Programación II, 2016-2017

Escuela Politécnica Superior, UAM

Práctica 1: Estructuras de Datos y Tipos Abstractos de Datos

OBJETIVOS

- Profundizar en el concepto de **TAD (Tipo Abstracto de Dato).**
- Aprender a elegir la **estructura de datos** apropiada para implementar un TAD.
- Codificar sus **primitivas** y utilizarlo en un programa principal.

NORMAS

Los ***programas*** que se entreguen deben:

- Estar escritos en ***ANSI C,*** siguiendo las **normas de programación** establecidas.
- Compilar sin errores ni *warnings* incluyendo las banderas "-ansi" y "-pedantic" al compilar.
- **Ejecutarse sin problema en una consola de comandos**.
- Incorporar un adecuado **control de errores.** Es justificable que un programa no admita valores inadecuados, pero no que se comporte de forma anómala con dichos valores.

PLAN DE TRABAJO

Semana 1: P1_E1 completo y funciones más importantes de P1_E2.
Cada profesor indicará en clase si se ha de realizar alguna entrega y cómo: papel, e-mail, Moodle, etc.

Semana 2: todos los ejercicios.
La entrega final se realizará a través de Moodle, **siguiendo escrupulosamente las instrucciones** indicadas en el enunciado de la práctica 0 referentes a la organización y nomenclatura de ficheros y proyectos. Se recuerda que el fichero comprimido que se debe entregar debe llamarse **Px_Prog2_Gy_Pz**, siendo '**x**' el número de la práctica, '**y**' el grupo de prácticas y '**z**' el número de pareja (ejemplo de entrega de la pareja 5 del grupo 2161: P1_Prog2_G2161_P05.zip).

Las **fechas de subida a Moodle del fichero** son las siguientes:

- Los alumnos de ***Evaluación Continua, la semana del 20 de febrero*** *(cada grupo puede realizar la entrega hasta las 23:55 h. de la noche anterior a su clase de prácticas).*
- Los alumnos de ***Evaluación Final,*** según lo especificado en la normativa.

Contenido del .zip para esta Práctica 1:

- Carpeta de la práctica, siguiendo al pie de la letra las instrucciones indicadas en la P0.
- Fichero donde se respondan a las cuestiones planteadas en la PARTE 2 de este documento, con nombre y apellidos de la pareja de prácticas. En esta primera práctica no será necesario elaborar ninguna memoria adicional.

PARTE 1: CREACIÓN DEL TAD NODO Y GRAFO

En esta práctica se va a implementar un grafo de nodos. Para ello, primero se comenzará definiendo el tipo NODO (*Node*), y a continuación se trabajará sobre el TAD GRAFO (*Graph*). Observe que el contenido de algunos de los ficheros necesarios para esta práctica se proporcionan al final de este enunciado (ver apéndice 1).

EJERCICIO 1 (P1_E1)

1. Definición del tipo de dato NODO (*Node*). Implementación: selección de estructura de datos e implementación de primitivas.

En esta práctica, un nodo se representará mediante un ***id*** (un entero) y un nombre (una cadena fija de caracteres).

- Para definir la estructura de datos necesaria para representar el TAD NODO conforme ha sido descrita anteriormente, hay que escribir la siguiente declaración en *node.h*:

```
typedef struct _Node Node;
```

Además, en node.c hay que incluir la implementación del tipo abstracto de datos previamente declarado:

```
struct _Node {
   char name[100];
   int id;
};
```

- Para poder trabajar con datos de tipo Node serán necesarias, al menos, las funciones básicas o primitivas cuyos prototipos se encuentren declarados en el fichero *node.h* (ver apéndice 2). Escribid el código asociado a su definición en el fichero *node.c*.

2. Comprobación de la corrección de la definición del tipo Node y sus primitivas.

Se deberá crear un fichero **p1_e1.c** que defina un programa (de nombre **p1_e1**) con las siguientes operaciones:

- Declarar dos nodos.
- Inicializarlos de modo que el primero sea un nodo con nombre “first” e id 111 y el segundo otro nodo con nombre “second” e id 222.
- Imprimir ambos nodos e imprimir después un salto de línea.
- Comprobar si los dos nodos son iguales
- Imprimir el id del primer nodo junto con una frase explicativa (ver ejemplo más abajo).
- Imprimir el nombre del segundo nodo (ver ejemplo más abajo)
- Copiar el primer nodo en el segundo.
- Imprimir ambos nodos.
- Comprobar si los dos nodos son iguales
- Liberar ambos nodos.

Salida:

```
[111, first][222, second]
Son iguales? No
Id del primer nodo: 111
Nombre del segundo nodo es: second
[111, first][111, first]
Son iguales? Sí
```

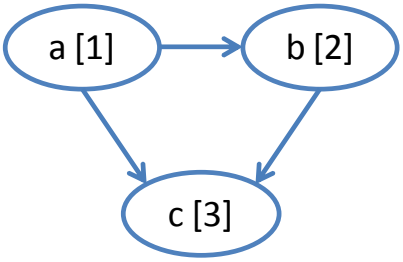
EJERCICIO 2 (P1_E2)

Definición del tipo abstracto de dato GRAFO.

1. Implementación: selección de estructura de datos e implementación de primitivas.

En esta parte de la práctica se definirá el Tipo Abstracto de Datos (TAD) GRAFO como un conjunto de elementos homogéneos (del mismo tipo) y otros atributos que nos permitirán saber cómo están conectados los elementos.

Se tiene que **definir una estructura de datos para representar el TAD GRAFO** (en los ficheros *graph.c* y *graph.h*), suponiendo que los datos que hay que almacenar en él son de tipo **Node** y que su capacidad máxima son 4096 elementos. Tened en cuenta que la información sobre qué nodos están conectados con qué otros nodos se suele almacenar en una *matriz de adyacencia* (una matriz de 0's y 1's indicando si el nodo correspondiente a la fila está conectado con el nodo correspondiente a esa columna). Es decir, el siguiente grafo tendría la matriz de adyacencia que se muestra a continuación:



	Nodo a	Nodo b	Nodo c
Nodo a	0	1	1
Nodo b	0	0	1
Nodo c	0	0	0

Para implementar los TADs relacionados con el grafo:

- Definid en el fichero *graph.h* el nuevo tipo de dato **Graph**,
- Definid en graph.c la estructura de datos **_Graph** que contendrá los nodos del grafo.
- Para poder trabajar con datos de tipo **Graph** serán necesarias, al menos, las funciones básicas o primitivas indicadas en el apéndice 3. Las funciones se declaran, mediante sus prototipos, en *graph.h*. Escribid su código en *graph.c*.

2. Comprobación de la corrección de la definición del tipo Graph y sus primitivas.

Definid un programa en un fichero de nombre **p1_e2.c** cuyo ejecutable se llame **p1_e2**, y que realice las siguientes operaciones:

- Declarar dos nodos e inicializarlos igual que en el ejercicio p1_e1 (el primero con nombre "first" e id 111 y el segundo con nombre "second" e id 222).
- Inicializar e imprimir el grafo.
- Insertar nodo 1 e imprimir el grafo.
- Insertar nodo 2 e imprimir el grafo.
- Comprobar si el nodo 1 está conectado con el nodo 2 (ver mensaje más abajo).
- Comprobar si el nodo 2 está conectado con el nodo 1 (ver mensaje más abajo).
- Insertar conexión entre nodo 2 y nodo 1 e imprimir el grafo.
- Volver a realizar las dos comprobaciones.
- Insertar conexión entre nodo 1 y nodo 2 e imprimir el grafo.
- Volver a realizar las dos comprobaciones.
- Insertar conexión entre nodo 2 y nodo 1 e imprimir el grafo.
- Volver a realizar las dos comprobaciones.
- Destruir nodos y grafo.

Salida:

```
N=0, E=0:
Insertando nodo 1...
N=1, E=0:
[111, first]->0
Insertando nodo 2...
N=2, E=0:
[111, first]->0 0
[222, second]->0 0
Conectados nodo 1 y nodo 2? No
Conectados nodo 2 y nodo 1? No
Insertando nodo 2 -> nodo 1
N=2, E=1:
[111, first]->0 0
[222, second]->1 0
Conectados nodo 1 y nodo 2? No
Conectados nodo 2 y nodo 1? Sí
Insertando nodo 1 -> nodo 2
N=2, E=2:
[111, first]->0 1
[222, second]->1 0
Conectados nodo 1 y nodo 2? Sí
Conectados nodo 2 y nodo 1? Sí
Insertando nodo 2 -> nodo 1
N=2, E=2:
[111, first]->0 1
[222, second]->1 0
Conectados nodo 1 y nodo 2? Sí
Conectados nodo 2 y nodo 1? Sí
```

3. Comprobación del funcionamiento de un grafo a través de ficheros.

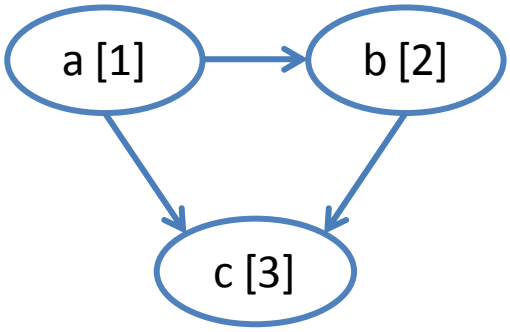
Junto con este enunciado, se os entrega un programa que permite probar los grafos usando ficheros. Este programa (**graph_test.c**) permite cargar un grafo a partir de información leída en un fichero de texto y a continuación mostrarlo por pantalla haciendo uso de métodos del prototipo de Graph (nótese que la salida de este programa y la de graph_print no debe ser la misma).

La ejecución de este programa debería funcionar sin problemas, incluyendo, aunque no sea obligatorio en este momento, la gestión de memoria (es decir, valgrind no debería mostrar fugas de memoria al ejecutarse).

Un ejemplo de fichero de datos de entrada es el siguiente, donde la primera línea indica el número de nodos que se van a introducir en el grafo y en las siguientes la información correspondiente a cada nodo (id y nombre); después de estas líneas aparecerán hasta que se acabe el fichero pares de enteros indicando qué nodos están conectados con cuáles:

```
3
1 a
2 b
3 c
1 2
1 3
2 3
```

De esta forma, este fichero y el siguiente grafo son equivalentes:



PARTE 2: PREGUNTAS SOBRE LA PRÁCTICA

Responded a las siguientes preguntas **completando el fichero disponible en el .zip**. Renombrad el fichero para que se corresponda con su nº de grupo y pareja de prácticas y adjuntadlo al fichero .zip que entreguéis.

1. ¿Sería posible implementar la función de copia de nodos empleando el siguiente prototipo **STATUS node_copy(Node nDest, const Node nOrigin);** ? ¿Por qué?

2. ¿Es imprescindible el puntero Node* en **int node_print(FILE * pf, const Node* n);** o podría ser **int node_print(FILE * pf, const Node p); ?**
   Si la respuesta es sí: ¿Por qué?
   Si la respuesta es no: ¿Por qué se utiliza, entonces?

3. ¿Qué cambios habría que hacer en la función de copiar nodos si quisiéramos que recibiera un nodo como argumento donde hubiera que copiar la información? Es decir, ¿cómo se tendría que implementar si en lugar de **Node* node_copy(const Node* nOrigin)**, se hubiera definido como **STATUS node_copy(const Node* nSource, Node* nDest)**? ¿Lo siguiente sería válido: **STATUS node_copy(const Node* nSource, Node** nDest)**? Discute las diferencias.

4. Indica qué se tendría que cambiar en **graph.c/h** para tener grafos que pudieran almacenar cualquier estructura de datos, es decir, que no estuviera limitado a almacenar nodos definidos como el TAD Node.

Apéndice 1: types.h

```
/*
 * File:   types.h
 * Author: Profesores de PROG2
 */

#ifndef TYPES_H
#define  TYPES_H

typedef enum {
   ERROR = 0, OK = 1
} Status;

typedef enum {
   FALSE = 0, TRUE = 1
} Bool;

#endif    /* TYPES_H */
```

Apéndice 2: node.h

```
#ifndef NODE_H
#define  NODE_H

#include <stdio.h>
#include "types.h"

typedef struct _Node Node;

/* Inicializa un nodo, reservando memoria y devolviendo el nodo inicializado si lo ha hecho correctamente, NULL en otro caso */
Node * node_ini();
/* Libera la memoria dinámica reservada para un nodo */
void node_destroy(Node * n);

/* Devuelve el id de un nodo dado, o -1 si se produce algún error */
int node_getId(const Node * n);
/* Devuelve un puntero al nombre de un nodo dado, o NULL si se produce algún error */
char* node_getName(const Node * n);

/* Modifica el id de un nodo dado, devuelve NULL si se produce algún error */
Node * node_setId(Node * n, const int id);
/* Modifica el nombre de un nodo dado, devuelve NULL si se produce algún error */
Node * node_setName(Node * n, const char* name);

/* Devuelve TRUE si los dos nodos pasados como argumento son iguales (revisando todos sus campos).
   Devuelve FALSE en otro caso. */
Bool node_equals(const Node * n1, const Node * n2);
/* Copia los datos de un nodo a otro devolviendo el nodo copiado (incluyendo la reserva de la memoria necesaria) si todo ha ido bien, o NULL en otro caso */
Node * node_copy(const Node * src);

/* Imprime en pf los datos de un nodo con el siguiente formato:  [id, name]. Por ejemplo, un nodo con nombre “aaa” e id 123 se imprimirá como [123, aaa]. Además devolverá el número de caracteres que se han escrito con éxito (mirar documentación de fprintf)  */
int node_print(FILE *pf, const Node * n);

#endif    /* NODE_H */
```

Apéndice 3: graph.h

```
#ifndef GRAPH_H
#define  GRAPH_H

#include "node.h"

#define MAX_NODES 4096

typedef struct _Graph Graph;

/* Inicializa un grafo, reservando memoria y devolviendo el grafo inicializado si lo ha hecho correctamente, o NULL si no */
Graph * graph_ini();
/* Libera la memoria dinámica reservada para un grafo */
void graph_destroy(Graph * g);

/* Devuelve el número de nodos de un grafo, o -1 si se produce algún error */
int graph_getNnodes(const Graph * g);
/* Devuelve la lista de ids de nodos contenidos en el grafo usando nueva memoria para el puntero que se devuelve (quien llame a esta función se tiene que encargar de liberar dicho puntero al terminar). Si hay algún error se devuelve NULL */
int* graph_getNodeIds(const Graph * g);
/* Devuelve el número de conexiones que se almacenan en un grafo, o -1 si se produce algún error */
int graph_getNedges(const Graph * g);

/* Se añade un nodo al grafo reservando memoria nueva para dicho nodo, actualizando los atributos internos del grafo que sean necesarios. Se devuelve el grafo actualizado si todo ha ido bien, o NULL en otro caso */
Graph * graph_addNode(Graph * g, const Node* n);
/* Se añade una conexión (entre el nodo con id nId1 y el nodo con id nId2) al grafo, actualizando los atributos internos del grafo que sean necesarios. Se devuelve el grafo actualizado si todo ha ido bien, o NULL en otro caso */
Graph * graph_addEdge(Graph * g, const int nId1, const int nId2);

/*
Función privada recomendada: permite encontrar el índice (del array) en el que se almacena un nodo en el grafo con un id dado. Devuelve ese índice si lo encuentra o -1 si no o si ha habido algún otro error
int find_node_index(Graph * g, int nId1);
*/

/* Devuelve un puntero a la estructura Node almacenada en el grafo tal que dicho nodo tiene como id nId. Si no se encuentra o hay algún error, se devuelve NULL */
Node * graph_getNode(const Graph * g, int nId);
```

```
/* Devuelve TRUE o FALSE según si los nodos correspondientes a los ids pasados como argumento  están
conectados en el grafo o no (existe una conexión entre nId1 y nId2, en ese sentido). Si hay algún error se devuelve
FALSE */
Bool graph_areConnected(const Graph * g, const int nId1, const int nId2);

/* Devuelve el número de conexiones que se conocen en el grafo desde un nodo, si hay algún error devuelve -1 */
int graph_getNumberOfConnectionsFrom(const Graph * g, const int fromId);
/* Devuelve una lista con los ids de los nodos conectados desde un nodo dado, si hay algún error devuelve NULL*/
int* graph_getConnectionsFrom(const Graph * g, const int fromId);
/* Devuelve el número de conexiones que se conocen en el grafo hacia un nodo, si hay algún error devuelve -1 */
int graph_getNumberOfConnectionsTo(const Graph * g, const int toId);
/* Devuelve una lista con los ids de los nodos conectados desde un nodo dado, si hay algún error devuelve NULL*/
int* graph_getConnectionsTo(const Graph * g, const int toId);

/* Imprime en pf los datos de un grafo, devuelve el número de caracteres que se han escrito con éxito (mirar
documentación de fprintf). El formato a seguir es: imprimir una primera línea con el número de nodos y conexiones,
después, se imprime una línea por nodo, incluyendo la información del nodo y las conexiones con el resto de
nodos como 1 o 0.
La salida para el grafo pintado en el ejercicio 2.3 de la parte 1 es:
N=3, E=3:
[1, a]->0 1 1
[2, b]->0 0 1
[3, c]->0 0 0
*/
int graph_print(FILE *pf, const Graph * g);

#endif    /* GRAPH_H */
```