

FUNDAMENTOS DE PROGRAMACIÓN (Ing. Informática)
FUNDAMENTOS DE PROGRAMACIÓN I (Ing. Tec. Telemática)
1ª Convocatoria. 30 de Enero de 2006

NORMAS DE EXAMEN:

La asignatura se califica según el siguiente criterio: 60% nota del examen, 40% nota de laboratorio. Se exige una nota mínima de 4.0 en el examen para realizar el promedio. Notas inferiores a 4.0 en este ejercicio implican suspender la asignatura.

Si no se ha asistido a prácticas o no se han entregado los trabajos solicitados, la calificación de laboratorio será 0 (con un peso del 40% sobre la nota final).

Tiempo: Para realizar este examen se dispone de 2 horas de tiempo

1. La concejalía de deportes de una ciudad tiene una serie de instalaciones deportivas que pueden ser utilizadas por los ciudadanos empadronados en la ciudad y que sean socios. El archivo consta de una serie de fichas (en papel) que contienen la siguiente información: **DNI, NOMBRE, EDAD, DOMICILIO Y SERVICIOS**, donde servicios es una lista de servicios que desea utilizar el socio (tenis, aerobio, natación, pilates,..etc). El número máximo de servicios que puede utilizar un socio es de **cinco**. La concejalía desea informatizar este archivo y te contrata para que realices la aplicación. En este ejercicio no se pide que realices la aplicación completa, pero si que:
- Definas la estructura que va a almacenar la información de la ficha (y otros tipos que consideres necesarios para la definición de la estructura).
 - Que definas el vector (de tamaño máximo 1000) que va a contener todas las fichas.
 - Que escribas el prototipo de la función que busca un socio en el vector a partir de su DNI.
 - Que escribas el prototipo de la función que permite introducir información de un nuevo socio.
 - Que escribas el prototipo **e implementes** la función que calcula la cuota anual que debe pagar un socio (la cuota anual depende del número de servicios que usa el socio a razón de 100 euros por servicio).

APARTADO a):

```
const int MAX_FICHAS = 1000;
const int MAX_SERVICIOS = 5;

typedef string Servicios[MAX_SERVICIOS];

struct socio{
    string DNI;
    string nombre;
    int edad;
    string domicilio;
    Servicios serv;
}
```

APARTADO b):

```
typedef socio Fichas[MAX_FICHAS];
```

APARTADO c):

```
// Busca un socio en las fichas. Devuelve el indice
// del vector de fichas en el que se encuentra el socio
```

```
int busca_socio(const Fichas f, const string DNI)
```

APARTADO d):

```
// Introduce informacion sobre un nuevo socio (paso por referencia):
```

```
void nuevo_socio(socio &soc)
```

APARTADO e):

```
// Voy a suponer que se tiene una constante CUOTA_SERVICIO que contiene  
// el precio de la cuota anual de cada servicio. También sería posible  
// pasarlo como un argumento a la función.
```

```
float calcula_cuota(const socio soc){  
    float cuota;  
    int i;  
  
    for (i==0; i < MAX_SERVICIOS; i++)  
        // Voy a suponer que la no existencia de servicio se denota  
        // mediante una cadena que contiene un espacio en blanco.  
        if (soc.serv[i] != " ")  
            cuota = cuota + 1;  
    cuota = cuota * CUOTA_SERVICIO;  
    return cuota;  
}
```

2. Escribe una función que dada una cadena de texto calcule el número de palabras de una letra, de dos letras, de tres letras, etc. que aparecen en esa cadena, y guarde esta información en un vector definido de la siguiente manera:

```
typedef int VTam[MAX];  
void LongitudPalabras(string cadena, VTam v);
```

Por ejemplo, la frase:

El dinero no da la felicidad pero procura una sensación tan parecida que necesita un especialista muy avanzado para verificar la diferencia

tiene 0 palabras de 1 letra, 6 palabras de 2 letras, 3 de 3 letras.....

```
void LongitudPalabras(string cadena,vTam v)  
{  
    int i, ini, fin, tam;  
    bool continuar = true;  
  
    //Inicializar vector  
    for(i = 0; i < 20; i++)  
        v[i]=0;  
  
    //busco espacios porque separan palabras  
    ini = 0;  
    do{  
        fin = cadena.find(' ',ini);  
        if ( fin >=0) //si he encontrado un espacio  
        {  
            tam = fin -ini;  
  
            v[tam -1]++; //incremento el contador correspondiente  
  
            ini = fin + 1; //actualizo para la siguiente búsqueda  
  
        } else //si estoy al final de la palabra  
        {  
            fin = cadena.length(); //el caso particular de la ultima palabra  
            tam = fin -ini;  
  
            v[tam -1]++;  
  
            continuar = false; //cambio la variable para salir del bucle  
        }  
    }while(continuar);  
  
    return;  
}
```

3. Escribe una función en C++ que reciba como parámetros de entrada un vector de elementos parcialmente lleno y el tamaño del vector ocupado (el número de elementos del vector con valores) y devuelva como resultado el número de elementos que se repiten dentro del vector. El tipo del vector se define de la siguiente manera:

```
typedef int Vector[1000];  
Ejemplo:  
Si el vector contiene los siguientes valores  
{2, 7, 6, 1, 6, 2, 2, 9 }  
 ^   ^   ^   ^   ^  
la función devuelve como resultado 5.
```

```
//la versión más simple, no la más eficiente  
int CuentaElementosRepetidos ( Vector v, int tam )  
{  
    int i, j;  
    int elem_repetidos = 0;  
    bool repetido;  
  
    for ( i = 0; i < tam; i ++ ) //recorro el vector buscando repetidos  
    {  
        repetido = false;  
        j = 0;  
  
        while ( !repetido && j < tam )  
        {  
            if ( i != j && v[i] == v[j] )  
                repetido = true;  
  
            j++;  
        }  
  
        if ( repetido )  
            elem_repetidos ++;  
    }  
  
    return elem_repetidos;  
}
```

4. En un programa para jugar al Oteló se permite guardar el estado de las partidas para poder reanudar el juego a partir de una jugada concreta. El formato con el que se guarda la información es el siguiente: una cabecera que incluye información sobre el tamaño del tablero, el jugador que debe continuar la partida y un nombre identificativo de la partida y, a continuación, el estado del tablero de juego ('.' = casilla libre, 'O' = ficha blanca, 'X' = ficha negra). Cada dato almacenado viene precedido por una etiqueta que lo identifica. Las únicas etiquetas que pueden aparecer en el archivo son: #tamanyo, #jugador, #nombre y #tablero. Detrás de cada etiqueta aparece el valor asociado, excepto para la etiqueta #tablero, para la que los datos aparecen a partir de la siguiente línea del archivo. Las etiquetas #tamanyo, #jugador y #nombre constituyen la cabecera del archivo y siempre preceden a la etiqueta #tablero (no hay ningún archivo donde esto no ocurra). Las tres etiquetas de la cabecera pueden aparecer en distinto orden en diferentes archivos. Así, por ejemplo, es posible tener 3 archivos con los siguientes contenidos y todos representar la misma partida:

#tamanyo 8	#jugador 1	#nombre Oteló-45
#jugador 1	#nombre Oteló-45	#jugador 1
#nombre Oteló-45	#tamanyo 8	#tamanyo 8
#tablero	#tablero	#tablero
.....
.....
...X....	...X....	...X....
...XX...	...XX...	...XX...
...XO...	...XO...	...XO...
.....
.....
.....

Se pide, escribir la función que permita cargar la información de estos archivos en memoria para poder continuar el juego. El prototipo de la función debe de ser el siguiente:

```
int cargar_partida (tablero& t);
```

El valor entero que devuelve la función indica el jugador que debe continuar la partida y el tipo tablero, al cual pertenece el argumento t, responde a las siguientes declaraciones:

```
const int MAX_TAM = 10;
typedef char tablero_grande [MAX_TAM][MAX_TAM];
struct tablero
{
    tablero_grande casillas;
    int tam;
};
```

La información representada por la etiqueta #nombre no es preciso guardarla en el programa, puesto que sólo tiene sentido para diferenciar unos archivos de otros.

Nota: Se debe realizar una única función que sea capaz de cargar ficheros de cualquiera de tipos anteriores.

```
int cargar_partida (tablero& t)
{
    ifstream fichero;
    string linea, etiqueta;
    int jugador = 0;

    cout << "Nombre del archivo: ";
    cin >> nombre;

    fichero.open(nombre.c_str());
    if ( ! fichero )
```

```
        cerr << "Error. No es posible abrir el archivo." << endl;
    else
    {
        do
        {
            fichero >> etiqueta;
            if ( etiqueta == "#tamanyo" )
            {
                fichero >> t.tam;
                fichero.ignore(); //quitar el <CR>
            }
            else
            if ( etiqueta == "#nombre" )
                getline (fichero, linea);
            else
            if ( etiqueta == "#jugador" )
            {
                fichero >> jugador;
                fichero.ignore(); //quitar el <CR>
            }
        }
        while ( etiqueta != "#tablero" );

        fichero.ignore(); //quitar el <CR>
        for (int i = 0; i < t.tam; i++)
        {
            getline (fichero, linea);
            for (int j = 0; j < t.tam; j++)
                t.casillas[i][j] = linea[j];
        }
        fichero.close();
    }

    return (jugador);
}
```