su posición. La implementación duplicará la lista invocando a la función privada CrearListaDeLunas; lo recorrerá esta lista con strtok. Al encontrar la posición buscada, reservará memoria para esa luna, la copiará y retornará su puntero. Es responsabilidad de la función llamante liberar el espacio de la luna retornada.

2. **InsertLuna** : Planeta × DespuesDePosición × Luna → Planeta
   Inserta la luna dada luego de la posición indicada, retorna un Planeta igual al Planeta dado pero con la lista de lunas modificada.

3 **RemoveLuna** : Planeta × Luna → Planeta.
   Remueve la luna dada, retorna un Planeta igual al Planeta dado pero con la lista de lunas modificada.

4. **GetCantidadDeLunas** : Planeta → CantidadDeLunas.

De Comparación

5. **EsDeMenorOrbita**: Planeta × Planeta → Boolean

Operaciones de Consulta (Getters, Proyección) y de Modificación (Setters, Mutación)

6. **GetNombre** : Planeta → Nombre
7. **SetNombre** : Planeta × Nombre → Planeta
8, 9. Get / Set **Tipo**
10, 11. Get / Set **Radio**
12, 13. Get / Set **Diámetro**. Operación Set: especificar, y a continuación comentar como se implementaría, pero no implementarla.
14, 15. Get / Set **MasaRelativaAaLaTierra**
16, 17. Get / Set **RadioPromedioDeOrbita**

De Salida

18. Mostrar : Planeta × Flujo → Flujo
   Nombre: *nombre***\tTipo:** *tipo***\n**
   Lunas: *luna***\t***luna***\t***luna***\n**
   \tRadio: *radio***\n**
   \tDiámetro: *diámetro***\n**
   \tMasa relativa a la tierra: *masaRelativaTierra***\n**
   \tRadio promedio de órbita: *radioPromedioOrbita***\n**
   tipo ::= **Planeta Menor** | **Rocoso** | **Gigante Helado** | **Gigante Gaseoso**
   Implementar la siguiente función privada para poder imprimir el tipo del planeta:
   `static char* GetTipoAsString(TipoDePlaneta unTipoDePlaneta);`

De Persistencia

19, 20. Read / Write
21, 22. Scan / Print. En un archivo de texto, los planetas se persisten con el siguiente formato
   {*nombre***,** *tipo***,** *diametroConElRadio***,** *cantidadDeLunas***,** *lunas***,**
   *masaRelativaAaLaTierra***,** *radioPromedioDeLaOrbita***}\n**
   tipo ::= **M** | **R** | **I** | **G**
   diametroConElRadio ::= *NumeroAstronomico*
   masaRelativAaTierra ::= *ver %6f de fprintf*
   lunas ::= **ε** | *nombreDeLuna* | *nombreDeLuna* **;** *lunas*
   cantidadDeLunas ::= *ver %d de fprintf*
   radioPromedioDeLaOrbita ::= *NumeroAstronomico*

–

Sintaxis y Semántica de Lenguajes 2005.2

TP #2 – Sistema Planetario – TAD SistemaPlanetario

20051112

Por José María Sola
con colaboración de Tomás Colombo

Introducción

El TAD SistemaPlanetario permite a programadores del área astronómica modelar el concepto de SistemaPlanetario, el cual incluye estrellas, órbitas y planetas entre otros conceptos. Las siguientes dos secciones son ***guías*** y restricciones para el diseño del TAD, pero no son ni la especificación ni la implementación.

Valores

Cinco-uplas del tipo (*nombre*, *nombre de la estrella*, *masa de la estrella*, *nombres de asteroides*, *planetas*).

La masa de la estrella se expresa relativa a la de al Tierra. La componente *nombres de los asteroides* es un conjunto finito de cadenas, que puede estar vacío. La componente *planetas* es un conjunto de valores del tipo Planeta.

¿Qué ocurre si se desea modelar sistemas planetarios binarios (de dos estrellas) o de varias estrellas? El *nombre de la estrella* pasa a ser la concatenación de los nombres de las estrellas del sistema, y la *masa de la estrella* sería la suma de las masas de cada estrella.

```
typedef struct {
     const char*  elNombre;
     const char*  elNombreDeLaEstrella;
     const float  laMasaDeLaEstrella;
     int          laCantidadDeAsteroides;
     char**       losNombresDeLosAsteroides;
     int          laCantidadDePlanetas;
     Planeta**    losPlanetas;
} SistemaPlanetario;
```

La implementación de los conjuntos de asteroides y de planetas sigue el modelo de ANSI C para los parámetros de la función `int main(int argc, char*argv[])`, explicados en [K&R1988] "*5.10 Command Line Arguments*". El conjunto de asteroides se implementa con un arreglo de punteros a char (i.e. un arreglo de cadenas). El miembro `laCantidadDeAsteroides` indica la longitud del arreglo, y `losAsteroides` es un puntero al primer elemento del arreglo. Los elementos del arreglo son del tipo de dato *puntero a char*, por lo que `losAsteroides` debe ser del tipo de dato *puntero a puntero a char*. Análogamente se implementa el conjunto de planetas. Al implementar las operaciones que manipulan estos conjuntos, deberán utilizar `calloc` ó `malloc` y `realloc`.

Sea `SistemaPlanetario* unSistemaPlanetario; int a; int p;` variables correctamente inicializadas, se utilizarán las siguientes expresiones para acceder a los elementos del arreglo desde las implementaciones:

```
unSistemaPlanetario->losNombresDeLosAsteroides[a]
unSistemaPlanetario->losPlanetas[p]
```

Operaciones

1. **Crear** : Nombre × NombreDeLaEstrella × MasaDeLaEstrella → SistemaPlanetario
   Se debe construir la función:
   ```
   void SistemaPlanetario_Destruir(SistemaPlamentario* unSistemaPlanetario);
   ```
   Es una función pública de implementación (que no debe ser especificada) que invoca a:
   ```
   void Planeta_Destruir(Planeta* unPlaneta);
   ```
   que a su vez invoca a:
   ```
   void NumeroAstronomico_Destruir(NumeroAstronomico* unNumeroAstronomico);
   ```

Operaciones de Consulta (Getters, Proyección) y de Modificación (Setters, Mutación)

2. **GetNombre** 3. **GetNombreDeLaEstrella** 4. **GetMasaDeLaEstrella**
5. **GetCantidadDeLunas**. Cantidad de lunas presentes en el sistema planetario.

6. **GetMasaTotal**. Masa de la estrella sumada a la de los planetas. Para la implementación, construir la función privada:
```
static double GetMasaDePlanetas(int unaCantidadDePlanetas, const Planeta * losPlanetas);
```

Operaciones sobre la lista de Asteroides

7. **AddAsteroide** : SistemaPlanetario × NombreDeAsteroide → SistemaPlanetario
8. **RemoveAsteroide** : SistemaPlanetario × NombreDeAsteroide → SistemaPlanetario
9. **GetCantidadDeAsteroides**
10. **GetAsteroide** : SistemaPlanetario × Índice → NombreDeAsteroide

Operaciones sobre la lista de Planetas

11. **AddPlaneta** : SistemaPlanetario × Planeta → SistemaPlanetario
12. **RemovePlaneta**
13. **GetCantidadDePlanetas**
14. **GetPlanetaPorNombre**
15. **GetPlanetaPorOrbita**. Dado un sistema planetario y una órbita, retorna un planeta. La órbita 1 es la menor, la más cercana a la estrella del sistema planetario.
16. **GetOrbitaDePlaneta**. Dado un sistema planetario y un planeta, retorna su órbita.

De Salida

17. Mostrar : SistemaPlanetario × Flujo → Flujo

```
Nombre: nombre\n
\tEstrella: elNombreDeLaEstrella\tMasa: laMasaDeLaestrella\n
\tAsteroides: elNombreDelAsteroide\telNombreDelAsteroide\n
\tPlanetas:\n
\t\tnumeroDeOrbita\tnombreDelPlaneta\n
\t\tnumeroDeOrbita\tnombreDelPlaneta\n
\tLunas: laCantidadDeLunas\n
\tMasa: laMasaTotal\n
```

De Persistencia

18, 19. Read / Write.

20, 21. Scan / Print. En un archivo de texto, los sistemas planetarios se persisten con un formato similar al siguiente.

```
{\n
\t       elNombre, \n
\t       elNombreDeLaEstrella, \n
\t       laMasaDeLaEstrella, \n
\t       laCantidadDeAsteroides, \n
\t       {\n
\t       \t       elNombreDelAsteroide, \n
\t       \t       elNombreDelAsteroide, ...
\t       \t       elNombreDelAsteroide\n
\t       }\n
\t       laCantidadDePlanetas, \n
\t       {\n
\t       \t       planeta, \n
\t       \t       planeta, ...
\t       \t       planeta\n
\t       }\n
}\n
```

Nota: El planeta sigue el formato especificado las operaciones de persistencia de texto del TAD Planeta.—