EXAMEN DE SISTEMAS INFORMÁTICOS INDUSTRIALES
(SOLUCIÓN) (TEORÍA)

La duración del examen es de 2 horas. **DICIEMBRE 2016**

1. Indicar si las siguientes afirmaciones son verdaderas o falsas:

(a) Al declarar que una función recibe como parámetro una matriz, se puede dejar vacío el tamaño correspondiente a la primera dimensión de la matriz. Ejemplo: **void LeerMatriz(float m[][3])**. **V**

(b) En C++ una clase derivada no puede heredar de más de una clase base. **F**

(c) Una función en C no puede llamarse a sí misma. **F**

(d) En C++ se puede utilizar para mostrar el valor de una variable entera **i** tanto **"cout<<i;"** como **"printf("%d",i);" V**

(e) En C++ se pueden implementar diferentes destructores para la misma clase. **F**

(f) Para comunicar dos equipos mediante sockets, sólo se puede utilizar el protocolo TCP. **F**

(g) Al definir el operador **=** en una clase, siempre se debe devolver ***this**. **V**

(h) En C, los arrays se pasan por dirección, es decir, no se hace una copia para la función de los valores de los elementos del array. **V**

(2 puntos)

2. Diseñar un programa que lea información por línea de comandos (argc y argv) y la almacene en una matriz de la forma que se indica a continuación. La información introducida constará (repetidas veces) de un número y luego una palabra. El número indicará cuantas veces deberá copiarse la palabra en la matriz. Primero habrá que determinar cuántas filas serán necesarias para almacenar la información, reservar dinámicamente la memoria, rellenar la matriz y finalmente mostrarla. Se pide:

(a) Determinar la dimensión a reservar en función de la información introducida por línea de comandos. *(0.5 puntos)*

(b) Reservar dinámicamente memoria para la matriz suponiendo que la longitud máxima de una palabra es de 50. *(0.75 puntos)*

(c) Rellenar la matriz copiando cada palabra tantas veces como indique el número que tiene delante. *(1.0 puntos)*

(d) Finalmente mostrar la información almacenada en la matriz. *(0.5 puntos)*

NOTA: se puede usar la función *atoi* para convertir una cadena a un *int* y la función *strcpy* para copiar una cadena en otra.

Ejemplo de funcionamiento:

```
ejercicio4.exe 2 hola 1 adios 0 edu 4 sistemas
Dimensión=7
matriz[0]->hola
matriz [1]->hola
matriz [2]->adios
matriz [3]->sistemas
matriz [4]->sistemas
matriz [5]->sistemas
matriz [6]->sistemas
```

SOLUCIÓN:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char *argv[]){
    char **matriz;
    int i,j;
    int dimension=0;
    int contador=0;
    // Determinar dimension a reservar
    for (i=1;i<argc;i=i+2){
        dimension+=atoi(argv[i]);
        }
    printf("Dimension=%d\n",dimension);
    // Reserva de memoria
    matriz = (char **)malloc(dimension*sizeof(char*));
    for (i=0;i<dimension;i++){
        matriz[i] = (char *)malloc(50*sizeof(char));
    }
    // Se copia cada palabra tantas veces como se indique
    for (i=1;i<argc;i=i+2) {
            for (j=0;j<atoi(argv[i]);j++) {
               strcpy(matriz[contador],argv[i+1]);
               contador++;
            }
        }
    // Se muestra la matriz
    for (i=0;i<dimension;i++) {
        printf("matriz[%d]->%s\n",i,matriz[i]);
    }
}
```

3. Dada la definición de una clase en C++ que permite crear vectores de coordenadas multidimensionales, se diseñarán los diferentes métodos y sobrecarga de operadores para trabajar con ella. En las operaciones entre clases se asumirá que tienen la misma dimensión. Al final del ejercicio tenéis la definición de la clase, el main y un ejemplo de funcionamiento. Se pide:

(a) Implementar el *constructor* reservando memoria dinámicamente en función de la dimensión. *(0.5 puntos)*

(b) Implementar el *destructor* liberando la memoria necesaria. *(0.25 puntos)*

(c) Realizar el método *rellenar* solicitando al usuario los datos para rellenar el vector. *(0.5 puntos)*

(d) Realizar el método *visualizar* para mostrar los datos del vector. *(0.5 puntos)*

(e) Sobrecargar el *operador* - para obtener una nueva clase_coordenada cuyas coordenadas sean el punto medio entre los dos vectores (punto a punto). *(0.5 puntos)*

(f) Sobrecargar el *operador* = para poder guardar el resultado de la resta anterior en una nueva clase. *(0.5 puntos)*

ejercicio_clase.h

```
class clase_coordenadas{
   protected:
       int dimension; // Dimension del vector
       double *vector; // Declaro el puntero al vector
   public:
       //Constructor con dimension por defecto 3
       clase_coordenadas(int dim=3);
       ~clase_coordenadas(void); //Destructor
       // Sobrecarga de operador - e = para guardar el resultado
       clase_coordenadas operator - (clase_coordenadas &der);
       clase_coordenadas& operator = (const clase_coordenadas &der);

       // Metodos rellenar y visualizar vector
       void rellenar(void);
       void visualizar(void);
};
```

ejercicio.cpp

```
#define __MAINMATRIZ_CPP__
#include <stdlib.h>
#include <iostream>
#include "ejercicio_clase.h"
using namespace std;
int main(void){
      clase_coordenadas v1(2),v2(2),v3(2);
// Creo 3 clases
      // Relleno y visualizo la primera
      v1.rellenar();
      v1.visualizar();
      // Relleno y visualizo la segunda
      v2.rellenar();
      v2.visualizar();
      v3=v1-v2;  //  Uso  operador  resta
sobrecargado  para  crear  una  nueva  con
punto medio
      v3.visualizar();
      system("pause");
      return 0;
}
```

Ejemplo de ejecución:

```
Introduzca          los
elementos del vector:
Elemento [0]: 5
Elemento [1]: 3
Se     muestran     los
elementos:
Elemento [0]: 5
Elemento [1]: 3
Introduzca          los
elementos del vector
Elemento [0]: 6
Elemento [1]: 2
Se     muestran     los
elementos:
Elemento [0]: 6
Elemento [1]: 2
Se     muestran     los
elementos:
Elemento [0]: 5.5
Elemento [1]: 2.5
Presione una tecla para
continuar . . .
```

SOLUCIÓN:

```
#define __CLASEMATRIZ_CPP__
#include <stdlib.h>
#include <iostream>
#include "ejercicio1_clase.h"
using namespace std;

// Constructor donde reservo memoria para el vector en función de la
dimensión
clase_coordenadas::clase_coordenadas(int dim){
      dimension=dim;
      vector = new double[dimension];
}

// Destructor donde se libera memoria del vector
clase_coordenadas::~clase_coordenadas(){
      delete [] vector;
}

// Método para pedir valores del vector en función de la dimensión
void clase_coordenadas::rellenar(void){
      int posicion;
      cout << "Introduzca los elementos del vector\n";
      for(posicion=0;posicion<dimension;posicion++){
            cout << "Elemento [" << posicion << "]: ";
            cin >> vector[posicion];
      }
}

// Método para mostrar los valores del vector
void clase_coordenadas::visualizar(void){
      int posicion;
      cout << "\n Se muestran los elementos:\n";
      for(posicion=0;posicion<dimension;posicion++)
            cout << "Elemento [" << posicion << "]: "<<vector[posicion]
<< "\n";
}

// Sobrecarga del operador - para obtener el punto medio entre los
// vectores coordenada a coordenada si tienen la misma dimensión
clase_coordenadas   clase_coordenadas::operator   -   (clase_coordenadas
&der)
{
      clase_coordenadas resultado(*this);
      int posicion;
      for(posicion = 0; posicion < der.dimension; posicion++)
                  resultado.vector[posicion]=(vector[posicion]-
der.vector[posicion])/2+ der.vector[posicion];
      return(resultado);
}

// Sobrecarga del operador = para igualar una clase a otra
clase_coordenadas&      clase_coordenadas::operator      =      (const
clase_coordenadas &der)
{
      int posicion;
      for(posicion = 0; posicion < der.dimension; posicion++)
                  vector[posicion]=der.vector[posicion];
      return (*this);
}
```

4. En este ejercicio se leerá un fichero estructurado sobre rutas y se mostrará información sobre él. En el fichero tendremos, en primer lugar, un número que indica cuantas rutas tenemos almacenadas (máximo 10). A continuación, tendremos la siguiente información de cada ruta: el nombre de la ruta (máximo 50 caracteres); la longitud de la ruta (float); y un campo llamado circular para determinar si la ruta es circular o no (int). Se pide:

(a) Definir la estructura y reservar memoria para un array de estructuras (máximo 10 posiciones). *(0.5 puntos)*

(b) Escribir el código necesario para abrir el fichero, leer su contenido e ir guardándolo en la estructura. *(0.75 puntos)*

(c) Hacer una función que reciba como parámetros el vector de estructuras y la cantidad de rutas. La función mostrará el nombre de la ruta más larga circular y el nombre de la ruta más larga no circular. Si el fichero no contiene información sobre algún tipo de ruta debe indicarse. *(1.25 puntos)*

NOTA: se puede usar la función *atoi* para convertir una cadena a un *int* y *atof* para convertir una cadena a un *float*.

entrada.txt	**entrada2.txt**
6 Ruta de la caldera 15 0 Ruta del agua 18 1 Barranco del infierno 13 0 Pantano de Elche 24 1 Cruz de la Muela 12 1 Transilicitana 100 0	3 Ruta de la caldera 15 0 Barranco del infierno 13 0 Transilicitana 100 0
Ejemplo de funcionamiento: Leidas 6 rutas Ruta circular mas larga: - Pantano de Elche Ruta no circular mas larga: - Transilicitana	**Ejemplo de funcionamiento:** Leidas 3 rutas Ruta circular mas larga: - No hay ninguna ruta de este tipo en el fichero Ruta no circular mas larga: - Transilicitana

SOLUCIÓN:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct rutas {
char nombre[50]
float longitud;
int circular;
   };
void informacion_rutas(struct rutas [], int cantidad);

int main(int argc, char *argv[]){
struct rutas lista[10];
int n,i,valor;
```

```
char aux[50];
FILE *fichero;
// Se abre el fichero
fichero = fopen("entrada.txt","r");
if (fichero == NULL) {
     printf("No ha sido posible abrir el fichero\n");
     return -1;
         }
// Se lee el primer numero que indica la cantidad de rutas
fgets(aux,50,fichero);
n=atoi(aux);

// Se lee la información de cada ruta
// y se almacena en el vector de estructuras
for (i=0;i<n;i++){
     fgets(lista[i].nombre,50,fichero);
     fgets(aux,50,fichero);
     lista[i].longitud=atof(aux);
     fgets(aux,50,fichero);
     lista[i].circular=atoi(aux);
    }
printf("Leidas %d rutas\n",n);
informacion_rutas(lista,n);
fclose(fichero);
system("PAUSE");
return(0);
}

void informacion_rutas(struct rutas lista[], int cantidad){
int I,max_circular=-1,max_no_circular=-1;
for (i=0;i<cantidad;i++){
     if (lista[i].circular==1){ // Caso rutas circulares
          if (max_circular==-1){
// Si es la primera que nos encontramos la guardamos como la mayor
               max_circular=i;
          } // Nos quedamos con el índice de la mayor
          else if (lista[i].longitud>lista[max_circular].longitud){
               max_circular=i;
              }
         }
     else{ // Se hace lo mismo para rutas no circulares
          if (max_no_circular==-1){
               max_no_circular=i;
              }
          else if (lista[i].longitud>lista[max_no_circular].longitud){
               max_no_circular=i;
              }
         }
    }
// En caso de no haberse encontrado ruta de algún tipo se indica
printf("Ruta circular mas larga: ");
if (max_circular==-1)
     printf("No hay ninguna ruta de este tipo en el fichero\n");
 else
     printf("%s",lista[max_circular].nombre);
printf("Ruta circular mas larga: ");
if (max_no_circular==-1)
     printf("No hay ninguna ruta de este tipo en el fichero\n");
 else
     printf("%s",lista[max_no_circular].nombre);
}
```