

RECOPILACIÓN RESUELTA EJERCICIOS EXÁMENES

(1.5 pts) 2/- Un antiguo método de cifrado se basa en el uso de una clave alfanumérica, compuesta por todas las letras minúsculas (sin incluir la letra ñ) y todos los dígitos, dispuesta en una tabla bidimensional de 6×6 elementos como en el siguiente ejemplo:

	0	1	2	3	4	5
0	p	k	a	f	5	v
1	e	o	9	t	y	0
2	s	3	z	7	d	j
3	r	b	n	u	m	1
4	2	w	4	h	8	g
5	c	x	6	q	i	l

Diseña un procedimiento *cifrar* con tres parámetros. El primero es de entrada para recibir una clave almacenada en una tabla de cifrado como la especificada anteriormente. El segundo parámetro también es de entrada para recibir una cadena de caracteres conteniendo el texto a cifrar. El tercer parámetro es de salida y será una cadena de caracteres conteniendo el texto cifrado. El procedimiento se realiza de la siguiente forma:

- Para cada par de caracteres del texto de entrada (texto a cifrar), se producen también dos caracteres en el texto cifrado, según el siguiente proceso:
 - Se buscan ambos caracteres en la tabla de la clave (*se puede suponer que los dos caracteres estarán en la tabla*), devolviendo los índices de la fila y columna donde está almacenado cada carácter ($f1, c1$) y ($f2, c2$). Por ejemplo, usando la clave mostrada anteriormente, para el par de caracteres *ab*, se obtienen las coordenadas (0, 2) y (3, 1) de la tabla.
 - Se obtienen los caracteres que se encuentran en la tabla en las posiciones ($f1, f2$) y ($c1, c2$). Siguiendo con el ejemplo de clave anterior, en estas coordenadas se encuentran los caracteres *f3* según la tabla.
 - Estos nuevos caracteres se añaden al final del texto cifrado (tercer parámetro).
- Se repite el proceso anterior por cada par de caracteres del texto de entrada. En caso de que el texto de entrada tenga un número de caracteres impar, se desecha el último carácter del mismo, y no será tenido en cuenta.
- Se puede suponer que en el texto de entrada sólo aparecerán letras minúsculas y dígitos. También se puede suponer que la tabla clave también es correcta (aparecen todas las letras minúsculas y todos los dígitos).

Importante: sólo se completará el código del procedimiento *cifrar* en el fichero *ejercicio2.cpp* proporcionado en el campus virtual. Puedes añadir más procedimientos o funciones si lo estimas necesario. No se debe modificar el resto del código proporcionado.

La ejecución del código suministrado (tras diseñar el procedimiento) será:

El texto cifrado es: wbc6e4j89e

```

#include <iostream>
#include <array>
#include <string>
using namespace std;
const int N=6;
typedef array <char,N>TFilas;
typedef array <TFilas,N>TMatriz;

void aCero(TMatriz& mat){
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            mat[i][j]=0;
        }
    }
}

void leerDatos(TMatriz& mat,string& secuencia){
    cout<<"Introduzca la matriz clave: "<<endl;
    for(int fi=0;fi<N;fi++){
        for(int co=0;co<N;co++){
            cin>>mat[fi][co];
        }
    }

    cout<<"Introduzca su secuencia de caracteres: ";
    cin>>secuencia;
}

```

```

void buscar(TMatriz& mat, char letra,int& fi,int& co){
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            if(letra==mat[i][j]){
                fi=i;
                co=j;
            }
        }
    }
}

void cifrar(TMatriz& mat,string secuencia,string& cifrado){
    int j=0,tam,f1,c1,f2,c2;
    cifrado="";
    for(int i=0;i<secuencia.size();i=i+2){
        buscar(mat,secuencia[i],f1,c1);
        buscar(mat,secuencia[i+1],f2,c2);
        cifrado[j]=mat[f1][f2];
        cifrado[j+1]=mat[c1][c2];
        j=j+2;
    }

    for(int k=0;k<secuencia.size();k++){
        cout<<cifrado[k];
    }
}

int main(){
    TMatriz mat;
    string secuencia,cifrado;
    aCero(mat);
    leerDatos(mat,secuencia);
    cifrar(mat,secuencia,cifrado);
    return 0;
}

```

✓ (4 ptos) 4.- Diseña un algoritmo que lea de teclado

- un número natural k
- y una colección de valores enteros para completar una matriz de tamaño $N \times M$ (siendo N y M dos constantes definidas en el programa).

A partir de los datos leídos, el algoritmo calculará y mostrará por pantalla los k valores de la matriz que más veces se repiten.

Notas:

- Si k es mayor que el número de valores distintos de la matriz, entonces se mostrarán todos esos valores distintos (ver ejemplo 2).
- Si hay valores de la matriz que se repiten igual número de veces y alguno de ellos no se puede mostrar porque de hacerlo se excedería el valor de k , da igual los valores que queden sin mostrarse (ver ejemplo 3).

Ejemplos:

Ejemplo 1)

Entrada:

Introduzca k: 4

Introduzca la matriz 3x4 (por filas):

45 -17 867 45

2 867 -17 3

1 -2 45 3

Salida:

Los valores que mas se repiten son: 45 -17 867 3

Explicación:

Ya que para $k = 4$ y la matriz ($N = 3$, $M=4$):

45	-17	867	45
2	867	-17	3
1	-2	45	3

Se mostrarán los números (da igual el orden): 45 -17 867 3

Ejemplo 2)

Entrada:

Introduzca k: 8

Introduzca la matriz 3x4 (por filas):

45 -17 867 45

2 867 -17 3

1 -2 45 3

Salida:

Los valores que mas se repiten son: 45 -17 867 3 2 1 -2

Explicación:

Ya que para la misma matriz y $k = 8$, se mostrarían los números (da igual el orden): 45 -17 867 3 2 1 -2

(Aunque se pide mostrar 8 números, sólo se muestran 7, los que hay)

Ejemplo 3)

Entrada:

```
Introduzca k: 3
Introduzca la matriz 3x4 (por filas):
45 -17 867 45
2 867 -17 3
1 -2 45 3
```

Salida:

```
Los valores que mas se repiten son: 45 -17 867
```

Explicación:

Ya que para la misma matriz y $k = 3$, se mostrarían los números (da igual el orden): 45 -17 867

(También podrían mostrarse: 45 -17 3) (También podrían mostrarse: 45 867 3) ...

```
#include <iostream>
#include <array>
using namespace std;
const int N=3;
const int M=4;
typedef array <int,M>TFilas;
typedef array <TFilas,N>TMatriz;
typedef array <int,N*M>TDatos;

void aCero(TMatriz& mat,TDatos& v){
    for(int i=0;i<N;i++){
        for(int j=0;j<M;j++){
            mat[i][j]=0;
        }
    }
    for(int k=0;k<N*M;k++){
        v[k]=0;
    }
}

void leerDatos(TMatriz& mat, int& k){
    cout<<"Introduzca un natural k: ";
    cin>>k;

    cout<<"Introduzca una matriz "<<N<<"x"<<M<<": "<<endl;
    for(int fi=0;fi<N;fi++){
        for(int co=0;co<M;co++){
            cin>>mat[fi][co];
        }
    }
}
```

```
int calcularRep(TMatriz& mat, int num) {  
    int cont=0;  
    for(int i=0; i<N; i++) {  
        for(int j=0; j<M; j++) {  
            if(mat[i][j]==num) {  
                cont++;  
            }  
        }  
    }  
    return cont;  
}  
  
bool esta(TDatos& v, int num) {  
    int cont=0;  
    for(int i=0; i<N*M; i++) {  
        if(num==v[i]) {  
            cont++;  
        }  
    }  
    return cont!=0;  
}
```

```

void encontrarMayorRep (TMatriz& mat, int k, TDatos& v) {
    int mayorRep=0, copia, cont=0, numero, z=0;
    cout<<"Los " <<k<<" valores que mas se repiten son: ";
    while (cont<k) {
        for (int i=0; i<N; i++) {
            for (int j=0; j<M; j++) {
                if (!esta (v, mat[i][j])) {
                    copia=calcularRep (mat, mat[i][j]);
                    if (copia>mayorRep) {
                        mayorRep=copia;
                        numero=mat[i][j];
                    }
                }
            }
        }
        if (!esta (v, numero)) {
            v[z]=numero;
            z++;
        }
        mayorRep=0;
        cont++;
    }
}

```

```

void escribir (TDatos& v, int k) {
    for (int i=0; i<k; i++) {
        if (v[i]!=0) {
            cout<<v[i]<<" ";
        }
    }
}

```

```

int main () {
    TMatriz mat;
    TDatos v;
    int k;
    aCero (mat, v);
    leerDatos (mat, k);
    encontrarMayorRep (mat, k, v);
    escribir (v, k);

    return 0;
}

```

(2 pts) 2. ✓ Una matriz de números enteros *doblemente estocástica normalizada* es aquella que cumple las siguientes condiciones:

1. Todos sus elementos son no negativos y menores que 100.
2. La suma de los elementos de cada fila es igual a 100.
3. La suma de los elementos de cada columna es igual a 100.

Diseña un algoritmo que lea de teclado una colección de números enteros con los que rellenar una matriz cuadrada de tamaño NxN (siendo N una constante definida), muestre por pantalla después el contenido de dicha matriz y finalmente indique si dicha matriz es o no *doblemente estocástica normalizada*. Recuerda que en la solución se valorará la eficiencia de la misma y el uso de diseño descendente.

Dos ejemplos de ejecución (N = 3):

20	30	50
50	0	50
30	70	0

```
Introduce los numeros enteros para una matriz cuadrada de 3x3:
```

```
20 30 50
```

```
50 0 50
```

```
30 70 0
```

```
La matriz introducida es:
```

```
20 30 50
```

```
50 0 50
```

```
30 70 0
```

```
La matriz introducida SI es doblemente estocastica normalizada
```

20	30	50
55	-5	50
25	75	0

```
Introduce los numeros enteros para una matriz cuadrada de 3x3:
```

```
20 30 50
```

```
55 -5 50
```

```
25 75 0
```

```
La matriz introducida es:
```

```
20 30 50
```

```
55 -5 50
```

```
25 75 0
```

```
La matriz introducida NO es doblemente estocastica normalizada
```



```
#include <iostream>
#include <array>
using namespace std;
const int N=3;
typedef array <int,N>TFilas;
typedef array <TFilas,N>TMatriz;
```

```
void aCero(TMatriz& mat){
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            mat[i][j]=0;
        }
    }
}
```

```
void leerDatos(TMatriz& mat){
    cout<<"Introduzca una matriz "<<N<<"x"<<N<<": "<<endl;
    for(int fi=0;fi<N;fi++){
        for(int co=0;co<N;co++){
            cin>>mat[fi][co];
        }
    }
}
```

```
bool elementos(const TMatriz& mat) {
    int cont=0;
    for(int fi=0;fi<N;fi++){
        for(int co=0;co<N;co++){
            if(mat[fi][co]>=0&&mat[fi][co]<100){
                cont++;
            }
        }
    }

    return cont==N*N;
}

int sumaFila(const TMatriz& mat,int numFila){
    int suma=0;
    for(int co=0;co<N;co++){
        suma=suma+mat[numFila][co];
    }

    return suma;
}

int sumaColumna(const TMatriz& mat,int numColum){
    int suma=0;
    for(int fi=0;fi<N;fi++){
        suma=suma+mat[fi][numColum];
    }

    return suma;
}
```


```

bool sumas(const TMatriz& mat) {
    int sumaFi, sumaCo, cont=0;
    for(int i=0; i<N; i++) {
        sumaFi=sumaFila(mat, i);
        sumaCo=sumaColumna(mat, i);
        if(sumaCo==100) {
            cont++;
        }
        if(sumaFi==100) {
            cont++;
        }
    }

    return cont==N*2;
}

int main() {
    TMatriz mat;
    bool elem, sum;
    aCero(mat);
    leerDatos(mat);
    elem=elementos(mat);
    sum=sumas(mat);
    if(elem&&sum) {
        cout<<"La matriz SI es doblemente estocastica normalizada.";
    } else {
        cout<<"La matriz NO es doblemente estocastica normalizada.";
    }
    return 0;
}

```

(1 pto) .- Diseña un **algoritmo** que lea de teclado una secuencia de longitud indefinida de números enteros positivos separados por un espacio y terminada en 0, y muestre por pantalla el *mayor de los números perfectos* leídos. Un *número perfecto* es aquel cuyo valor coincide con la suma de sus divisores (excepto él mismo). Por ejemplo, el 28 es un número perfecto porque $28 = 1+2+4+7+14$. En caso de que no haya ningún número perfecto en la secuencia de entrada, se mostrará un mensaje indicándolo. Puedes suponer que la entrada es correcta.

Importante: La puntuación de este problema será de 1 punto sólo en el caso de que el algoritmo funcione correctamente. En otro caso, la puntuación será de 0 puntos.

Dos ejemplos de ejecución:

```
Introduzca una secuencia de enteros positivos acabada
en 0: 2 12 6 15 28 42 9 0
```

```
El mayor numero perfecto de la secuencia es: 28
```

```
Introduzca una secuencia de enteros positivos acabada
en 0: 1 2 15 8 25 36 9 0
```

```
No hay ningun numero perfecto en la secuencia
```

```
#include <iostream>
#include <array>
using namespace std;
const int MAX=10;
typedef array<int,MAX>TVector;

struct TNumeros{
    TVector datos;
    int num;
};

void aCero(TNumeros& v){
    for(int i=0;i<MAX;i++){
        v.datos[i]=0;
    }
    v.num=0;
}

void leerDatos(TNumeros& v){
    int valor,i=0;

    cout<<"Introduzca una secuencia de enteros positivos acabada en 0: ";
    cin>>valor;
    while(valor!=0){
        v.datos[i]=valor;
        v.num++;
        i++;
        cin>>valor;
    }
}
```

```

bool esPerfecto(int num) {
    int suma=0;
    for(int i=1;i<num;i++){
        if(num%i==0){
            suma=suma+i;
        }
    }

    return suma==num;
}

void mayorPerfecto(TNumeros& v) {
    int mp=0;
    for(int i=0;i<v.num;i++){
        if(esPerfecto(v.datos[i])&&v.datos[i]>mp){
            mp=v.datos[i];
        }
    }
    if(mp==0){
        cout<<"No hay ningun numero perfecto en la secuencia.";
    }else{
        cout<<"El mayor numero perfecto de la secuencia es: "<<mp;
    }
}

int main() {
    TNumeros v;
    aCero(v);
    leerDatos(v);
    mayorPerfecto(v);
    return 0;
}

```

(2 ptos) 2✓ Dada una constante definida N mayor o igual que 2, **diseña un algoritmo** que lea de teclado una colección de números enteros con los que rellenar una matriz cuadrada $M1$ de tamaño $N \times N$. A continuación, leerá de teclado los índices de una fila y una columna de la matriz $M1$ (fil y col). Hay que asegurarse que dichos valores son válidos. Con esta información, construirá otra matriz cuadrada $M2$ de tamaño $(N-1) \times (N-1)$ que almacenará los mismos números de la matriz $M1$, exceptuando los que forman la fila fil y la columna col de la misma. Finalmente, mostrará por pantalla el contenido de la matriz $M2$.

Ejemplo de ejecución:

```
Introduce los numeros enteros para una matriz cuadrada de 4x4:
```

```
4 -8 32 15
```

```
12 9 -5 -8
```

```
-4 7 41 65
```

```
6 45 -8 92
```

```
Introduce los indices de la fila y columna:
```

```
1 2
```

```
La matriz construida 3x3 es:
```

```
4 -8 15
```

```
-4 7 65
```

```
6 45 92
```

```

#include <iostream>
#include <array>
using namespace std;
const int N=4;
typedef array <int,N> TFilas;
typedef array <TFilas,N> TMatriz;

void aCero(TMatriz& mat){
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            mat[i][j]=0;
        }
    }
}

void leerDatos(TMatriz& mat,int& fil, int& col){
    cout<<"Introduzca una matriz "<<N<<"x"<<N<<": "<<endl;
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            cin>>mat[i][j];
        }
    }
    do{
        cout<<"Introduzca los indices de la fila y la columna: ";
        cin>>fil>>col;
    }while(fil>N||col>N);
}

```



```


void nuevaMatriz(TMatriz& mat,int fil,int col){

    cout<<"La nueva matriz "<<N-1<<"x"<<N-1<<" es: "<<endl;

    for(int i=0;i<N;i++){
        if(i!=fil){
            for(int j=0;j<N;j++){
                if(j!=col){
                    cout<<mat[i][j]<<" ";
                }
            }
            cout<<endl;
        }
    }
}

int main(){
    TMatriz mat;
    int fil,col;
    aCero(mat);
    leerDatos(mat,fil,col);
    nuevaMatriz(mat,fil,col);
    return 0;
}

```

(3.5 pts)  3.- Diseña un algoritmo para gestionar las apuestas que realiza un grupo de amigos sobre el resultado de un determinado partido de fútbol.

Para ello, el algoritmo deberá leer de teclado las apuestas realizadas por cada persona (el nombre, como una cadena de caracteres sin espacios en blanco, el resultado, como un número natural, y la cantidad de dinero apostada, la cual es un número real), hasta que se introduzca como nombre **FIN**, considerando que no habrá más de $MAX = 10$ personas distintas. En caso de que la misma persona realice múltiples apuestas, se puede suponer que todas son con el mismo resultado, por lo que se unificarán las apuestas de la misma persona en una única apuesta con la suma de las cantidades apostadas. Los resultados posibles son cero (0) en caso de empate, uno (1) en caso de que gane el primer equipo, y dos (2) en caso de que gane el segundo equipo.

A continuación, debe leer de teclado el resultado del partido (0, 1 o 2).

Podemos suponer que los datos de entrada son correctos.

Una vez leídos todos los datos, se mostrará por pantalla para cada persona su nombre, el resultado por el que apuesta y la cantidad de dinero total apostada (ver ejemplo de ejecución).

Después, realizará un reparto del dinero total apostado entre las personas que han acertado el resultado, de forma proporcional a las cantidades apostadas por cada uno. Para ello, realizará los siguientes cálculos:

1. Calcula T1 como la suma total de las cantidades apostadas.
2. Calcula T2 como la suma total de las cantidades apostadas que han acertado el resultado.
3. Calcula la ratio que corresponde a cada unidad apostada con acierto, es decir el cociente real resultado de dividir los dos totales anteriores $T1 / T2$.

Finalmente, mostrará por pantalla los valores calculados anteriormente, las apuestas de cada persona y el reintegro que le corresponde a cada una, siendo cero en caso de fallar el resultado o el dinero proporcional correspondiente en caso de acertar el resultado (la cantidad apostada multiplicada por la ratio calculada anteriormente). Consulta el ejemplo de ejecución para ver cómo aparecerán por pantalla todos estos datos.

Ejemplo de ejecución:

Entrada:

```
Introduzca nombres, resultados y cantidades apostadas (FIN para terminar)
Nombre: luis
Resultado (0 1 2): 0
Cantidad (> 0): 2
Nombre: lola
Resultado (0 1 2): 1
Cantidad (> 0): 3
Nombre: juan
Resultado (0 1 2): 2
Cantidad (> 0): 4
Nombre: ana
Resultado (0 1 2): 2
Cantidad (> 0): 2
Nombre: luis
Resultado (0 1 2): 0
Cantidad (> 0): 1
Nombre: lola
Resultado (0 1 2): 1
Cantidad (> 0): 6
Nombre: FIN
```

```
Introduzca el resultado final de la apuesta (0 1 2): Resultado (0 1 2): 2
```

Salida:

```
luis 0 3
lola 1 9
juan 2 4
ana 2 2
```

```
Total apostado: 18
Total ganador: 6
Ratio: 3
```

```
luis 0 3 -> 0
lola 1 9 -> 0
juan 2 4 -> 12
ana 2 2 -> 6
```

```
#include <iostream>
#include <array>
#include <string>
using namespace std;
const int MAX=10;
```

```
struct TPersona{
    string nombre;
    int gastos=0;
    int resul;
};

typedef array<TPersona,MAX>TPersonas;
```

```
struct TDatos{
    TPersonas persona;
    int npers=0;
};
```

```
bool esta(TDatos& v,string& nombre){
    int i=0;
    while(i<v.npers&&v.persona[i].nombre!=nombre){
        i++;
    }
    return i<v.npers;
}
```

```

int totalApostado(const TDatos& v) {
    int suma=0;
    for(int i=0;i<v.npers;i++){
        suma=suma+v.persona[i].gastos;
    }
    return suma;
}

int totalGanador(const TDatos& v,int resultado){
    int suma=0;
    for(int i=0;i<v.npers;i++){
        if(v.persona[i].resul==resultado){
            suma=suma+v.persona[i].gastos;
        }
    }
    return suma;
}

int calcularRatio(const TDatos& v,int ta,int tg){
    int radio=ta/tg;
    return radio;
}

int buscarInd(const TDatos& v,string& nombre){
    int ind=0;
    while(ind<v.npers&&v.persona[ind].nombre!=nombre){
        ind++;
    }
    return ind;
}

```

```

void leerDatos(TDatos& v,int& resultado){
    string nombre;
    int ind,dinero;
    v.npers=0;

    cout<<"Introduzca nombres, resultados y cantidades apostadas (FIN para te);
    cout<<"Nombre: ";
    cin>>nombre;
    while (nombre!="FIN"&&v.npers<MAX) {
        if (!esta(v,nombre)) {
            v.persona[v.npers].nombre=nombre;
            cout<<"Resultado: ";
            cin>>v.persona[v.npers].resul;
            cout<<"Cantidad: ";
            cin>>v.persona[v.npers].gastos;
            v.npers++;
        }else{
            ind=buscarInd(v,nombre);
            cout<<"Resultado: ";
            cin>>v.persona[ind].resul;
            cout<<"Cantidad: ";
            cin>>dinero;
            v.persona[ind].gastos=v.persona[ind].gastos+dinero;
        }

        cout<<"Nombre: ";
        cin>>nombre;
    }

    cout<<"Introduzca el resultado final de la apuesta: ";
    cin>>resultado;
}

void calcularResultados(TDatos& v, int resultado){
    int ta,tg,dinero,rt;
    for(int i=0;i<v.npers;i++){
        cout<<v.persona[i].nombre<<" "<<v.persona[i].resul<<" "<<v.persona[i].gastos;
        cout<<endl;
    }

    ta=totalApostado(v);
    tg=totalGanador(v,resultado);
    rt=calcularRatio(v,ta,tg);
    cout<<"Total apostado: "<<ta<<endl;
    cout<<"Total ganador: "<<tg<<endl;

    for(int j=0;j<v.npers;j++){
        cout<<v.persona[j].nombre<<" "<<v.persona[j].resul<<" "<<v.persona[j].gastos<<"->";
        if(v.persona[j].resul==resultado){
            dinero=v.persona[j].gastos*rt;
            cout<<dinero;
        }else{
            cout<<"0";
        }
        cout<<endl;
    }
}

```

```

int main() {
    TDatos v;
    int resultado;
    leerDatos(v, resultado);
    cout<<endl;
    calcularResultados(v, resultado);
    return 0;
}

```

(3.5 pts) 4-✓ Diseña un algoritmo que lea de teclado una secuencia indefinida de palabras terminada en FIN y las muestre por pantalla ordenadas en orden creciente por su longitud. Si hay varias palabras con una misma longitud, en la salida aparecerán en el mismo orden relativo que tenían esas palabras en la entrada. En la salida no habrá palabras repetidas.

Ejemplo de ejecución:

Entrada:

```

Introduzca un texto (FIN para terminar): CREO QUE VOY A IR ESTA
TARDE AL CINE Y LUEGO VOY A IR A CENAR MAS TARDE FIN

```

Salida:

```

Las palabras ordenadas de menor a mayor longitud son:

A Y IR AL QUE VOY MAS CREO ESTA CINE TARDE LUEGO CENAR

```

NOTAS:

- El texto contiene un número indefinido de palabras.
- El texto termina con la palabra FIN.
- Cada palabra tiene un número indefinido pero limitado de caracteres (todos alfabéticos mayúsculas).
- En el texto habrá un número máximo MAX_PAL_DIST (una constante) de palabras distintas.
- El carácter separador de palabras es el espacio en blanco.

```

#include <iostream>
#include <array>
#include <string>
using namespace std;
const int MAX_PAL_DIST=20;
typedef array <string,MAX_PAL_DIST>TDatos;
struct TPalabras{
    TDatos palabra;
    int nPal;
};

bool esta(const TPalabras& v,string& palabra){
    int i=0;
    while(i<v.nPal&&v.palabra[i]!=palabra){
        i++;
    }
    return i<v.nPal;
}

void leerDatos(TPalabras& v){
    string palabra;
    cout<<"Introduzca una texto (FIN para terminar): "<<endl;
    cin>>palabra;
    v.nPal=0;
    while(palabra!="FIN"&&v.nPal<MAX_PAL_DIST){
        if(!esta(v,palabra)){
            v.palabra[v.nPal]=palabra;
            v.nPal++;
        }
        cin>>palabra;
    }
}

```



```
void ordenar(TPalabras& v) {  
    int tam=1, cont=0;  
    while(cont<v.nPal) {  
        for(int i=0; i<v.nPal; i++) {  
            if(int(v.palabra[i].size())==tam) {  
                cout<<v.palabra[i]<<" ";  
                cont++;  
            }  
        }  
        tam++;  
    }  
}  
  
int main() {  
    TPalabras v;  
    leerDatos(v);  
    ordenar(v);  
    return 0;  
}
```

- (4 puntos) 1.- Diseñe un algoritmo que lea un texto terminado con la palabra "FIN" y muestre por pantalla *cuántas palabras y cuáles* cumplen la propiedad de estar asociadas por vocales fantasmas a la primera palabra que aparece dentro de dicho texto. Sean u y v dos palabras formadas por letras mayúsculas. Diremos que u está asociada a v por vocales fantasmas, si u se puede obtener a partir de v después de un cambio de orden de las letras que **mantenga inalterado el orden relativo de las consonantes**. Por ejemplo, PERLA está asociada por vocales fantasmas a PARLE, APERL, PEARL pero no a LEPRAL, al no conservarse el orden relativo de las consonantes (P - R - L).

1^{er} Ejemplo de Ejecución:

Introduzca un texto acabado en FIN, donde la primera palabra será el patrón a comprobar:
PERLA PARLE CALA APERL APRENDER PEARL LEPRAL PAERL PRALE FIN
El número de palabras asociadas por vocales fantasmas con el patrón PERLA es 5
Palabra n. 1 : PARLE
Palabra n. 2 : APERL
Palabra n. 3 : PEARL
Palabra n. 4 : PAERL
Palabra n. 5 : PRALE

2^o Ejemplo de Ejecución:

Introduzca un texto acabado en FIN, donde la primera palabra será el patrón a comprobar:
CASO ACASO CUSO COSA CESO COSA FIN
El número de palabras asociadas por vocales fantasmas con el patrón CASO es 1
Palabra n. 1 : COSA

Hay que tener en cuenta que:

- La salida del programa debe ser la mostrada en los ejemplos de ejecución: primero se muestra el número de palabras que cumplen la condición y luego la lista de palabras.
- El texto contiene un número indefinido de palabras.
- El texto contendrá al menos la palabra patrón.
- El texto termina con la palabra FIN.
- Cada palabra tiene un número indefinido pero limitado de caracteres (todos alfabéticos mayúsculas).
- El carácter separador de palabras es el espacio en blanco.
- En el texto aparecerán un número máximo MAX_PAL_DIST (una constante) de palabras que cumplen estar asociadas por vocales fantasma a la primera (patrón).
- En el texto podrá aparecer la misma palabra más de una vez. Si esa palabra está asociada por vocales fantasmas con el patrón sólo se mostrará por pantalla una sola vez.

```

#include <iostream>
#include <array>
#include <string>
using namespace std;
const int MAX_PAL_DIST=20;
typedef array <string,MAX_PAL_DIST>TPalabras;
struct TDatos{
    TPalabras palabra;
    int tam;
};

void borrar(string& palabra,int pos){
    string uno,dos;
    for(int i=0;i<pos;i++){
        uno=uno+palabra[i];
    }
    for(int i=palabra.size();i>pos;i--){
        dos=dos+palabra[i];
    }
    palabra=uno+dos;
}

bool cumpleFantasma(const string primera,const string palabra){
    bool cumple=true,ya=false;
    char letral,letra2;
    int pos,j=0;

    for(int i=0;i<primera.size();i++){
        if(primera[i]!='A'&&primera[i]!='E'&&primera[i]!='I'&&primera[i]!='O'&&primera[i]!='U'&&cumple){
            while(cumple&&!ya&&j<palabra.size()){
                if(palabra[j]!='A'&&palabra[j]!='E'&&palabra[j]!='I'&&palabra[j]!='O'&&palabra[j]!='U'){
                    if(primera[i]==palabra[j]){
                        cumple=true;
                        ya=true;
                        pos=j;
                    }else{
                        cumple=false;
                    }
                }
                j++;
            }
            j=pos+1;
            ya=false;
        }
    }

    return cumple;
}

bool esta(const TDatos& v,const string palabra){
    int i=0;
    while((i<v.tam)&&(v.palabra[i]!=palabra)){
        i++;
    }
    return i<v.tam;
}

void mostrar(TDatos& v,string primera){
    cout<<"El numero de palabras asociadas por vocales fantasmas con el patron "<<primera<<": "<<endl;
    for(int i=0;i<v.tam;i++){
        cout<<"Palabra n."<<i+1<<": "<<v.palabra[i];
        cout<<endl;
    }
}

```

```

int main() {
    TDatos v;
    string primera, palabra;

    cout<<"Introduzca un texto acabado en FIN: "<<endl;
    cin>>primera;
    v.tam=0;
    cin>>palabra;
    while (palabra!="FIN"&&v.tam<MAX_PAL_DIST) {
        if (!esta(v, palabra) && cumpleFantasma(primera, palabra)) {
            v.palabra[v.tam]=palabra;
            v.tam++;
        }
        cin>>palabra;
    }

    mostrar(v, primera);

    return 0;
}

```

- (2 ptos) 1.- Vamos a trabajar con listas de números enteros de hasta un tamaño máximo de MAX (una constante cualquiera). Define el tipo de datos TLista para ello y diseña un procedimiento criba que recibe como parámetros de entrada una lista de números enteros lista1 de tipo TLista y un número natural x. El procedimiento devolverá como parámetro de salida otra lista lista2 de tipo TLista que contendrá sólo aquellos números de lista1 que están repetidos x veces. En la lista lista2 no habrá elementos repetidos.

Ejemplo:

<p>El contenido de lista1 es: 1 3 4 3 1 3 0 -6 4 El valor de x es: 2 El contenido de lista2 será: 1 4</p>

```
#include <iostream>
#include <array>
using namespace std;
const int MAX=9;
typedef array <int,MAX>TLista;
```

```
void leerDatos(TLista& v,int& x){
    cout<<"Introduzca "<<MAX<<" numeros enteros: ";
    for(int i=0;i<MAX;i++){
        cin>>v[i];
    }
    cout<<"Introduzca el valor de x: ";
    cin>>x;
}
```

```
bool esta(TLista& v,int num){
    int i=0;
    while(i<MAX&&v[i]!=num){
        i++;
    }
    return i<MAX;
}
```

```
int repeticiones(const TLista& v,const int num){
    int cont=0;
    for(int i=0;i<MAX;i++){
        if(v[i]==num){
            cont++;
        }
    }
    return cont;
}
```

```

void criba(TLista& v1,TLista& v2,int x,int& j){
    int rep;
    j=0;

    for(int i=0;i<MAX;i++){
        rep=repeticiones(v1,v1[i]);
        if(rep==x&&!esta(v2,v1[i])){
            v2[j]=v1[i];
            j++;
        }
    }
}

void mostrar(TLista& v2,int j){
    cout<<"El contenido de lista2 sera: ";
    for(int i=0;i<j;i++){
        cout<<v2[i]<<" ";
    }
}

int main(){
    TLista lista1,lista2;
    int x,j;
    leerDatos(lista1,x);
    criba(lista1,lista2,x,j);
    mostrar(lista2,j);
    return 0;
}

```

(3.5 pts) 3. **Diseña** un algoritmo que lea de teclado una cadena de caracteres (sufijo) y un texto, y muestre por pantalla un listado de todas las palabras del texto que terminen con la cadena sufijo. En la salida no habrá palabras repetidas.

Ejemplo:

Entrada:

Introduzca el sufijo: OS

Introduzca el texto (FIN para terminar):

CREO QUE IREMOS A LA DIRECCION QUE NOS DIERON Y
TAMBIEN IREMOS A COMER TODOS AUNQUE PIENSO QUE TODOS
NO PODREMOS LLEGAR FIN

Salida:

Las palabras que terminan por OS son:

IREMOS NOS TODOS PODREMOS

NOTAS :

- El texto contiene un número indefinido de palabras.
- El texto termina con la palabra FIN.
- Cada palabra tiene un número indefinido pero limitado de caracteres (todos alfabéticos mayúsculas).
- En el texto aparecerán un número máximo MAX_PAL_DIST (una constante) de palabras distintas.
- El carácter separador de palabras es el espacio en blanco.

```
#include <iostream>
#include <array>
#include <string>
using namespace std;
const int MAX_PAL_DIST=20;
typedef array <string,MAX_PAL_DIST>TPalabras;
```

```
bool esta(const TPalabras& v,const string palabra,const int tam){
    int i=0;
    while(i<tam&&v[i]!=palabra){
        i++;
    }
    return i<tam;
}
```

```
bool cumpleSufijo(const string sufijo,const string palabra){
    int cont=0,tope;

    tope=palabra.size()-sufijo.size();

    for(int i=sufijo.size();i>0;i--){
        for(int j=palabra.size();j>tope;j--){
            if(sufijo[i]==palabra[j]){
                cont++;
            }
        }
    }

    return cont==sufijo.size();
}
```

```

void mostrar(TPalabras& v,int tam){
    for(int i=0;i<tam;i++){
        cout<<v[i]<<" ";
    }
}

int main(){
    TPalabras v;
    string sufijo,palabra;
    int tam=0;
    cout<<"Introduzca el sufijo: ";
    cin>>sufijo;

    cout<<"Introduzca el texto (FIN para terminar): ";
    cin>>palabra;
    while(palabra!="FIN"){
        if(cumpleSufijo(sufijo,palabra)&&!esta(v,palabra,tam)){
            v[tam]=palabra;
            tam++;
        }
        cin>>palabra;
    }

    mostrar(v,tam);

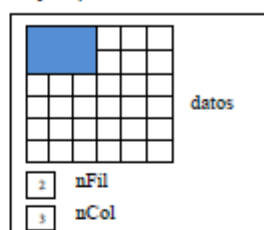
    return 0;
}

```


(2 pts) 1. ✓ Vamos a trabajar con **matrices** de números enteros de hasta un tamaño máximo de **MAX * MAX** (siendo **MAX** una constante cualquiera). Cada matriz podrá tener un número de filas (**nFil**) y columnas (**nCol**) diferente y siempre menor o igual a **MAX**. Para ello definimos el siguiente tipo **TMatriz**:

```
const unsigned MAX = 10;
typedef int Tfila[MAX];
typedef Tfila TArrayBid[MAX];
struct TMatriz {
    TArrayBid datos;
    unsigned nFil, nCol;
};
```

Ejemplo: matriz 2*3



Diseña un algoritmo (la función **main** y los procedimientos y funciones que sean necesarios) que lea de teclado dos matrices A y B, calcule la multiplicación $A * B$ y muestre el resultado por pantalla.

Notas:

- A la hora de leer una matriz primero se leerá el número de filas y de columnas que tiene (de forma reiterada hasta que ambos números sean menores o iguales que **MAX**) y después los números enteros que la forman.
- Para poder realizar el producto $A * B$, el número de columnas de A debe ser igual al número de filas de B. Si no se cumple esta condición, se mostrará un mensaje de error por pantalla, finalizando el programa.
- Recordemos que el resultado de multiplicar dos matrices A y B es otra matriz C, con el mismo número de filas que la matriz A y con el mismo número de columnas que la matriz B y en la que cada elemento $C[i][j]$ se calcula mediante la suma acumulada de los productos dos a dos de los elementos de la fila i-ésima de A por los correspondientes elementos de la columna j-ésima de B, es decir:

$$\sum_{k=0}^{columnas_de_A-1} A[i][k] * B[k][j]$$

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & \dots & b_{1p} \\ \dots & \dots & \dots \\ b_{n1} & \dots & b_{np} \end{bmatrix}$$

$$AB = \begin{bmatrix} a_{11}b_{11} + \dots + a_{1n}b_{n1} & \dots & a_{11}b_{1p} + \dots + a_{1n}b_{np} \\ \dots & \dots & \dots \\ a_{m1}b_{11} + \dots + a_{mn}b_{n1} & \dots & a_{m1}b_{1p} + \dots + a_{mn}b_{np} \end{bmatrix}$$

Ejemplo:

$$A \cdot B = \begin{pmatrix} 2 & 0 & 1 \\ 3 & 0 & 0 \\ 5 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix} =$$
$$= \begin{pmatrix} 2 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 & 2 \cdot 0 + 0 \cdot 2 + 1 \cdot 1 & 2 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 \\ 3 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 & 3 \cdot 0 + 0 \cdot 2 + 0 \cdot 1 & 3 \cdot 1 + 0 \cdot 1 + 0 \cdot 0 \\ 5 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 & 5 \cdot 0 + 1 \cdot 2 + 1 \cdot 1 & 5 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 \end{pmatrix} =$$
$$= \begin{pmatrix} 3 & 1 & 2 \\ 3 & 0 & 3 \\ 7 & 3 & 6 \end{pmatrix}$$

```
#include <iostream>
#include <array>
using namespace std;
const int MAX=10;
typedef int Tfila[MAX];
typedef Tfila TarrayBid[MAX];
struct TMatriz{
    TarrayBid datos;
    int nFil,nCol;
};

void leerDatos(TMatriz& a,TMatriz& b){
    cout<<"MATRIZ A: "<<endl;
    cout<<"Introduzca la dimension: ";
    cin>>a.nFil>>a.nCol;
    cout<<"Introduzca la matriz A: "<<endl;
    for(int i=0;i<a.nFil;i++){
        for(int j=0;j<a.nCol;j++){
            cin>>a.datos[i][j];
        }
    }

    cout<<"MATRIZ B: "<<endl;
    cout<<"Introduzca la dimension: ";
    cin>>b.nFil>>b.nCol;
    cout<<"Introduzca la matriz B: "<<endl;
    for(int k=0;k<b.nFil;k++){
        for(int z=0;z<b.nCol;z++){
            cin>>b.datos[k][z];
        }
    }
}
```

```

int suma(TMatriz& a,TMatriz& b,int fila,int columna){
    int suma=0;

    for(int i=0;i<a.nCol;i++){
        suma= suma + (a.datos[fila][i]*b.datos[i][columna]);
    }

    return suma;
}

void multiplicar(TMatriz& a,TMatriz& b,TMatriz& c){

    c.nFil=a.nFil;
    c.nCol=b.nCol;

    for(int i=0;i<c.nFil;i++){
        for(int j=0;j<c.nCol;j++){
            c.datos[i][j]=suma(a,b,i,j);
        }
    }
}

void mostrar(TMatriz& c){
    cout<<"MATRIZ C: "<<endl;

    for(int i=0;i<c.nFil;i++){
        for(int j=0;j<c.nCol;j++){
            cout<<c.datos[i][j]<<" ";
        }
        cout<<endl;
    }
}

int main(){
    TMatriz a,b,c;

    leerDatos(a,b);

    if(a.nFil==b.nCol){
        multiplicar(a,b,c);
        mostrar(c);
    }else{
        cout<<"No se pueden multiplicar.";
    }

    return 0;
}

```

(3.5 pts) 2.- **Diseña** la siguiente función:

```
unsigned numOcurrencias(const TNumeros& numeros,
                        const TPermutacion& permutacion)
```

La función recibe dos parámetros de entrada:

- `numeros`, un array de tipo `TNumeros` de tamaño `TAM1` que contiene números naturales. **Define** previamente el tipo `TNumeros`.
- `permutacion`, un array de tipo `TPermutacion` de tamaño `TAM2`, que contiene también números naturales. **Define** previamente el tipo `TPermutacion`.

La función devuelve el número de ocurrencias de cualquier permutación (mismos elementos aunque puedan estar en orden diferente) de los elementos del array `permutacion` en el array `numeros`, dispuestos de forma consecutiva.

Para calcular las ocurrencias del array `permutacion` en el array `numeros` se hace lo siguiente (para el ejemplo mostrado suponemos que `TAM1` es 11 y `TAM2` es 4 y que la parte sombreada del array `numeros` es lo que se comprueba en cada paso):

1) Se “coloca” el array `permutacion` sobre el array `numeros` en la posición 0 de éste y se comprueba si los elementos de `permutacion` son una permutación de la parte del array `numeros` sobre la que está colocado el array `permutacion`. Por ejemplo:

1	4	1	12	permutacion							SÍ hay ocurrencia
12	1	1	4	1	7	14	1	12	12	4	numeros

2) Se “desplaza” el array `permutacion` a la posición 1 sobre el array `numeros` y se hace la misma comprobación.

1	4	1	12	permutacion							NO hay ocurrencia
12	1	1	4	1	7	14	1	12	12	4	numeros

3) Se repite el mismo proceso mientras sea posible “colocar” el array `permutacion` sobre el array `numeros`.

Permutacion				1	4	1	12	No hay ocurrencia			
12	1	1	4	1	7	14	1	12	12	4	numeros

Para este ejemplo la función devolvería el valor 1, ya que hay una sola ocurrencia.

NOTAS:

- Podrá haber números repetidos en ambos arrays.
- Los elementos contenidos en los arrays serán siempre naturales mayores que 0.
- Pueden usarse arrays auxiliares si se considera necesario.

```

#include <iostream>
#include <array>
using namespace std;
const int TAM1=11;
const int TAM2=4;
typedef array <int,TAM1>TNumeros;
typedef array <int,TAM2>TPermutacion;

void leerDatos(TNumeros& n,TPermutacion& p){
    cout<<"Introduzca "<<TAM2<<" numeros naturales (permutacion): ";
    for(int i=0;i<TAM2;i++){
        cin>>p[i];
    }
    cout<<"Introduzca "<<TAM1<<" numeros naturales (numeros): ";
    for(int j=0;j<TAM1;j++){
        cin>>n[j];
    }
}

int frecuenciaN(const TNumeros& n,const int num,int pos){
    int cont=0;
    for(int i=pos;i<TAM2+pos;i++){
        if(n[i]==num){
            cont++;
        }
    }
    return cont;
}

int frecuenciaP(const TPermutacion& p,const int num){
    int cont=0;
    for(int i=0;i<TAM2;i++){
        if(p[i]==num){
            cont++;
        }
    }
    return cont;
}

bool cumplePermutacion(const TNumeros& n,const TPermutacion& p,const int pos){
    int frecp,frecn,i=pos;
    bool cumple=true;
    while(i<TAM2+pos&&cumple){
        frecp=frecuenciaP(p,n[i]);
        frecn=frecuenciaN(n,n[i],pos);
        if(frecn!=frecp){
            cumple=false;
        }
        i++;
    }
    return cumple;
}

```

```

int numOcurrencias(const TNumeros& n, const TPermutacion& p) {
    int cont=0;
    for(int i=0; i<TAM1; i++) {
        if(cumplePermutacion(n, p, i)) {
            cont++;
        }
    }
    return cont;
}

int main() {
    TNumeros n;
    TPermutacion p;
    int ocurrencias;

    leerDatos(n, p);
    ocurrencias=numOcurrencias(n, p);
    cout<<"La cantidad de ocurrencias es de: "<<ocurrencias<<".";
    return 0;
}

```

(3.5 pts) **3.-** Diseña un algoritmo que lea de teclado una cadena de caracteres (patrón) y un texto, y muestre por pantalla un listado de todas las palabras del texto que contengan la cadena patrón de “forma relativa”. Esto significa que la palabra contendrá todas las letras de la cadena patrón en el mismo orden, pero no tienen que estar dispuestas juntas. En la salida no habrá palabras repetidas.

Ejemplo:

Entrada:

Introduzca el patrón: EN

Introduzca el texto (FIN para terminar):

CREO QUE IREMOS A LA DIRECCION QUE NOS DIERON Y
TAMBIEN IREMOS A LA DIRECCION NUEVA QUE YO CONOCIA
FIN

Salida:

DIRECCION DIERON TAMBIEN

NOTAS:

- El texto contiene un número indefinido de palabras.
- El texto termina con la palabra FIN.
- Cada palabra tiene un número indefinido pero limitado de caracteres (todos alfabéticos mayúsculas).
- En el texto aparecerán un número máximo MAX_PAL_DIST (una constante) de palabras distintas.
- El carácter separador de palabras es el espacio en blanco.

```

#include <iostream>
#include <array>
#include <string>
using namespace std;
const int MAX_PAL_DIST=20;
typedef array <string,MAX_PAL_DIST>TPalabras;
struct TTexto{
    TPalabras palabra;
    int tam;
};

bool esta(const TTexto& v,const string& palabra){
    int i=0;
    while(i<v.tam&&v.palabra[i]!=palabra){
        i++;
    }
    return i<v.tam;
}

bool cumplePatron(const string& patron,const string& palabra){
    int i=0,j=0;
    bool cumple=true;
    bool ya=false;

    while(i<patron.size() && cumple){
        while(j<palabra.size() && !ya){
            if(patron[i]==palabra[j]){
                ya=true;
                cumple=true;
            }else{
                cumple=false;
                j++;
            }
        }
        ya=false;
        i++;
    }

    return cumple;
}

```



```

void leerDatos(TTexto& v, string& patron) {
    string palabra;
    v.tam=0;
    cout<<"Introduzca el patron: ";
    cin>>patron;
    cout<<"Introduzca el texto (FIN para terminar): "<<endl;
    cin>>palabra;
    while (palabra!="FIN") {
        if (!esta(v, palabra) && cumplePatron(patron, palabra)) {
            v.palabra[v.tam]=palabra;
            v.tam++;
        }
        cin>>palabra;
    }
}

void mostrar(TTexto& v, string patron) {
    cout<<"Las palabras que contienen el patron "<<patron<<" son: "<<endl;
    for(int i=0; i<v.tam; i++) {
        cout<<v.palabra[i]<<" ";
    }
}

int main() {
    TTexto v;
    string patron;
    leerDatos(v, patron);
    mostrar(v, patron);
    return 0;
}

```

(2 ptos) ✓.- El triángulo de Pascal es un triángulo de números naturales, infinito y simétrico. Se empieza con un 1 en la primera fila, y en las filas siguientes los números de los extremos son un 1 y cada uno de los restantes números se calcula como la suma de los dos números que tiene encima. Así, el triángulo con 7 filas sería:


```

      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
1 6 15 20 15 6 1

```

El triángulo de Pascal se puede almacenar en una matriz cuadrada girado 45° en el sentido contrario a las agujas del reloj, de forma que cada elemento de la primera fila y la primera columna de la matriz es un 1, los elementos a la derecha de la diagonal secundaria o inversa son un 0 y cada uno de los restantes números se calcula como la suma de los dos números que tiene a su izquierda y encima. Así, el triángulo anterior quedaría almacenado de la siguiente forma en una matriz 7x7:

1	1	1	1	1	1	1
1	2	3	4	5	6	0
1	3	6	10	15	0	0
1	4	10	20	0	0	0
1	5	15	0	0	0	0
1	6	0	0	0	0	0
1	0	0	0	0	0	0

Diseña un algoritmo que, para una constante dada TAM, construya el triángulo de Pascal en una matriz cuadrada TAMxTAM y muestre por pantalla el contenido de dicha matriz (los valores 0 no se muestran). Así, la ejecución del algoritmo para TAM = 7, mostraría por pantalla los siguientes números:

```

1 1 1 1 1 1 1
1 2 3 4 5 6
1 3 6 10 15
1 4 10 20
1 5 15
1 6
1

```

```

#include <iostream>
#include <array>
using namespace std;
const int TAM=7;
typedef array <int,TAM>TFilas;
typedef array <TFilas,TAM>TMatriz;

```

```

void aCero(TMatriz& mat){
    for(int i=0;i<TAM;i++){
        for(int j=0;j<TAM;j++){
            mat[i][j]=0;
        }
    }
}

```

```

void construirMatriz(TMtriz& mat){
    int suma=0, izq, arriba, tope;

    for(int i=0; i<TAM; i++){
        mat[0][i]=1;
        mat[i][0]=1;
    }

    for(int j=1; j<TAM; j++){
        for(int k=1; k<TAM; k++){
            tope=TAM-j;
            if(k<tope){
                izq=k-1;
                arriba=j-1;
                suma=mat[arriba][k]+mat[j][izq];
                mat[j][k]=suma;
            }
        }
    }
}

void mostrar(TMtriz& mat){
    for(int i=0; i<TAM; i++){
        for(int j=0; j<TAM; j++){
            if(mat[i][j]!=0){
                cout<<mat[i][j]<<" ";
            }
        }
        cout<<endl;
    }
}

int main(){
    TMtriz mat;
    aCero(mat);
    construirMatriz(mat);
    mostrar(mat);
    return 0;
}

```

(3.5 pts) 3.- Diseña un algoritmo que lea de teclado un texto, y muestre por pantalla un listado de todas las palabras del texto cuyos caracteres estén ordenados alfabéticamente en orden creciente. Puede haber más de una ocurrencia del mismo carácter en una palabra. En la salida no habrá palabras repetidas.

Ejemplo:

Entrada:

Introduzca el texto (FIN para terminar):

ELLOS CANTAN EL HIMNO DE SU EQUIPO DE FUTBOL CON
PASION Y YO NO CANTO EL HIMNO FIN

Salida:

Las palabras cuyos caracteres estan ordenados son:
ELLOS EL HIMNO DE SU Y NO

NOTAS:

- El texto contiene un número indefinido de palabras.
- El texto termina con la palabra FIN.
- Cada palabra tiene un número indefinido pero limitado de caracteres (todos alfabéticos mayúsculas).
- En el texto aparecerán un número máximo MAX_PAL_DIST (una constante) de palabras distintas.
- El carácter separador de palabras es el espacio en blanco.

```

#include <iostream>
#include <array>
#include <string>
using namespace std;
const int MAX_PAL_DIST=20;
typedef array <string,MAX_PAL_DIST>TPalabras;
struct TDatos{
    TPalabras palabra;
    int tam;
};

bool esta(const TDatos& v,const string palabra){
    int i=0;
    while(i<v.tam&&v.palabra[i]!=palabra){
        i++;
    }
    return i<v.tam;
}

bool estaOrdenado(const string& palabra){
    char anterior=palabra[0];
    bool ordenado=true;
    int i=1;
    while(i<palabra.size()&&ordenado){
        if(palabra[i]<anterior){
            ordenado=false;
        }
        anterior=palabra[i];
        i++;
    }
    return ordenado;
}

```

```

void leerDatos(TDatos& v) {
    string palabra;
    v.tam=0;
    cout<<"Introduzca el texto (FIN para terminar): "<<endl;
    cin>>palabra;
    while(palabra!="FIN") {
        if(estaOrdenado(palabra) && !esta(v, palabra)) {
            v.palabra[v.tam]=palabra;
            v.tam++;
        }
        cin>>palabra;
    }
    cout<<"Las palabras ordenadas: "<<endl;
    for(int i=0; i<v.tam; i++) {
        cout<<v.palabra[i]<<" ";
    }
}

int main() {
    TDatos v;
    leerDatos(v);
    return 0;
}

```

(3.5 pts) 4.- Deseamos simular el proceso de “Escalado de Imágenes” que muchos dispositivos (ej. televisiones) llevan a cabo para aumentar o disminuir la resolución de las mismas. Pensemos la imagen como una matriz M de F filas y C columnas y cuyo contenido en cada celda es un número natural mayor o igual que 1 (representando un determinado color). Para aumentar la resolución de la imagen, se construye una nueva matriz T con el doble de filas de M y con el mismo número de columnas. Para rellenar las $2 \cdot F$ filas de la matriz T se realiza el siguiente proceso:

- Primero se calcula la media entera (sin decimales) de todos los valores de M . Llamemos a este valor $mediaM$.
- Las filas de M pasan a ser las filas pares de T (la fila 0 de M se copia en la fila 0 de T , la fila 1 de M se copia en la fila 2 de T y así sucesivamente).
- Las filas impares de T se inicializan a 0. Posteriormente, se calculan los valores a almacenar en cada celda de cada fila impar de T de la siguiente manera: se recorre la fila de izquierda a derecha y en cada celda se almacena la media entera de los valores de los vecinos de esa celda (si alguno de esos vecinos es un 0, se toma como valor de ese vecino la media $mediaM$ antes calculada. Ver tercer paso del ejemplo mostrado a continuación).

Nota: Las celdas de las esquinas tienen sólo 3 vecinos; las de los bordes que no son esquinas tienen 5 y las restantes tienen 8. Hay que controlar bien esos cálculos.

Veamos un ejemplo:

Matriz M

3	2	5
6	5	3



Matriz T

3	2	5
4	4	3
6	5	3
5	4	4

Pasos para conseguir T

- Calculamos $mediaM = 4$ $((3+2+5+6+5+3) / 6 = 4)$
- Tras copiar las filas de M a las filas pares de T e inicializar las impares a 0 la matriz T queda de la siguiente forma:

3	2	5
0	0	0
6	5	3
0	0	0

- La primera fila impar de T se calcula de la siguiente forma:
 - La celda $(1,0)$ se calcula a partir de sus vecinos que son 5 (3, 2, 0, 6, 5). Tras sustituir el 0 por 4 (que es $mediaM$), sumarlos y dividir el resultado entre 5 nos da el valor 4 $((3+2+4+6+5) / 5 = 4)$. En la celda $(1,0)$ ponemos por tanto un 4.

- La celda (1,1) tiene 8 vecinos de los cuales uno es 0, que se sustituye por 4; la suma de todos es 32; que dividido entre 8 vecinos resulta $32/8=4$. En la celda (1,1) ponemos por tanto un 4.
- Para la celda (1,2) se suman sus 5 vecinos y el valor resultante es 3 $((2+5+4+5+3) / 5 = 3)$.

Tras estos cálculos la matriz T queda:

3	2	5
4	4	3
6	5	3
0	0	0

- Realizando un proceso similar para la otra fila impar, terminamos la matriz T:

3	2	5
4	4	3
6	5	3
5	4	4

Diseña un algoritmo que, para dos constantes dadas F y C, lea de teclado el contenido de la matriz M (F filas y C columnas), construya la matriz T (2 * F filas y C columnas) y muestre por pantalla el contenido de la matriz T. Así, la ejecución del algoritmo para F = 2, C = 3 y la matriz M ejemplo que hemos usado antes, mostraría por pantalla:

```
Introduzca la matriz M (2x3):
3 2 5
6 5 3

La matriz T (4x3) es la siguiente:
3 2 5
4 4 3
6 5 3
5 4 4
```

Notas:

- Se recomienda no utilizar el tipo unsigned, sino el tipo int para trabajar con valores de tipo entero y natural.
- Se recomienda ir probando la ejecución del algoritmo conforme se van acometiendo los diferentes pasos de la construcción de la matriz T.

```

#include <iostream>
#include <array>
using namespace std;
const int F=2;
const int C=3;
typedef array <int,C>TFilas;
typedef array <TFilas,F>TMatrizM;
typedef array <TFilas,2*F>TMatrizT;

void leerDatos(TMatrizM& m){
    cout<<"Introduzca una matriz " <<F<<"x"<<C<<": " <<endl;
    for(int i=0;i<F;i++){
        for(int j=0;j<C;j++){
            cin>>m[i][j];
        }
    }
}

int calcularMedia(const TMatrizM& m){
    int media,total=0;
    for(int i=0;i<F;i++){
        for(int j=0;j<C;j++){
            total=total+m[i][j];
        }
    }
    media=total/(C*F);
    return media;
}

bool esVecino(const int fi,const int co,const int i,const int j){
    bool vecino=false;

    if(fi==i&&co==j){
        vecino=false;
    }

    if((fi-i==0)&&((co-j==1)|| (j-co==1))){
        vecino=true;
    }

    if(((co-j==1)|| (j-co==1))&&((fi-i==1)|| (i-fi==1))){
        vecino=true;
    }

    if((co-j==0)&&((fi-i==1)|| (i-fi==1))){
        vecino=true;
    }

    return vecino;
}

```



```

int mediaVecinos(const TMatrizT& t, const int fi, const int co) {
    int media, cont=0, total=0;

    for(int i=0; i<F*2; i++) {
        for(int j=0; j<C; j++) {
            if(esVecino(fi, co, i, j)) {
                total=total+t[i][j];
                cont++;
            }
        }
    }

    media=total/cont;

    return media;
}

void mostrar(TMatrizT& t) {
    cout<<"Matriz T: "<<endl;
    for(int i=0; i<F*2; i++) {
        for(int j=0; j<C; j++) {
            cout<<t[i][j]<<" ";
        }
        cout<<endl;
    }
}

```

```

void construirMatrizT(TMatrizM& m,TMatrizT& t){
    int filaT,sig,mediaM,mediaV;
    mediaM=calcularMedia(m);

    //cout<<"PASO 1: "<<endl;
    for(int i=0;i<F;i++){
        filaT=2*i;
        sig=filaT+1;
        for(int j=0;j<C;j++){
            t[filaT][j]=m[i][j];
            t[sig][j]=0;
        }
    }

    //cout<<"PASO 2: "<<endl;//k,z
    for(int k=0;k<(F*2);k++){
        for(int z=0;z<C;z++){
            if(t[k][z]==0){
                t[k][z]=mediaM;
            }
        }
    }

    //cout<<"PASO 3: "<<endl;
    for(int fi=1;fi<F*2;fi=fi+2){
        for(int co=0;co<C;co++){
            mediaV=mediaVecinos(t,fi,co);
            t[fi][co]=mediaV;
        }
    }
}

int main(){
    TMatrizM m;
    TMatrizT t;

    leerDatos(m);
    construirMatrizT(m,t);
    mostrar(t);

    return 0;
}

```

(2 ptos) 2.- Diseña un algoritmo que lea de teclado números naturales para completar una matriz de tamaño $F \times C$ (siendo F y C dos constantes naturales establecidas) y muestre por pantalla todas las cimas que existen en la matriz. Para cada cima se mostrarán la fila, la columna y valor de la misma.

Una cima es un elemento de la matriz que es mayor o igual que los elementos vecinos que están a su izquierda, derecha, arriba y abajo (no se tienen en cuenta las diagonales). Un elemento puede tener 2, 3 o 4 vecinos según sea esquina, borde-no-esquina o no-borde-no-esquina, respectivamente.

Ejemplos ($F = 3$ y $C = 3$):

<p>Entrada: 4 5 3 6 2 2 1 8 7</p> <p>Salida: Las cimas de la matriz son: Fila 0 columna 1 valor 5 Fila 1 columna 0 valor 6 Fila 2 columna 1 valor 8</p>	<p>Entrada: 4 4 4 4 4 4 4 4 4</p> <p>Salida: Las cimas de la matriz son: Fila 0 columna 0 valor 4 Fila 0 columna 1 valor 4 Fila 0 columna 2 valor 4 Fila 1 columna 0 valor 4 Fila 1 columna 1 valor 4 Fila 1 columna 2 valor 4 Fila 2 columna 0 valor 4 Fila 2 columna 1 valor 4 Fila 2 columna 2 valor 4</p>
<p>Entrada: 1 2 3 4 5 6 7 8 9</p> <p>Salida: Las cimas de la matriz son: Fila 2 columna 2 valor 9</p>	<p>Entrada: 9 8 7 6 5 4 3 2 1</p> <p>Salida: Las cimas de la matriz son: Fila 0 columna 0 valor 9</p>

```

#include <iostream>
#include <array>
using namespace std;
const int F=3;
const int C=3;
typedef array <int,C>TFilas;
typedef array <TFilas,F>TMatriz;

void leerDatos(TMatriz& mat){
    cout<<"Introduzca una matriz "<<F<<"x"<<C<<": "<<endl;
    for(int i=0;i<F;i++){
        for(int j=0;j<C;j++){
            cin>>mat[i][j];
        }
    }
}

bool esVecino(const int fi, const int co, const int i, const int j){
    bool vecino=false;
    if(fi==i&&co==j){
        vecino=false;
    }

    if(fi-i==0&&(co-j==1||j-co==1)){
        vecino=true;
    }
    if(co-j==0&&(fi-i==1||i-fi==1)){
        vecino=true;
    }
    return vecino;
}

```

```

int mayorVecino(const TMatriz& mat, const int fi, const int co) {
    int mayorV=0;

    for(int i=0; i<F; i++) {
        for(int j=0; j<C; j++) {
            if(esVecino(fi, co, i, j) && mat[i][j] > mayorV) {
                mayorV=mat[i][j];
            }
        }
    }

    return mayorV;
}

void calcularCimas(TMatriz& mat) {
    int mayorV;
    cout<<"Las cimas de la matriz son: "<<endl;

    for(int i=0; i<F; i++) {
        for(int j=0; j<C; j++) {
            mayorV=mayorVecino(mat, i, j);
            if(mat[i][j] >= mayorV) {
                cout<<"Fila "<<i<<" columna "<<j<<" valor "<<mat[i][j]<<endl;
            }
        }
    }
}

int main() {
    TMatriz mat;
    leerDatos(mat);
    calcularCimas(mat);
    return 0;
}

```

(3.5 pts) 3.- Diseña un algoritmo que lea de teclado números enteros para completar una matriz de tamaño 9 x 9. Posteriormente comprobará si dicha matriz constituye un tablero de sudoku válido o no y mostrará por pantalla el mensaje correspondiente.

Un tablero de sudoku es una matriz de 9 x 9 casillas dividida en 9 regiones de 3 x 3 casillas cada una (ver ejemplo de abajo). Las reglas para que un tablero de sudoku sea válido son las siguientes:

- Cada *casilla* almacena un número comprendido entre 1 y 9 (ambos inclusive) o bien está vacía (para nosotros estará vacía si contiene un 0)
- En una misma *fila* no puede haber números repetidos (los 0 no cuentan).
- En una misma *columna* no puede haber números repetidos (los 0 no cuentan).
- En una misma *región* no puede haber números repetidos (los 0 no cuentan).

Un ejemplo de tablero de sudoku válido sería el siguiente:

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

región

```
#include <iostream>
#include <array>
using namespace std;
const int TAM=9;
typedef array <int,TAM>TFilas;
typedef array <TFilas,TAM>TMatriz;
typedef array <int,TAM/3>TFilasR;
typedef array <TFilasR,TAM/3>TRegion;

void leerDatos(TMatriz& mat){
    cout<<"Introduzca una matriz " <<TAM<<"x" <<TAM<<": " <<endl;
    for(int i=0;i<TAM;i++){
        for(int j=0;j<TAM;j++){
            cin>>mat[i][j];
        }
    }
}

bool cumpleCasilla(const TMatriz& mat){
    bool cumple=true;
    int i=0,j=0;
    while(i<TAM&&cumple){
        while(j<TAM&&cumple){
            if(mat[i][j]<0||mat[i][j]>TAM){
                cumple=false;
            }
            j++;
        }
        i++;
        j=0;
    }
    return cumple;
}
```

```
int frecuenciaFi(const TMatriz& mat, const int fi, const int num) {  
    int cont=0;  
    for(int i=0; i<TAM; i++) {  
        if(mat[fi][i]==num) {  
            cont++;  
        }  
    }  
    return cont;  
}  
  
bool cumpleFila(const TMatriz& mat) {  
    bool cumple=true;  
    int num=1, frec, fi=0;  
    while(fi<TAM && cumple) {  
        while(num<TAM && cumple) {  
            frec=frecuenciaFi(mat, fi, num);  
            if(frec>1) {  
                cumple=false;  
            }  
            num++;  
        }  
        num=1;  
        fi++;  
    }  
  
    return cumple;  
}
```

```

int frecuenciaCo(const TMatriz& mat, const int co, const int num) {
    int cont=0;
    for(int i=0; i<TAM; i++) {
        if(mat[i][co]==num) {
            cont++;
        }
    }
    return cont;
}

bool cumpleColumna(const TMatriz& mat) {
    bool cumple=true;
    int num=1, frec, co=0;
    while(co<TAM && cumple) {
        while(num<TAM && cumple) {
            frec=frecuenciaCo(mat, co, num);
            if(frec>1) {
                cumple=false;
            }
            num++;
        }
        num=1;
        co++;
    }

    return cumple;
}

```



```
void construirRegion(const TMatriz& mat, TRegion& r, int num) {
    int fi=0, co=0;

    if(num<3) {
        for(int i=0; i<3; i++) {
            for(int j=num*3; j<(num*3)+3; j++) {
                r[fi][co]=mat[i][j];
                co++;
            }
            fi++;
            co=0;
        }
    }

    if(num>=3&&num<6) {
        for(int i=3; i<6; i++) {
            for(int j=(num-3)*3; j<(num-3)*3+3; j++) {
                r[fi][co]=mat[i][j];
                co++;
            }
            fi++;
            co=0;
        }
    }

    if(num>=6) {
        for(int i=6; i<TAM; i++) {
            for(int j=(num-6)*3; j<(num-6)*3+3; j++) {
                r[fi][co]=mat[i][j];
                co++;
            }
            fi++;
            co=0;
        }
    }
}
```

```

int frecuenciaR(const TRegion& r, const int num) {
    int cont=0;
    for(int i=0; i<TAM/3; i++) {
        for(int j=0; j<TAM/3; j++) {
            if(r[i][j]==num) {
                cont++;
            }
        }
    }
    return cont;
}

bool cumpleRegion(TRegion& r, TMatriz& mat) {
    bool cumple=true;
    int num=1, frec, n=0;
    while(n<TAM) {
        construirRegion(mat, r, n);
        while(num<TAM && cumple) {
            frec=frecuenciaR(r, num);
            if(frec>1) {
                cumple=false;
            }
            num++;
        }
        num=1;
        n++;
    }

    return cumple;
}

int main() {
    TMatriz mat;
    TRegion r;
    bool casilla, fila, columna, region;

    leerDatos(mat);

    casilla=cumpleCasilla(mat);
    fila=cumpleFila(mat);
    columna=cumpleColumna(mat);
    region=cumpleRegion(r, mat);

    if(casilla && fila && columna && region) {
        cout<<"SI";
    } else {
        cout<<"NO";
    }

    return 0;
}

```

(3.5 pts) **4/4** Diseña un algoritmo que lea de teclado un patrón y un texto, y muestre por pantalla un listado de todas las palabras del texto indicando el número de caracteres que tienen en común con el patrón. Habrá que asegurarse que el patrón tenga una longitud determinada (definida por una constante (en el ejemplo de abajo tiene el valor 5)) y que no contenga letras repetidas. El orden en el que aparezcan las letras en el patrón y en la palabra no es importante. Si una letra del patrón aparece varias veces en una palabra del texto solo se cuenta una vez. **En la salida no habrá palabras repetidas.**

Ejemplo:

Entrada:

```
Introduzca un patron (long = 5, sin letras repetidas): EL
Introduzca un patron (long = 5, sin letras repetidas): CANTA
Introduzca un patron (long = 5, sin letras repetidas): CANTO
```

```
Introduzca un texto (FIN para terminar): ANTERIORMENTE IBA A TRABAJAR
EN TREN PERO AHORA VOY A TRABAJAR EN AUTOMOVIL FIN
```

Salida:

```
Palabras y numero de letras que coinciden con el patron:
ANTERIORMENTE 4
IBA 1
A 1
TRABAJAR 2
EN 1
TREN 2
PERO 1
AHORA 2
VOY 1
AUTOMOVIL 3
```

NOTAS:

- El texto contiene un número indefinido de palabras.
- El texto termina con la palabra FIN.
- Cada palabra tiene un número indefinido pero limitado de caracteres (todos alfabéticos mayúsculas).
- El patrón tendrá una longitud de TAM_CAR (una constante) caracteres.
- En el texto habrá un número máximo MAX_PAL_DIST (una constante) de palabras distintas.
- El carácter separador de palabras es el espacio en blanco.

```

#include <iostream>
#include <array>
#include <string>
using namespace std;
const int TAM_CAR=5;
const int MAX_PAL_DIST=20;
struct TPalabra{
    string palabra;
    int num=0;
};
typedef array<TPalabra,MAX_PAL_DIST>TPalabras;
struct TDatos{
    TPalabras datos;
    int tam;
};

bool repetidas(const string patron){
    int cont=0;
    char letra='A';
    bool cumple=true;
    while(letra<='Z'&&cumple){
        for(int i=0;i<patron.size();i++){
            if(patron[i]==letra){
                cont++;
            }
        }
        if(cont>1){
            cumple=false;
        }
        cont=0;
        letra++;
    }

    return cumple;
}

```

```

bool esta(const TDatos& v, const string palabra) {
    int i=0;
    while(i<v.tam&&v.datos[i].palabra!=palabra){
        i++;
    }
    return i<v.tam;
}

bool estaP(const string palabra, const char letra) {
    int i=0;
    while(i<palabra.size()&&palabra[i]!=letra){
        i++;
    }
    return i<palabra.size();
}

int caracteresComun(const string patron, const string palabra) {
    int cont=0;
    for(int i=0;i<patron.size();i++){
        if(estaP(palabra,patron[i])){
            cont++;
        }
    }
    return cont;
}

void mostrar(TDatos& v) {
    for(int i=0;i<v.tam;i++){
        cout<<v.datos[i].palabra<<" "<<v.datos[i].num<<endl;
    }
}

```

```

int main() {
    TDatos v;
    string patron, palabra;
    bool norep;
    v.tam=0;
    do{
        cout<<"Introduzca el patron (long = "<<TAM_CAR<<" , sin letras repetidas):"<<endl;
        cin>>patron;
        norep=repeticiones(patron);
    }while(patron.size() !=TAM_CAR || !norep);

    cout<<"Introduzca el texto (FIN para terminar): "<<endl;
    cin>>palabra;
    while(palabra!="FIN") {
        if(!esta(v,palabra)) {
            v.datos[v.tam].num=caracteresComun(patron,palabra);
            v.datos[v.tam].palabra=palabra;
            v.tam++;
        }
        cin>>palabra;
    }

    mostrar(v);

    return 0;
}

```

(1 pto) **✓**- Diseña una función `sumaMayoresImpares` que recibe como parámetro una matriz de `FIL` filas y `COL` columnas (`FIL` y `COL` dos constantes definidas) rellena de números naturales y devuelve la suma de los mayores números impares de cada fila de la misma. Si en una fila no hay números impares, el valor de esa fila para la suma es 0. Si en una fila el mayor número impar se repite varias veces, sólo se tiene en cuenta su valor una vez para la suma.

Importante: No debes modificar el código ya proporcionado en el campus virtual (*ejercicio1.cpp*) para probar la función. Debes completar el código de la función `sumaMayoresImpares`. Si para su diseño necesitas más procedimientos o funciones, puedes añadirlos. Si el código no funciona correctamente la puntuación de este problema será de 0 puntos.

La ejecución del código suministrado una vez diseñada la función `sumaMayoresImpares` mostrará por pantalla lo siguiente:

```

El contenido de la matriz de prueba es:
6 4 12 2
5 2 7 3
4 9 5 11

La suma de los mayores impares de las filas es: 18

```

```


#include <iostream>
#include <array>
using namespace std;
const int FIL=3;
const int COL=4;
typedef array <int, COL>TFilas;
typedef array <TFilas, FIL>TMatriz;

void leerDatos(TMatriz& mat){
    cout<<"Introduzca una matriz "<<FIL<<"x"<<COL<<" : "<<endl;
    for(int i=0;i<FIL;i++){
        for(int j=0;j<COL;j++){
            cin>>mat[i][j];
        }
    }
}

int sumaMayoresImpares(const TMatriz& mat){
    int mayor=0, suma=0;
    for(int i=0;i<FIL;i++){
        for(int j=0;j<COL;j++){
            if(mat[i][j]>mayor&&mat[i][j]%2!=0){
                mayor=mat[i][j];
            }
        }
        suma=suma+mayor;
        mayor=0;
    }
    return suma;
}

int main(){
    TMatriz mat;
    int suma;
    leerDatos(mat);
    suma=sumaMayoresImpares(mat);
    cout<<"La suma de los mayores impares de las filas es: "<<suma;
    return 0;
}

```

(2 ptos) .- Diseña una función `mayorLongitud` que recibe como parámetro un array unidimensional de tamaño `TAM` (una constante definida) relleno de números naturales y devuelve la longitud (número de valores) de la mayor sub-sucesión de números ordenados. A continuación diseña un programa principal (función

`main`) para probar el funcionamiento de dicha función. El programa leerá por teclado una secuencia de `TAM` números naturales y mostrará por pantalla la longitud de la mayor sub-sucesión ordenada de dicha secuencia.

Ejemplo 1: Con los valores del array de entrada (para `TAM = 10`): 42 7 56 87 64 2 35 8 9 0, obtendríamos las siguientes sub-sucesiones ordenadas: 42 ; 7 56 87 ; 64 ; 2 35 ; 8 9 ; 0, siendo la mayor sub-sucesión ordenada: 7 56 87, de longitud 3.

Ejemplo 2: Con los valores del array de entrada (para `TAM = 10`): 12 21 23 1 23 3 2 43 43 333, obtendríamos las siguientes sub-sucesiones ordenadas: 12 21 23 ; 1 23 ; 3 ; 2 43 43 333, siendo la mayor sub-sucesión ordenada: 2 43 43 333, de longitud 4.

Para estos ejemplos la ejecución del programa mostraría por pantalla lo siguiente:

```
Introduzca 10 numeros naturales: 42 7 56 87 64 2 35 8 9 0
La longitud de la mayor sub-sucesion es: 3
```

```
Introduzca 10 numeros naturales: 12 21 23 1 23 3 2 43 43 333
La longitud de la mayor sub-sucesion es: 4
```



```

#include <iostream>
#include <array>
using namespace std;
const int TAM=10;
typedef array <int,TAM>TVector;

void leerDatos(TVector& v){
    cout<<"Introduzca " <<TAM<<" numeros naturales: ";
    for(int i=0;i<TAM;i++){
        cin>>v[i];
    }
}

int mayorLongitud(const TVector& v){
    int mayortam=0,tam=1,anterior=v[0];

    for(int i=0;i<TAM;i++){
        if(v[i]>=anterior){
            tam++;
        }else{
            tam=1;
        }
        anterior=v[i];
        if(tam>mayortam){
            mayortam=tam;
        }
    }
    return mayortam;
}

int main(){
    TVector v;
    int mayorL;
    leerDatos(v);
    mayorL=mayorLongitud(v);
    cout<<"La longitud de la mayor subsucesion es: " <<mayorL;
    return 0;
}

```

(3.5 pts) **3.-** Diseña un algoritmo que lea de teclado primero una cadena de caracteres que constituirá un determinado patrón y después un texto. El algoritmo mostrará por pantalla un listado de todas las palabras del texto cuya suma de valores de sus caracteres en ASCII coincida con la suma de los valores de los caracteres de la cadena patrón. **En la salida no habrá palabras repetidas.**

Ejemplo:

Entrada:

Introduzca el patrón: CASA

Introduzca el texto (FIN para terminar):

SACA DE CASA EL CAKI SOLO BEBO AGUA EN CASA CON EL
CAKI FIN

Salida:

Las palabras que cumplen la condición son:

SACA CASA CAKI BEBO

NOTAS:

- El texto contiene un número indefinido de palabras.
- El texto termina con la palabra FIN.
- Cada palabra tiene un número indefinido pero limitado de caracteres (todos alfabéticos mayúsculas).
- En el texto aparecerán un número máximo MAX_PAL_DIST (una constante) de palabras distintas.
- El carácter separador de palabras es el espacio en blanco.

```

#include <iostream>
#include <array>
#include <string>
using namespace std;
const int MAX_PAL_DIST=20;
typedef array <string,MAX_PAL_DIST>TVector;
struct TPalabras{
    TVector palabra;
    int tam;
};

bool esta(const TPalabras& v,const string palabra){
    int i=0;
    while(i<v.tam&&v.palabra[i]!=palabra){
        i++;
    }
    return i<v.tam;
}

int sumaASCII(const string& palabra){
    int suma=0;
    for(int i=0;i<palabra.size();i++){
        suma=suma + int(palabra[i]);
    }
    return suma;
}

void mostrar(TPalabras& v){
    for(int i=0;i<v.tam;i++){
        cout<<v.palabra[i]<<" ";
    }
}

```

```

int main() {
    TPalabras v;
    int sumapatron, sumap;
    string patron, palabra;
    v.tam=0;
    cout<<"Introduzca el patron: ";
    cin>>patron;
    sumapatron=sumaASCII(patron);

    cout<<"Introduzca el texto (FIN para terminar): "<<endl;
    cin>>palabra;
    while(palabra!="FIN") {
        sumap=sumaASCII(palabra);
        if(sumap==sumapatron&&!esta(v,palabra)) {
            v.palabra[v.tam]=palabra;
            v.tam++;
        }
        cin>>palabra;
    }
    cout<<endl;
    cout<<"Las palabras que cumplen la condicion son: "<<endl;
    mostrar(v);

    return 0;
}

```