UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

Algoritmos y programación 3

Trabajo práctico final: TDA generación de archivos JSON

1ero cuatrimestre, 2018

Alumnos:

Nro	Nombre	Legajo	Email
1	Claudio Ricciardi	25541	gabriel10060@gmail.com
2	Luis Verges	21839	chinoverges@gmail.com
3	Santiago Rojo	18424	tiagox@gmail.com

Fechas de entrega programadas:

- 1er Entrega: 05/05/2018
- 2da Entrega: 19/05/2018
- 3er Entrega: 09/06/2018
- 4ta y última Entrega: 30/06/2018

Nota Final:

Introducción

Objetivo del trabajo

Aplicar los conceptos enseñados en la materia a la resolución de un problema así como también la aplicación de buenas prácticas, trabajando en forma grupal y utilizando un lenguaje de tipado estático de bajo nivel (Lenguaje C).

Consigna general

Desarrollar un conjunto de TDAs que permitan operar con un conjunto de JSON (JavaScript Object Notation). Una vez creado este conjunto de datos utilizarlo en la generación de un archivo JSON. La aplicación deberá ser acompañada con la apropiada documentación de diseño. En la siguiente sección se describe la aplicación a desarrollar.

Descripción de la aplicación a desarrollar

Definir un conjunto de TDAs que permitan generar el siguiente archivo JSON:

```
{
    "size": "0 bytes",
    "hash": "37eb1ba1849d4b0fb0b28caf7ef3af52",
    "bytes": 0,
    "thumb_exists": false,
    "rev": "714f029684fe",
    "modified": "Wed, 27 Apr 2011 22:18:51 +0000",
    "path": "/Photos",
    "is_dir": true,
    "icon": "folder",
    "root": "dropbox",
    "contents": [
        {
            "size": "2.3 MB",
            "rev": "38af1b183490",
            "thumb_exists": true,
            "bytes": 2453963,
            "modified": "Mon, 07 Apr 2014 23:13:16 +0000",
            "client_mtime": "Thu, 29 Aug 2013 01:12:02 +0000",
            "path": "/Photos/flower.jpg",
            "photo_info": {
                "lat_long": [
                    37.77256666666666,
                    -122.45934166666667
                ],
                "time_taken": "Wed, 28 Aug 2013 18:12:02 +0000"
            },
            "is_dir": false,
            "icon": "page_white_picture",
            "root": "dropbox",
            "mime_type": "image/jpeg",
            "revision": 14511
        }
    ],
    "revision": 29007
}
```

Consignas para el programa principal:

Luego de definir apropiadamente las estructuras, crear el programa principal que generará este archivo. Dicho programa podrá recibir como parámetro:

- **Ningún parámetro**: En cuyo caso el JSON se imprimirá en la consola (standard output).
- **Parámetro: `-f <archivo_destino>`**: En cuyo caso el JSON generado deberá escribirse en un archivo llamado `archivo_destino`.

La API para manejar JSON deberá proveer funcionalidad para realizar las siguientes tareas:

1. Crear un nuevo nodo JSON (en adelante nJson) asignándoles nombre y valor al nJson.
2. Cambiar el contenido* a un nJson.
3. Agregar un nuevo nJson a un nJson existente.

* Respecto a los contenidos posibles:

1. nJson.
2. Cualquier tipo de dato de punto flotante.
3. Cualquier tipo de dato entero.
4. Strings.
5. Booleans.
6. Arrays
    - De cualquier tipo entre 1-5 (solo un tipo por vez).

NOTA:

Los contenidos del nJson en ocasiones pueden ser números enteros, flotantes, booleanos, strings, etc. En estos casos el usuario de la API deberá poder operar con los tipos de datos nativos provistos por C que mejor se adapten al dato en cuestión.

Pautas de calidad al momento de resolver el TP

- Los atributos del TDA deberán accederse mediante funciones provistas por la API exclusivamente (Application Programming Interface).
- Dentro de las funciones que operan sobre el TDAs evitar los llamados a funciones que impriman en las salidas estándar de errores o en la salida estándar (`printf`, `fprintf`, etc).
- Cada uno de los TDAs definidos deberán estar apropiadamente documentadas, tanto sus campos como todas sus primitivas.
- Elegir un estándar de codificación y seguirlo a lo largo del proyecto.
- Toda la memoria reservada dentro del TDA será responsabilidad de la API, es decir, si el TDA reserva memoria deberá proveer las herramientas necesarias para liberarla apropiadamente.
- Si necesita utilizar números mágicos, utilice `#define` y documéntelos apropiadamente.
- Los arrays utilizados dentro del TP deberán poder ser de una longitud indefinida (sin restricciones en su cantidad de elementos).
- La estructura del proyecto deberá ser:
    - Archivo `main.c` con el ejemplo de uso de el/los TDA/s
    - Archivo `<declaración_del_tda>.h`
    - Archivo `<definición_del_tda>.c`
- El proyecto deberá ser implementado utilizando la interfase Eclipse CDT.

Formas de entrega

Habrá 4 entregas formales. Las mismas tendrán una calificación de **APROBADO** o **NO APROBADO** en el momento de la entrega. Cada uno de las entregas deberá contar con:

- Estructura de proyecto tal como lo crea el entorno Eclipse CDT.
- Este informe en el formato que lo distribuyó el docente.
- El informe deberá contar con los datos de los integrantes del grupo.
- Cada uno de los requerimientos de la entrega deberán estar agrupados en un contexto (scope) independiente.
- La entrega se realiza digitalmente vía mail. Nombrar el archivo como:

  `TP_<apellido_0>_<apellido_1>_..._<apellido_n>.zip`

1er Entrega:

- Estructura completa del proyecto.
- Creación del `main.c` y la lectura de parámetros que provean la posibilidad de ejecutar la aplicación en los dos modos.

Mínimamente (se aconseja avanzar todo lo posible).

2da Entrega:

Tener la posibilidad de:

- Entrega 1.
- Crear un nJson.
- Poder asignarle cualquier contenido (salvo array) para todos los tipos primitivos de C.
- Agregar la posibilidad de poder imprimir el nJson.

Mínimamente (se aconseja avanzar todo lo posible).

3ra Entrega:

Tener la posibilidad de:

- Entrega 2.
- Cambiar el contenido* a un nJson.
- Agregar un nuevo nJson a un nJson existente.
- Poder asignar como contenido de un nJson un array.
- Completar la salida por pantalla.
- Crear otro JSON y mostrarlo en la salida utilizando el parámetro -c.

Mínimamente (se aconseja avanzar todo lo posible).

4ta y última Entrega:

- Trabajo práctico completo, funcionando, con el informe y cumpliendo todas las normas de calidad.

IMPORTANTE:
Si el TP no se puede ejecutar por cualquier tipo de error (acceso inválido de memoria, goteo de memoria, etc. el TP estará desaprobado.

Fechas de entrega programadas

- 1er Entrega: 05/05/2018
- 2da Entrega: 19/05/2018
- 3er Entrega: 09/06/2018
- 4ta y última Entrega: 30/06/2018

Informe

Supuestos

[Documentar todos los supuestos hechos sobre el enunciado. Asegurarse de validar con los docentes]

- Una estructura de datos en formato JSON es una estructura recursiva: nosotros nos basamos en el estándar definido en https://json.org/.
- Para poder representar esta estructura en forma de árbol recursivo, se realizó un diseño basado en el patrón de diseño *Composite*.

Modelo de dominio

[Explicar los elementos más relevantes del diseño. Es decir: que TDAs se han creado, qué responsabilidades tienen asignadas, cómo se relacionan, etc]

La base que da soporte a la creación de estructuras JSON es el TDA denominado `nJson` (nodo JSON). Dicho TDA tiene como única estructura que se utiliza para representar todos los tipos de nodos posibles de un objeto JSON: objetos, arrays, tipos numéricos, cadenas de caracteres y datos booleanos.

Cada nodo cuenta con la posibilidad de contener la siguiente información:

- **Nombre del atributo**: este dato representa el nombre que llevará la clave de cada atributo contenido en un objeto. Es un dato obligatorio si el nodo es un atributo de un objeto y está prohibido si el nodo es un elemento de un array.
- **Valor del nodo**: esta propiedad almacena cualquier tipo de datos que sea un dato simple. Ya sea datos numéricos, cadenas de caracteres o datos booleanos. Este valor será anulado en caso de que el nodo sea un array o un objeto que contiene atributos.
- **Indicador de array**: un valor booleano que determina si nodo que contiene elementos es de tipo array o no (en cuyo caso será un nodo de tipo objetos).
- **Elementos**: conjunto de elementos de tipo `nJson` contenidos en el nodo. Pueden representar tanto atributos de un objeto como elementos de un array.
- **Función de impresión**: a todos los nodos se les deberá proveer una función que se encargará de la impresión del tipo de dato almacenado.

La combinación de estos datos va a determinar con qué tipo de nodo estamos operando.

Las primitivas del TDA nJson permiten:

- Inicializar un nodo nJson.
- Liberar los recursos consumidos por el TDA.
- Clonar un nodo nJson.
- Asignar un nombre y un valor al nodo nJson.
- Agregar un elemento a un nodo nJson.
- Eliminar un elemento de un nodo nJson.
- Reemplazar un elemento de un nodo nJson por otro.
- Encontrar por nombre de atributo o por índice (en el caso de los arrays) un elemento nJson.

Detalles de implementación

[Deben detallar/explicar qué estrategias utilizaron para resolver los puntos más conflictivos del trabajo práctico.]

Estructura principal

En vistas de que JSON es una estructura recursiva, se optó por utilizar un único tipo de estructura para representar todos los posibles tipos de datos contenidos.

Actualmente, el hecho de que un nodo nJson cuente con ciertos atributos son los que determinan qué tipo de elemento es. Por ejemplo:

Un **objeto** (posiblemente el nodo raíz del JSON) tendría la siguiente forma:

```
- name: NULL
- value: NULL
- is_array: FALSE
- elements: {} // Array literal.
```

Un valor numérico (podría ser un elemento contenido en un array), tendría la siguiente configuración:

```
- name: NULL
- value: 192838.4756822
- is_array: FALSE
- elements: NULL
```

Un atributo de un objeto, contaría con la siguiente combinación de datos:

```
- name: “content”
- value: NULL
- is_array: FALSE
- elements: {}
```

En este caso el atributo, contiene como valor un objeto JSON.

Almacenamiento dinámico de elementos

En una primera versión, la resolución de cómo manejar los elementos agregados a un nodo, se encaró implementando listas enlazadas ya que este tipo de estructuras facilitan el agregado y remoción de elementos. Este acercamiento generó una descentralización de los punteros y mayor dificultad en el acceso aleatorio a los elementos del nodo.

Luego de la recomendación del docente, nos decidimos por implementar estas listas de elementos como arrays dinámicos en memoria; los cuales son aumentados o disminuidos en tamaño según se agregan o se quitan elementos. De esta forma, los punteros que enlazan a los elementos del nodo, están centralizados y es mucho más fácil liberar memoria que haya sido asignada en tiempo de ejecución.

Impresión

Data la naturaleza recursiva de la estructura, fue obvia la estrategía de que el método de escritura del JSON resultante sea de la misma naturaleza. Basado en la combinación de las diferentes propiedades del nodo, se imprimirán diferente orden y caracteres que delimitar objetos, arrays, nombres de atributos, valores y/o separan elementos entre sí.

Liberación de memoria

Para poder realizar una liberación completa de los recursos reservados por el TDA, fue necesario evitar almacenar cualquier tipo de puntero o referencia que llegase como argumento de cualquier primitiva.

Para esto se creó una primita que es capaz de realizar una copia (clon) de todo el contenido de un nJson que es recibido como parámetro y quedar totalmente desacoplado de esos punteros que se crearon fuera de la estructura principal del TDA.

Luego la liberación de memoria se realiza de forma recursiva a través de toda la estructura de árbol del JSON creado.

Checklist de corrección

Esta sección es para uso exclusivo de los docentes, por favor no modificar)

Carpeta

Generalidades

- ¿Son correctos los supuestos y extensiones?
- ¿Es prolija la presentación? (hojas del mismo tamaño, numeradas y con tipografía uniforme)

Modelo

- ¿Está completo? ¿Contempla la totalidad del problema?
- ¿Respeta encapsulamiento?
- ¿Pierde memoria?
- ¿Cumple con las buenas prácticas?

Código

Generalidades

- ¿Respeta estándares de codificación?
- ¿Está correctamente documentado?
- ¿Defne magic numbers? ¿Están documentados?
- ¿Respeta estructura del proyecto?