INFOR

- ESTRUCTURAS BASICAS:

  - VALIDACIONES

    - COMPROVAR ⟶ IF
    - VALIDAR ⟶ REPEAT. FOR. WHILE

  - SIN MENSAJE DE ERROR ⟶ REPEAT

    ADB

```
REPEAT
    write ('Introduce un nº positivo: ');
    Readln (iNum);
UNTIL  iNum > 0
```

  - CON MENSAJE DE ERROR ⟶ WHILE

    ADB

```
write ('Introduce un nº positivo');
Readln (iNum);
WHILE  iNum <= 0  DO
BEGIN
    writeln ('¡Error! El nº no es correcto');
    write ('Introduce un nº positivo: ');
    Readln(iNum);
END;
```

- REPETITIVAS

  << ¿Sabemos cuantas veces ejecuta? >>

  ↳ SÍ: FOR

  ↳ NO: << ¿Al menos cuantas? >>

    ↳ 1: REPEAT

    ↳ 0: WHILE

- SECUENCIA DE NUMEROS

ADB ——→ FACTORIA

```
¿Resul := 1 ——→ INICIALIZAR VBLE con el ELEMENTO NEUTRO
FOR ¿i:=n DOWNTO 1 DO
BEGIN
    ¿Resul := ¿Resul * ¿i ——→ ESTRUCTURA para resultado
END;
```

- SUBPROGRAMAS:

FUNCTION
- SIEMPRE GENERAN 1 (y solo 1) RESULTADO
- NUNCA PARAMETROS POR REFERENCIA

PROCEDURE
- NUNCA GENERAN RESULTADO
- PUEDEN RECIBIR PARAMETROS POR REFERENCIA

MENUA

```
PROGRAM menua;
USES Crt;
TYPE

 FUNCTION fnMenu:Char;
       VAR cOpcion: Char;
       BEGIN
           Writeln('1.- Guardar datos en el array.');
           Writeln('2.- Mostrar la media de los valores del array.');
           Writeln('3.- Visualizar el contenido del array (primero las posiciones impares y luego las pares)');
           Writeln('4.- Salir del programa');
           REPEAT
               Writeln('Elige un opcion: ');
               Readln(cOpcion);
           UNTIL (cOpcion>='1')AND(cOpcion<='4');
           fnMenu := cOpcion;
       END;

VAR
BEGIN
REPEAT
          cOpcion := fnMenu;
          CASE cOpcion OF
              '1': BEGIN
                   END;
              '2': BEGIN
                   END;
              '3': BEGIN
                   END;
          END;
          Writeln('Pulsa una INTRO para continuar...'); {Writeln('Pulsa una una tecla para
continuar...');}
          Readln; {ReadKey;}
          ClrScr; {Para poder usarlo tenemos que poner USES Crt;}
    UNTIL cOpcion='4';
    Readln;
    END.
```

ALGORITMOAK

1 - FITXATEGIAK

1.1 - CREAR SI NO EXISTE

1.2 - VOLCAR FICHERO → ARRAY

1.3 - VOLCAR ARRAY → FICHERO

1.4 - BUSCAR RG EN FICHERO y DEVOLVER POS.

1.5 - AÑADIR RG AL FINAL DEL FICHERO

1.6 - ACTUALIZAR CONTENIDO FICHERO

1.7 - DEVOLVER RG DE UNA POSICION DEL FICH.

1.8 - VISUALIZAR FICHERO

1.9 - OBTENER MAYOR ELEMENTO DE FICHERO

1.10 - OBTENER MAYOR ELEMENTO DE FICH. QUE CUMPLA COND.

1.11 - BORRAR RG DE F1 QUE CUMPLAN COND. Y PASAR A F2.

1.12 - BORRAR RG DE FICH. CON ARRAY

2 - ARRAYAK

2.1 - RELLENAR ARRAY HASTA QUE DIGA EL USUARIO

2.2 - BUSCAR ELEMENTO EN ARRAY → DEVOLVER POSICIÓN

2.3 - BORRAR ELEMENTO DE ARRAY Y DESPLAZAR A LA IZQ

2.4 - ELIMINAR CON CONDICIÓN [FOR DOWNTO · ← · / WHILE · → · / WHILE · ← ·]

2.5 - DEVOLVER MAYOR ELEMENTO DEL ARRAY POS

2.6 - ORDENAR UN ARRAY

2.7 - METER ELEMENTO DE FORMA ORDENADA

2.8 - HACER MEDIA DE LOS ELEMENTOS DEL ARRAY

2.9 RELLENAR MATRIZ

2.10. MOSTRAR MATRIZ

2.11. PEDIR DIMENISIONES MATRIZ

2.12 MOSTRAR MATRIZ POR COLUMNAS

2.13. RELLENAR ARRAY DE FORMA ORDENADA

2.14. ARRAYEKO ELEMENTU BAT EBAN ETA BORRATU

FITXATEGIAK

1.1. Procedimiento que crea el fichero si no existe.

```
PROCEDURE pCrearFicheroSiNoExiste(sNomfich: tsCad20);
VAR f: tfrgAlumno;
BEGIN
IF NOT fileexists(sNomfich) THEN
BEGIN
Assign(f,sNomfich);
Rewrite(f);
Close(f);
END;
END;
```

1.2. Procedimiento que vuelca el contenido del fichero a un array. PROCEDURE

```
pVolcarFicheroArray(sNomfich: tsCad20; VAR a: targAlumno; VAR iLong: Integer);
VAR f: tfrgAlumno;
rg :trgAlumno;
BEGIN
iLong := 0;
Assign(f,sNomfich);
Reset(f);
WHILE NOT eof(f) DO
BEGIN
Read(f,rg);
iLong := iLong + 1;
a[iLong] := rg;
END;
Close(f);
END;
```

1.3. Procedimiento que vuelca el contenido de un array al fichero. PROCEDURE

```
pVolcarArrayFichero(sNomfich: tsCad20; CONST a: targAlumno; iLong: Integer);
VAR f: tfrgAlumno;
i: Integer;
BEGIN
Assign(f,sNomfich);
Rewrite(f);
FOR i:=1 TO iLong DO
Write(f,a[i]);
Close(f);
END;
```

1.4. Función que busca en el fichero un registro y devuelve la posición en la que se encuentra o un -1 si no lo encuentra.

```
FUNCTION fnBuscarPosicionRegistro(sNomfich: tsCad20; sNom: tsCad20): Integer; VAR f:
tfrgAlumno;
rg: trgAlumno;
iPos: Integer;
boEnc: Boolean;
BEGIN
Assign(f,sNomfich);
Reset(f);
boEnc := FALSE;
WHILE (NOT eof(f)) AND (NOT boEnc) DO
BEGIN
Read(f,rg);
IF rg.sNom = sNom THEN
BEGIN
boEnc := TRUE;
END;
END;
IF boEnc THEN
iPos := filepos(f)-1
ELSE
iPos := -1;
Close(f);
fnBuscarPosicionRegistro := iPos;
END;
```

1.5. Procedimiento que añade un registro al final del fichero

```
PROCEDURE pAniadirRegistroAlFinal(sNomfich: tsCad20; CONST rg: trgAlumno); VAR f:
tfrgAlumno;
BEGIN
Assign(f,sNomfich);
Reset(f);
seek(f,filesize(f));
Write(f,rg);
Close(f);
END;
```

1.6. Procedimiento que actualiza el contenido del fichero.

```
PROCEDURE pActualizarRegistros(sNomfich: tsCad20);
VAR f: tfrgAlumno;
rg: trgAlumno;
BEGIN
Assign(f,sNomfich);
Reset(f);
WHILE NOT eof(f) DO
BEGIN
Read(f,rg);
rg.iEdad := rg.iEdad + 1;
seek(f,filepos(f)-1);
Write(f,rg);
END;
Close(f);
END;
```

1.7. Procedimiento que devuelve el registro situado en una determinada posición.

```
PROCEDURE pObtenerRegistroEnPosicion(sNomfich: tsCad20; iPos: Integer; VAR rg: trgAlumno);
VAR f: tfrgAlumno;
BEGIN
Assign(f,sNomfich);
Reset(f);
seek(f,iPos);
Read(f,rg);
Close(f);
END;
```

1.8. Procedimiento que visualiza el contenido de un fichero.

```
PROCEDURE pMostrarFichero(sNomfich: tsCad20);
VAR f: tfrgAlumno;
rg: trgAlumno;
BEGIN
Assign(f,sNomfich);
Reset(f);
WHILE NOT eof(f) DO
BEGIN
Read(f,rg);
WITH rg DO
Writeln(sNom:20,iEdad:10);
END;
Close(f);
END;
```

1.9. Procedimiento que obtiene el mayor elemento del fichero.

```
PROCEDURE pObtenerMayor(sNomfich: tsCad20; VAR rgMayor: trgAlumno); VAR f:
tfrgAlumno;
rg: trgAlumno;
boEnc: Boolean;
BEGIN
Assign(f,sNomfich);
Reset(f);
IF NOT eof(f) THEN
Read(f,rgMayor);
boEnc:=FALSE;
WHILE (NOT eof(f)) AND (NOT boEnc) DO
BEGIN
Read(f,rg);
IF rg.iEdad > rgMayor.iEdad THEN
BEGIN
rgMayor := rg;
END;
END;
Close(f);
END;
```

1.10. Procedimiento que obtiene el mayor elemento del fichero que cumpla una determinada condición.

```
PROCEDURE pObtenerMayorConCondicion(sNomfich: tsCad20; VAR rgMayor: trgAlumno); VAR f:
tfrgAlumno;
rg: trgAlumno;
boEnc: Boolean;
BEGIN
Assign(f,sNomfich);
Reset(f);
boEnc := FALSE;
WHILE (NOT eof(f)) AND (NOT boEnc) DO
BEGIN
Read(f, rgMayor);
IF rgMayor.iEdad MOD 2 = 0 THEN
boEnc:=TRUE;
END;
IF boEnc THEN
BEGIN
WHILE NOT eof(f) DO
BEGIN
Read(f,rg);
IF (rg.iEdad MOD 2 = 0) AND (rg.iEdad > rgMayor.iEdad) THEN
rgMayor := rg;
END;
END;
Close(f);
END;
```

1.11. Procedimiento que borra los registros del fichero que cumplen una determinada condición ayudándose de un fichero auxiliar.

```
PROCEDURE pBorrarDatosUsandoOtroFichero(sNomfich: tsCad20); VAR f1,f2:
tfrgAlumno;
rg: trgAlumno;
BEGIN
Assign(f1,sNomfich);
Assign(f2,'Aux.dat');
Reset(f1);
Rewrite(f2);
WHILE NOT eof(f1) DO
BEGIN
Read(f1,rg);
IF rg.iEdad >= 18 THEN
Write(f2,rg);
END;
Close(f1);
Close(f2);
Erase(f1);
Rename(f2,sNomfich);
END;
```

1.12. Procedimiento que borra los registros del fichero que cumplen una determinada condición ayudándose de un array auxiliar.

```
PROCEDURE pBorrarDatosUsandoUnArray(sNomfich: tsCad20);
VAR a: targAlumno;
iLong, iPos, i: Integer;
BEGIN
pVolcarFicheroArray(sNomfich,a,iLong);
iPos:=1;
WHILE iPos<=iLong DO
BEGIN
IF a[iPos].iEdad<18 THEN
BEGIN
FOR i:=iPos TO iLong-1 DO
a[i] := a[i+1];
iLong := iLong - 1;
END
ELSE
iPos := iPos + 1;
END;
pVolcarArrayFichero(sNomfich,a,iLong); END;
```

ARRAY ETA MATRIZEAK

2.1. Procedimiento en el que se le solicita al usuario números enteros hasta que diga que no quiere insertar más (máx 20) y se almacenan en un array.

```
PROCEDURE pRellenarArray(VAR aiNumeros: taiNumeros; VAR iLongitud: Integer); VAR
cResp: Char;
BEGIN
iLongitud := 0;
REPEAT
Write('Introduce un número: ');
iLongitud := iLongitud + 1;
Readln(aiNumeros[iLongitud]);
Write('¿Quieres insertar otro (S/N) ? ');
Readln(cResp);
cResp := UPCASE(cResp);
UNTIL (iLongitud = 20) OR (cResp = 'N');
END;
```

2.2. Función que, dado un array, su longitud y un número, busca ese número en el array, si lo encuentra devuelve la posición en la que está y si no lo encuentra, un -1. FUNCTION

```
fnBuscarElemento(CONST aiNumeros: taiNumeros; iLongitud,iNum: Integer):Integer;
VAR iPos: Integer;
boEnc: Boolean;
BEGIN
boEnc := FALSE;
iPos := 1;
WHILE (NOT boEnc) AND (iPos<=iLongitud) DO
BEGIN
IF aiNumeros[iPos] = iNum THEN
boEnc := TRUE
ELSE
iPos := iPos + 1;
END;
IF NOT boEnc THEN
iPos := -1;
fnBuscarElementos := iPos;
END;
```

2.3. Procedimiento que, dado un array, su longitud y una posición del mismo, borra del array el elemento situado en esa posición desplazando el resto de los elementos a la izquierda.

```
PROCEDURE pBorrarElemento(VAR aiNumeros: taiNumeros; VAR iLongitud: Integer; iPos: Integer);
VAR i: Integer;
BEGIN
FOR i:=iPos TO iLongitud - 1 DO
aiNumeros[i] := aiNumeros[i+1];
iLongitud := iLongitud - 1;
END;
```

2.4. Procedimiento que, dado un array y su longitud, elimina del mismo todos aquellos elementos

que sean negativos.

```
PROCEDURE pBorrarTodosLosNegativos(VAR aiNumeros: taiNumeros; VAR iLongitud: Integer);
VAR iPos,i: Integer;
BEGIN
iPos := 1;
WHILE iPos<=iLongitud DO
BEGIN
IF aiNumeros[iPos]<0 THEN
BEGIN
FOR i:=iPos TO iLongitud - 1 DO
aiNumeros[i] := aiNumeros[i+1];
iLongitud := iLongitud - 1;
END
ELSE
iPos := iPos+ 1;
END;
END;
```

2.5. Función que, dado un array y su longitud, devuelve la posición en la que se encuentra el mayor elemento del array.

```
FUNCTION fnObtenerPosicionMayor(CONST aiNumeros: taiNumeros; iLongitud: Integer): Integer;
VAR iMayor,iPos,i: Integer;
BEGIN
iMayor := aiNumeros[1];
iPos := 1;
FOR i:=2 TO iLongitud DO
IF aiNumeros[i] > iMayor THEN
BEGIN
iMayor := aiNumeros[i];
iPos := i;
END;
fnObtenerPosicionMayor := iPos;
END;
```

2.6. Procedimiento que, dado un array desordenado y su longitud, lo ordena ascendentemente.

```
PROCEDURE pOrdenarArray(VAR aiNumeros: taiNumeros; iLongitud: Integer); VAR
i,j,iAux: Integer;
BEGIN
FOR i:=1 TO iLongitud - 1 DO
FOR j:=i+1 TO iLongitud DO
IF aiNumeros[i] > aiNumeros[j] THEN
BEGIN
iAux := aiNumeros[i];
aiNumeros[i] := aiNumeros[j];
aiNumeros[j] := iAux;
END;
END;
```

2.7. Procedimiento que, dado un array ordenado, su longitud y un nuevo número, lo inserta en el array de forma que se mantiene el orden del array.

```
PROCEDURE pInsertarOrdenado(VAR aiNumeros: taiNumeros; VAR iLongitud: Integer;
```

```
iNum:Integer);
VAR iPos,i: Integer;
boEnc : Boolean;
BEGIN
iPos := 1;
boEnc := FALSE;
WHILE (NOT boEnc) AND (iPos<=iLongitud) DO
BEGIN
IF aiNumeros[iPos] > iNum THEN
boEnc:=TRUE
ELSE
iPos := iPos + 1;
END;
FOR i:=iLongitud+1 DOWNTO iPos+1 DO
aiNumeros[i] := aiNumeros[i-1];
aiNumeros[iPos] := iNum;
iLongitud := iLongitud + 1;
END;
```

2.8. Función que, dado un array y su longitud, devuelve la media de todos los elementos del array.

```
FUNCTION fnObtenerMedia(CONST aiNumeros: taiNumeros; iLongitud: Integer): Real; VAR
iSuma,i: Integer;
rMedia : Real;
BEGIN
iSuma := 0;
FOR i:=1 TO iLongitud DO
iSuma := iSuma + aiNumeros[i];
rMedia := iSuma / iLongitud;
fnObtenerMedia := rMedia;
END;
```

2.9. {Subprograma que rellena la matriz m1 con los 12 valores que inserte el usuario}

```
PROCEDURE pRellenarMatriz(VAR m1: tmEnteros);
VAR i,j: Integer; {i la usamos para recorrer las filas y j para recorrer las columnas} BEGIN
    {Recorremos la matriz de arriba a abajo y de izda a dcha}
    FOR i:=1 TO 3 DO
    BEGIN
        FOR j:=1 TO 4 DO
        BEGIN
            Write('Introduce el valor de la posición ',i,'-',j,': ');
            Readln(m1[i,j]);
        END;
    END;
END;
```

2.10. {Subprograma que muestra el contenido de la matriz}

```
PROCEDURE pMostrarMatriz(CONST m1: tmEnteros);
VAR i,j: Integer;
BEGIN
    Writeln('Los datos de la matriz m1 son: ');
    FOR i:=1 TO 3 DO
```

```
    BEGIN
        FOR j:=1 TO 4 DO
        BEGIN
            Write(m1[i,j]:6);
        END;
        Writeln;
    END;
END;
```

2.11. {Subprograma que le solicite al usuario las dimensiones de la matriz}

```
PROCEDURE pPedirDimensiones(VAR iNumFilas,iNumColumnas: Integer); BEGIN
    REPEAT
        Write('Introduce el número de filas de la matriz: ');
        Readln(iNumFilas);
    UNTIL (iNumFilas>0) AND (iNumFilas<=100);
    REPEAT
        Write('Introduce el número de columnas de la matriz: ');
        Readln(iNumColumnas);
    UNTIL (iNumColumnas>0) AND (iNumColumnas<=100);
END;
```

2.12. {Subprograma que muestra la matriz por columnas} PROCEDURE

```
pMostrarMatrizPorColumnas(CONST m:tmEnteros;
iNumFilas,iNumColumnas:Integer);
    VAR i,j: Integer; {i para las filas y la j para las columnas}
    BEGIN
        Writeln('El contenido de la matriz por columnas es: ');
        FOR j:=1 TO iNumColumnas DO
        BEGIN
            FOR i:=1 TO iNumFilas DO
            BEGIN
                Write(m[i,j]:6);
            END;
            Writeln;
        END;
    END;
```

2.13 PROCEDURE pRellenarArrayOrdenada(VAR a: targAlumno; VAR iLong: Integer); VAR i,

```
iPos, j: Integer;
    boEnc: Boolean;
    rgA : trgAlumno;
BEGIN
    iLong := 0;
    FOR i:=1 TO MAX DO {Vamos a pedirle al usuario los datos de MAX alumnos} BEGIN
        {Primer paso: pedimos los datos y los guardamos en una variable}
        {WITH rgA DO
        BEGIN
            Write('DNI: ');
            Readln(sDni);
```

```
    Write('EDAD: ');
    Readln(iEdad);
END;}
pPedirAlumno(rgA);
{Segundo paso: buscamos la posición del array en la que tenemos que almacenar el
alumno para que quede ordenado ascendentemente por edad}
iPos := 1 ;
boEnc := FALSE;
WHILE (iPos<=iLong) AND (boEnc=FALSE) DO
BEGIN
    IF a[iPos].iEdad > rgA.iEdad THEN
      boEnc := TRUE
    ELSE
        iPos := iPos + 1;
    END;
    {Tercer paso: Desplazamos a la derecha para dejar un hueco libre en esa posición}
    FOR j:= iLong+1 DOWNTO iPos+1 DO
       a[i] := a[i-1];
    {Cuarto paso: Almacenamos el nuevo elemento}
    a[iPos] := rgA;
    iLong := iLong + 1;
  END;
END;
```

2.14 Procedure con el que pedimos un elemento del array y nos lo borra PROCEDURE pJokalariaKendu(VAR argJokalariak: targJokalariak; VAR iLuz: Integer; iDortsal: Integer);

```
VAR i,iPos: Integer;
boAurki:boolean;
BEGIN
    boAurki := FALSE;
    iPos := 1;
    WHILE (NOT boAurki) AND (iPos<=iLuz) DO
    BEGIN
        IF argJokalariak[iPos].iDortsal = iDortsal THEN
        boAurki := TRUE
        ELSE
        iPos := iPos + 1;
    END;
    IF boAurki THEN
    BEGIN
    FOR i:=iPos TO iLuz - 1 DO
        argJokalariak[i] := argJokalariak[i+1];
    iLuz := iLuz - 1;
    END;
END;
```