PRÁCTICA 1: Utilización del concepto de Tipo Abstracto de Datos

OBJETIVOS

El objetivo de esta práctica es utilizar los conceptos aprendidos en las clases de teoría sobre los Tipos Abstractos de Datos (TAD) y reflexionar sobre su utilidad y conveniencia.

Además de la documentación que los entornos de desarrollo proporcionan sobre el lenguaje de programación C, mediante la página de Moodle de la asignatura los profesores de los laboratorios proporcionarán las indicaciones adicionales para desarrollar el trabajo de cada una de las sesiones.

NORMATIVA DE ENTREGA

Como resultado de esta práctica debe entregarse mediante la herramienta Moodle, en la actividad que los profesores de los laboratorios indiquen, un fichero empaquetado (.zip) que contenga todos los ficheros que permitan abrir, construir el ejecutable y ejecutar su proyecto **antes de la primera sesión de la práctica siguiente. Cada grupo puede entregar hasta las 12 de la noche del día anterior al que su profesor iniciará la siguiente práctica.** Este fichero empaquetado debe, por tanto, contener al menos los siguientes ficheros:

- README.txt: fichero de texto plano donde han de constar vuestros datos (nombres, e-mail, grupo de prácticas, número de grupo) y cualquier otra indicación adicional sobre la práctica. Sobre la descripción de vuestro trabajo es suficiente con que expliquéis las cuestiones que os hayan podido parecer relevantes de vuestra solución. No se trata de que repitáis las instrucciones del enunciado ni las indicaciones que vuestros profesores hayan hecho en clase. Sólo debéis centraros en las peculiaridades que vuestra solución contenga. Añade en éste fichero de forma clara la lista de comandos necesarios para generar los ejecutables.
- Los ficheros que deben entregarse en esta práctica se listan a continuación. Se adjunta a este enunciado un fichero .zip que contiene un esqueleto de estos ficheros. En particular se han proporcionado ya implementados las cabeceras y los programas de prueba de los TAD tweet y tweets.
    - tweet.h : Cabecera del TAD tweet
    - tweet.c : Implementación del TAD tweet
    - tweet_test.c: Programa de prueba para el TAD tweet.
    - tweets.h: Cabecera del TAD tweets.
    - tweets.c: Implementation del TAD tweets.
    - tweets_test.c: Programa de prueba para el TAD tweet.
    - twittero.h: Cabecera del TAD twittero
    - twittero.c: Implementación del TAD twittero
    - twittero_test.c: Programa de prueba para el TAD twittero.

- twitteros.h: Cabecera del TAD twitteros
- twitteros.c: Implementation del TAD twitteros.
- twitteros_test.c: Programa de prueba para el TAD twitteros.
- const.h: Fichero de cabecera con constantes globales de la aplicación.
- main.c: Programa principal que incluye toda la funcionalidad de la práctica.

Tenga en cuenta que la práctica está dividida en 3 sesiones.

El nombre del fichero comprimido tiene que seguir la siguiente estructura:

lab<turno>_gr<numero_grupo>_p1_s<numero_sesion>.zip

Donde:

- <turno> es el turno asignado (4111, 4112, 4113 o 4114)
- <numero_grupo> es el que identifica a tu grupo dentro de tu laboratorio
- <numero_sesion> es el de la sesión correspondiente

Es obligatorio respetar todos aquellos nombres y formatos que se describen en el enunciado.

¡No olvidéis incluir vuestros nombres en los ficheros entregados!

CRITERIOS BÁSICOS DE EVALUACIÓN

Las siguientes normas deben ser consideradas en la realización de los problemas propuestos y la entrega del material:

1. *Estilo de programación*: debe ajustarse a las normas básicas de estilo de la programación estructurada en C y que ya se han presentado en asignaturas anteriores de programación.
2. *Documentación del código*: todas las funciones que se programen deberán estar convenientemente y brevemente comentadas, tanto en su cabecera (funcionalidad y parámetros de entrada/salida), como en su cuerpo (descripción de etapas relevantes). Se evitarán comentarios obvios o redundantes.
3. *Pruebas*: es especialmente importante el control de errores. Es justificable que un programa no admita valores muy grandes de los datos, pero no es justificable que el programa tenga un comportamiento anómalo con dichos valores.
4. *Comprobación y propagación de errores*: se deberán comprobar todos los errores susceptibles de producirse en el programa, incluyendo ausencia de archivo de datos de entrada, errores de formato en datos de entrada, fallo de funciones del sistema operativo (malloc, etc.). Se comprobará que los parámetros pasados a las funciones sean correctos (p.ej., no nulos) utilizando preferentemente la función *assert*. Se considerará fallo grave que el programa aborte su ejecución por un tratamiento de errores insuficiente o inadecuado.
5. *Liberación de memoria dinámica*: al finalizar los programas, éstos deberán haber liberado toda la memoria dinámica que hayan reservado durante su ejecución.
6. *Uso de Valgrind*. Es imprescindible que todos los ejercicios hayan sido probados usando Valgrind para asegurarse de que no existen problemas de memoria.

INTRODUCCIÓN

En esta práctica se propone crear un sistema base de datos para almacenar y recuperar información acerca de twitteros y de sus tweets. El sistema tendrá las siguientes características, que se desarrollarán a lo largo de las tres partes que componen las prácticas de la asignatura:

- Lectura de datos de twitteros y tweets almacenados en ficheros de texto.
- Inserción de datos manualmente y su almacenamiento en memoria.
- Guardado de los datos en memoria en disco, siguiendo cierto formato.
- La base de datos mantendrá la consistencia de los datos, comprobando circunstancias como que no haya tweets y twitteros repetidas, etc.
- El sistema ofrecerá una serie de consultas como, “búsqueda de tweets por palabra”, “por *hashtag*”, “búsqueda de los twitteros por nombre”, etc. y medirá el tiempo de ejecución.
- Se utilizará un índice para acelerar algunas consultas.

Para ofrecer una primera impresión de la aplicación que se pretende desarrollar, a continuación se muestra el aspecto que el menú de la aplicación podría tener.

```
1. Twittear
2. Usuarios
       2.1 Iniciar sesión
       2.2 Registrarse
3. Consultas
       3.1 Buscar twitteros por nombre
       3.2 Buscar tweets por palabra
       3.3 Buscar menciones
       3.4 Buscar hashtags
4. Crear índices
5. Salir

> Elige una opción:
```

1ª SEMANA: TAD tweet y lista de tweets

Se implementará el TAD tweet, que contendrá la información de cada tweet antes descrita. La estructura de datos y las operaciones se almacenarán en los ficheros tweet.h y tweet.c. También se implementará un TAD tweets para gestionar listas de tweets.

Las funciones que es necesario implementar para el TAD tweet son las siguientes:

- Crear el tweet, para lo cual todos los campos de la estructura del tweet deben inicializarse a valores por defecto (por ejemplo, a NULL los punteros y a NULL_ID el id):

```
tweet * new_tweet();
```

- Eliminar el tweet y liberar sus recursos (no se debe liberar el puntero a autor):

```
void destroy_tweet(tweet * t);
```

- Funciones de modificación de los datos de los tweets:
  - Se deberá utilizar la función *assert* para comprobar que se cumplen las precondiciones de las funciones, por ejemplo, que el parámetro *tweet *t* no sea nulo.
  - Se deberá copiar la cadena text pasada como argumento. No se deberá asignar el puntero.
  - Se deberá asignar el puntero del autor del tweet a t->author.

```
void set_tweet_id(tweet * t, long long int id);
void set_tweet_text(tweet * t, const char * text);
void set_tweet_author(tweet * t, twittero *a);
```

- Funciones de acceso a los datos de los tweets:

```
long long int get_tweet_id(tweet * t);
const char * get_tweet_text(tweet * t);
twittero * get_tweet_author(tweet * t);
```

- Imprimir la información de un tweet por la salida estándar (`stdout`) en formato libre siempre que se muestren todos los campos:

```
void print_tweet(tweet * t);
```

Asociado a este TAD se definirá un TAD *tweets* para representar un array de tweets. La estructura de datos y las operaciones se almacenarán en los ficheros tweets.h y tweets.c que constará de las siguientes funciones:

- Funciones para la reserva y liberación de memoria. La función new_tweets creará la estructura *tweets* e inicializará sus campos adecuadamente (ej., el número de tweets será 0 y last_id también se asignará a 0). La función destroy_tweets liberará la memoria reservada para la estructura y el array pero no liberará la memoria reservada para los tweets en sí.

```
tweets * new_tweets();
void destroy_tweets(tweets * tts);
```

- Una función para añadir un nuevo tweet indicado por argumento a la colección de tweets. Si el tweet a añadir tiene un id igual a `NULL_ID` se le asignará automáticamente un id dado por el valor del atributo `last_id+1` de la estructura tweets y a continuación se incrementará `last_id` en 1. La función devuelve el id del tweet o bien `NULL_ID` si ha habido un error. No se debe copiar el tweet dentro de la lista, solo almacenar el puntero.

```
int add_tweet(tweets * tts, tweet * t);
```

- Función que devuelve el número total de tweets en el array:

```
int get_n_tweets(tweets * tts);
```

- Función que devuelve el tweet de la posición i-esima:

```
tweet * get_tweet_i(tweets * tts, int index);
```

- Función para buscar un tweet según su identificador:

```
tweet * get_tweet_by_id(tweets * ts, long long int id);
```

- Imprimir la información de la lista de tweets por la salida estándar (`stdout`) en formato libre siempre que se muestren todos los campos:

```
void print_tweets(tweets * ts);
```

Programa de prueba

Para probar el código que has creado se proporcionan dos programas de prueba test_tweet.c y test_tweets.c. Cada uno de estos programas incluye un *main* que ejecuta una serie de pruebas para los TAD tweet y tweets respectivamente. Las pruebas de estas funciones no son exhaustivas por lo que es conveniente que hagáis otras comprobaciones a vuestro código. A continuación se muestra un extracto del código de prueba de test_tweet.c.

```
#include <assert.h>
#include <string.h>
#include <stdio.h>
#include "tweet.h"

int test_set_get_text()
{
  char text[] = {"Supercalifragislisticuespialidoso"};
  int ret;

  tweet * t = new_tweet();

  set_tweet_text(t, text);
  ret = strcmp(get_tweet_text(t), name);

  destroy_tweet(t);

  return ret==0 ? 1 : 0;
}

int main(int argn, char**argv)
{
  assert(test_set_get_text());
  return 0;
}
```

La función *test_set_get_text*() realiza una serie de invocaciones de funciones del TAD tweet comprobando su correcto funcionamiento, por ejemplo, el valor establecido con *set_tweet_text* debe ser igual al obtenido al invocar a *get_tweet_text*. El resultado del test se almacena en la variable ret y es comprobado con la función *assert*() en el *main*. Por último, es importante ejecutar el programa de prueba utilizando la herramienta valgrind, para comprobar que no hay fugas de memoria ni accesos inválidos a memoria.

2ª SEMANA: TAD twittero y lista de twitteros

Se implementará el TAD twittero, que contendrá la información de una twittero. Una twittero tendrá un código representado como un entero de 64bits (long long int), un nombre y un alias. Además el TAD debe mantener una estructura *tweets* para almacenar los tweets emitidos por el twittero. También se implementará un TAD twitteros para gestionar listas de twitteros. El TAD twitteros es prácticamente equivalente al TAD tweets. Abajo podéis encontrar más detalles sobre el TAD twitteros.

Las funciones que es necesario implementar el TAD twittero son las siguientes:

- Crear un twittero, para lo cual todos los campos de la estructura deben inicializarse a valores por defecto (por ejemplo, NULL los punteros y 0 los de tipo numérico):

```
twittero * new_twittero();
```

- Eliminar un twittero y liberar sus recursos:

```
void destroy_twittero(twittero *u);
```

- Al igual que en el TAD *tweets*, se proporcionarán una serie de funciones para modificar los campos del TAD, en este caso:

```
void set_twittero_id(twittero * u, int id);
void set_twittero_name(twittero * u, const char * name);
void set_twittero_alias(twittero * u, const char * alias);
```

- También se implementarán funciones para acceder a los valores de tales campos:

```
int get_twittero_id(twittero * u);
const char * get_twittero_name(twittero * u);
const char * get_twittero_alias(twittero * u);
```

- Se crearán funciones para añadir y buscar tweets. La función de add_twittero_tweet añade un tweet a la lista de tweets del twittero. Además, deberá asignar el campo autor del tweet al twittero. Esto se hará llamando a la función add_tweet_author del TAD tweet.

```
void add_twittero_tweet(twittero * a, tweet *t);
BOOL has_twittero_tweet(twittero * a, tweet *t);
```

- Imprimir la información de un twittero por la salida estándar (`stdout`) en formato libre siempre que se muestren todos los campos:

```
void print_twittero(twittero * a);
```

Asociado a este TAD se definirá un TAD *twitteros* para representar una lista de twitteros. La estructura de datos y las operaciones se almacenarán en los ficheros twitteros.h y twitteros.c. Este TAD tiene funciones equivalentes al TAD *tweets* definido en la semana anterior. A continuación damos el detalle de la función add_twittero ya que tiene alguna diferencia.

- Una función para añadir un nuevo twittero indicado por argumento a la colección de twitteros. Antes de añadir al twittero se debe comprobar que el alias y el id no estén repetidos. Si ya está en la lista de twitteros entonces se debe devolver NULL_ID. Si el twittero a añadir tiene el id a NULL_ID se le asignará automáticamente un id dado por el valor del atributo last_id+1 de la estructura twitteros y a continuación se incrementará `last_id` en 1. La función devuelve el id del twittero o bien NULL_ID si ha habido un error. No se debe copiar el twittero dentro de la lista, solo almacenar el puntero.

```
int add_twittero(twitteros * ts, twittero * t);
```

Programa de prueba

Prueba los TAD *twittero* y *twitteros* propuestos para esta semana siguiendo un esquema similar al propuesto para los TAD *tweet* y *tweets*.

3ª SEMANA: Insertar datos y menú de la aplicación

Se pide implementar un programa principal que, usando los TAD y funciones anteriormente descritos, muestre un menú del sistema para realizar algunas operaciones básicas, que se describen a continuación:

- Tweets.
  - No se podrá twittear si no se ha iniciado sesión con algún usuario (ver más detalles en el siguiente punto). El sistema pedirá el texto del tweet y este se añadirá a una colección de tweets global que estará en memoria hasta la finalización del programa. Además, se añadirá el tweet a la lista de tweets del usuario registrado. No es necesario solicitar el id para el tweet. Éste será asignado automáticamente por el TAD tweets.
- Cuentas de twitteros.
  - Iniciar sesión/cerrar sesión: Si no se ha iniciado sesión se podrá iniciar introduciendo el alias de un twittero ya dado de alta en la lista de twitteros registrados. Con el alias será suficiente para inicial la sesión. No vamos a gestionar contraseñas. Si se hay una sesión iniciada, está opción cerrará la sesión.
  - Registrarse: Da de alta a un nuevo usuario pidiendo nombre y alias y lo almacenará en una colección de twitteros que estará en memoria hasta la finalización del programa. No es necesario pedir el id. Éste lo asignará automáticamente el TAD twitteros.
- Consultas.
  - Todos los tweets: imprimirá por pantalla todos los tweets almacenados en el sistema.
  - Todos los twitteros: imprimirá por pantalla todos los twitteros almacenados en el sistema.
  - Buscar tweet por palabra: Listar todos los tweets almacenados en el sistema que contengan el texto especificado. Se puede utilizar la función *strstr* para encontrar una subcadena dentro de la cadena.

A continuación se muestra un ejemplo de uso de la aplicación:

1. Ejecución del programa

```
$ ./practica1

************************************
*       1. Twittear                *
*       2. Cuentas                 *
*          2.1 Iniciar sesión      *
*          2.2 Registrarse         *
*       3. Consultas               *
*          3.1 Todos los tweets    *
*          3.2 Todos los twitteros *
*          3.3 Buscar tweets       *
*       5. Salir                   *
************************************
Elige una opción >
```

2. Registrarse

```
Elige una opción > 2.2
Introduce los datos:
  Nombre (140 caracteres): Pepe Perez
  Alias (140 caracteres): pepito

Elige una opción > 2.2
Introduce los datos:
  Nombre (140 caracteres): Juan Perez
  Alias (140 caracteres): juanito
```

3. Iniciar sesión y twittear

```
Elige una opción > 2.1
Introduce el alias (140 caracteres): juanito
Sesión iniciada correctamente

Elige una opción > 1
¿Qué sucede? (140 caracteres): Desayunando un chocolate con churros
```

```
Elige una opción > 1
¿Qué sucede? (140 caracteres): En el bus
```

3. Cerrar sesión, iniciar sesión con otro usuario y twittear

```
Elige una opción > 2.1
Adios Juan Perez, sesión cerrada correctamente

Elige una opción > 2.1
Introduce el alias (140 caracteres): pepito
Sesión iniciada correctamente

Elige una opción > 1
¿Qué sucede? (140 caracteres): En el dentista. Argh!!
```

4. Consultas (ejemplo de salida de una consulta)

```
Elige una opción > 3.1
Total tweets: 3
@juanito: Desayunando un chocolate con churros (id=1)
@juanito: En el bus (id=2)
@pepito: En el dentista. Argh!! (id=3)

Elige una opción > 3.2
Total twitteros: 2
Juan Perez (@juanito)
Pepe Perez (@pepito)

Elige una opción > 3.3
Introduce la palabra (140 caracteres): el
Tweet encontrado:
@juanito: En el bus (id=2)
@pepito: En el dentista. Argh!! (id=3)
```