GRUPO 4:
BRUNO, LEO JOAQUÍN - FAI 3268
HERRERA, JEREMÍAS EZEQUIEL - FAI 3297
MONSERRAT VIDAL, MARIA ELVIRA - FAI 1829

1. Diseñar un algoritmo en pseudocódigo que:
a. Cargue un arreglo de caracteres.
b. Permita al usuario elegir si lo quiere ver en el orden ingresado o invertido (Modularice apropiadamente)

```
ALGORITMO ejercicio01() RETORNA ∅
(*Este algoritmo muestra un arreglo o un arreglo invertido.*)
        ENTERO longitud, respuesta
        longitud ← verificarLongitudArreglo()
        ENTERO [] almacenCaracter ← CREAR ENTERO[longitud]
        cargarArreglo(almacenCaracter)

        ESCRIBIR(“Desea ver el arreglo en el orden ingresado o invertido?”)
        ESCRIBIR(“Ingrese 1 para el orden ingresado, y cualquier otro número para invertido”)
        LEER (respuesta)
        SI respuesta<>1 ENTONCES
                mostrarArregloInvertido(almacenCaracter)
        SINO
                mostrarArreglo(almacenCaracter)
        FIN SI
FIN ALGORITMO

MODULO verificarLongitudArreglo() RETORNA ENTERO
(*Verifica si el número ingresado por el usuario es un número válido para la longitud de arreglo*)
        ENTERO longitud ← 0
        REPETIR
                ESCRIBIR (“¿Cuántos caracteres desea ingresar?”)
                LEER (longitud)
                SI longitud <= 0 ENTONCES
                        ESCRIBIR (“El número ingresado es incorrecto. Por favor, inténtelo nuevamente”)
                FIN SI
        HASTA (longitud <= 0)
        RETORNA longitud
FIN MODULO

MODULO cargarArreglo (CARACTER [] almacenCaracter) RETORNA ∅
(*Este módulo se encarga de cargar cualquier arreglo unidimensional*)
        ENTERO i, arrayLength
        arrayLength ← LONGITUD(almacenCaracter)-1
        CARACTER caracterTemporal
        PARA i←0, HASTA arrayLength PASO 1 HACER
                ESCRIBIR(“Ingrese el ” + (i + 1) +“° carácter: ”)
                LEER (caracterTemporal)
                almacenCaracter[ i ] ← caracterTemporal
        FIN PARA
FIN MÓDULO

MODULO mostrarArreglo (CARACTER[] almacenCaracter) RETORNA ∅
(*Este modulo muestra por pantalla el arreglo original*)
        ENTERO i, arrayLength
        arrayLength ← LONGITUD(almacenCaracter)-1
        PARA i←0, HASTA arrayLength, PASO 1 HACER
                ESCRIBIR(almacenCaracter[ i ])
        FIN PARA
FIN MODULO

MODULO mostrarArregloInvertido (CARACTER [] almacenCaracter) RETORNA ∅
(*Este módulo muestra por pantalla el arreglo invertido*)
        ENTERO i, arrayLength
        arrayLength ← LONGITUD(almacenCaracter)-1
        PARA i←arrayLength, HASTA i>0, PASO -1 HACER
                ESCRIBIR(almacenCaracter[ i ])
        FIN PARA
FIN MODULO
```

2. Diseñar un algoritmo en pseudocódigo que lea un valor entero (N) y genere un arreglo con los 10 primeros múltiplos del mismo. Por ejemplo para N=7 deberá guardar en el arreglo: 7 14 21 28 35 42 49 56 63 70

```
ALGORITMO ejercicio02 () RETORNA ∅
(*Algoritmo que recibe un número entero y muestra los primeros 10 múltiplos diferentes de cero.*)
        ENTERO escalar
        ENTERO [] arregloMultiplo ← CREAR ENTERO [10]
        ESCRIBIR (“Ingrese el número”)
        LEER (escalar)
        cargarArregloConMultiplo(almacenNumero, escalar)
```

```
        mostrarArreglo(almacenNumero)
FIN ALGORITMO

MODULO cargarArregloConMultiplo (ENTERO [] almacenNumero, ENTERO escalar) RETORNA ∅
(*Este módulo se encarga de cargar los múltiplos de un numero ingresado por el usuario en cualquier arreglo unidimensional*)
        ENTERO i, arrayLength
        arrayLength ← LONGITUD(almacenCaracter)-1
        PARA i←0, HASTA arrayLength, PASO 1 HACER
                almacenNumero[ i ] ← escalar * (i+1)
        FIN PARA
FIN MÓDULO

MODULO mostrarArreglo (ENTERO[] almacenNumero) RETORNA ∅
(*Muestra el arreglo por pantalla*)
        ENTERO i, arrayLength
        arrayLength ← LONGITUD(almacenCaracter)-1
        PARA i←0, HASTA arrayLength, PASO 1 HACER
                ESCRIBIR(almacenCaracter[ i ])
        FIN PARA
FIN MODULO
```

3. Diseñar un algoritmo en pseudocódigo que dado un valor entero N y un arreglo de enteros, reemplace los valores en las posiciones pares del arreglo por el valor N y muestre el arreglo resultante.

```
ALGORITMO ejercicio03 () RETORNA ∅
(*Algoritmo que dado un numero entero reemplaza por dicho valor todos los valores pares de un arreglo.*)
        ENTERO numero, longitudArreglo
        longitudArreglo←verificarLongitudArreglo()
        ENTERO [] almacenNumero ← CREAR ENTERO [longitudArreglo]
        ESCRIBIR(“Ingrese el número con el que va a reeemplazar: ”)
        LEER (numero)
        cargarArreglo(almacenNumero)
        modificarParesArreglo(almacenNumero, numero)
        mostrarArreglos(almacenNumero)
FIN ALGORITMO

MODULO verificarLongitudArreglo() RETORNA ENTERO
(*Verifica si el número ingresado por el usuario es un número válido para la longitud de arreglo*)
        ENTERO longitud ← 0
        REPETIR
                ESCRIBIR (“¿Cuántos numeros desea ingresar?”)
                LEER (longitud)
                SI longitud <= 0 ENTONCES
                        ESCRIBIR (“El número ingresado es incorrecto. Por favor, inténtelo nuevamente”)
                FIN SI
        HASTA (longitud <= 0)
        RETORNA longitud
FIN MODULO

MODULO cargarArreglo (ENTERO [] almacenNumero) RETORNA ∅
(*Este módulo se encarga de cargar cualquier arreglo unidimensional*)
        ENTERO i,numeroTemporal ← 0, arrayLength
        arrayLength ← LONGITUD(almacenCaracter)-1
        PARA i←0, HASTA arrayLength, PASO 1, HACER
                ESCRIBIR(“Ingrese el ” + (i + 1) +“° número: ”)
                LEER (numeroTemporal)
                almacenNumero[ i ] ← numeroTemporal
        FIN PARA
FIN MÓDULO

MODULO modificarParesArreglo (ENTERO[] almacenNumero, ENTERO numero) RETORNA ∅
(*Actualiza con un numero ingresado por parametro los valores en las posiciones pares del arreglo*)
        ENTERO i, arrayLength
        arrayLength ← LONGITUD(almacenCaracter)-1
        PARA i←0, HASTA arrayLength, PASO 1 HACER
                SI (almacenNumero[ i ] MOD 2) = 0 ENTONCES
                        almacenNumero[ i ] ← numero
                FIN SI
        FIN PARA
FIN MODULO

MODULO mostrarArreglo (ENTERO[] almacenNumero) RETORNA ∅
(*Este arreglo muestra por pantalla el arreglo resultante*)
        ENTERO i, arrayLength
        arrayLength ← LONGITUD(almacenCaracter)-1
        PARA i←0, HASTA arrayLength, PASO 1 HACER
                ESCRIBIR(almacenNumero[ i ])
        FIN PARA
FIN MODULO
```

5. Diseñar un algoritmo en pseudocódigo que calcule el promedio de los valores almacenados en un arreglo de números.

```
ALGORITMO ejercicio05() RETORNA ∅
(*Algoritmo que calcula el promedio de los valores almacenados en un arreglo de numeros.*)
        ENTERO longitudArreglo
        longitudArreglo ← verificarLongitudArreglo()
        ENTERO [] almacenNumero ← CREAR ENTERO [longitudArreglo]
        cargarArreglo(almacenNumero)
        ESCRIBIR("El promedio de los valores ingresados es: "+calcularPromedioArreglo(almacenNumero))
FIN ALGORITMO

MODULO verificarLongitudArreglo() RETORNA ENTERO
(*Verifica si el número ingresado por el usuario es un número válido para la longitud de arreglo*)
        ENTERO longitud ← 0
        REPETIR
                ESCRIBIR ("¿Cuántos numeros desea ingresar?")
                LEER (longitud)
                SI longitud <= 0 ENTONCES
                        ESCRIBIR ("El número ingresado es incorrecto. Por favor, inténtelo nuevamente")
                FIN SI
        HASTA (longitud <= 0)
        RETORNA longitud
FIN MODULO

MODULO cargarArreglo (ENTERO [] almacenNumero) RETORNA ∅
(*Este módulo se encarga de cargar cualquier arreglo unidimensional*)
        ENTERO i,numeroTemporal, arrayLength
        arrayLength ← LONGITUD(almacenCaracter)-1
        numeroTemporal←0
        PARA i←0, HASTA arrayLength, PASO 1, HACER
                ESCRIBIR("Ingrese el " + (i + 1) +"° número: ")
                LEER (numeroTemporal)
                almacenNumero[ i ] ← numeroTemporal
        FIN PARA
FIN MÓDULO

MODULO calcularPromedioArreglo (ENTERO [] almacenNumero) RETORNA REAL
(*Calcula el promedio de todos los valores almacenados de un arreglo.*)
        ENTERO i, acum,arrayLength
        REAL promedio
        arrayLength ← LONGITUD(almacenCaracter)-1
        acum←0
                PARA i←0, HASTA arrayLength, PASO 1 HACER
                        acum←acum+almacenNumero[ i ]
                FIN PARA
        promedio ← acum/(arrayLength+1)
        RETORNA promedio
FIN MODULO
```

6. Diseñar un algoritmo en pseudocódigo que permita almacenar letras en un arreglo, cuya dimensión máxima es de 100 posiciones. El algoritmo debe verificar que el caracter leído sea una letra antes de guardarlo en el arreglo. Al finalizar la carga el algoritmo debe mostrar por pantalla la cantidad de letras guardadas.

```
ALGORITMO ejercicio06() RETORNA ∅
(*Verifica que los datos ingresados sean correctos.*)
        CARACTER [] almacenCaracter ← CREAR CARACTER [100]
        ESCRIBIR("La cantidad de letras dentro del arreglo es: "+cargarContarArreglo(almacenCaracter))
FIN ALGORITMO

MODULO cargarContarArreglo (CARACTER [] almacenCaracter) RETORNA ENTERO
(*Carga un arreglo con caracteres de tipo letra y retorna la cantidad de letras ingresadas.*)
        ENTERO i=0, arrayLength
        CARACTER caracterTemporal
        arrayLength ← LONGITUD(almacenCaracter)-1
        REPETIR
                ESCRIBIR("Ingrese un caracter (-1 para dejar de cargar): ")
                LEER (caracterTemporal)
                SI CARACTER(esLetra(caracterTemporal)) ENTONCES
                        almacenCaracter[ i ] ← caracterTemporal
                        i← i+1
                SINO
                        SI caracterTemporal<>-1 ENTONCES
                                ESCRIBIR("Debe ingresar una letra, intente nuevamente.")
                        FIN SI
                FIN SI
        HASTA caracterTemporal<>-1 AND i<99
        RETORNA i+1
FIN MÓDULO
```

7. Diseñar un algoritmo en pseudocódigo que permita:
a. Leer palabras y almacenarlas en un arreglo de string.
b. Generar una cadena con las palabras almacenadas en el arreglo separándolas por un espacio en blanco
c. Generar otra cadena con las palabras almacenadas en el arreglo en orden inverso separándolas por un guión ('-')
d. Mostrar ambas cadenas por pantalla.

```
ALGORITMO ejercicio07() RETORNA ∅
(*Algoritmo que muestra las palabras cargadas en un arreglo y muestra una cadena con palabras almacenadas en un arreglo separadas con espacio o guión.*)
        ENTERO longitudArreglo
        longitudArreglo ← verificarLongitudArreglo()
        TEXTO [] arregloPalabra ← CREAR TEXTO [longitudArreglo]
        cargarArreglo(arregloPalabra)
        ESCRIBIR("El arreglo como cadena con espacios es así: "+generarCadenaEspacios(arregloPalabra))
        ESCRIBIR("El arreglo como cadena invertida y guiones es así: "+generarCadenaInversaConGuion(arregloPalabra))
FIN ALGORITMO

MODULO verificarLongitudArreglo() RETORNA ENTERO
(*Verifica si el número ingresado por el usuario es un número válido para la longitud de arreglo*)
        ENTERO longitud ← 0
        REPETIR
                ESCRIBIR ("¿Cuántas palabras desea ingresar?")
                LEER (longitud)
                SI longitud <= 0 ENTONCES
                        ESCRIBIR ("El número ingresado es incorrecto. Por favor, inténtelo nuevamente")
                FIN SI
        HASTA (longitud <= 0)
        RETORNA longitud
FIN MODULO

MODULO cargarArreglo (TEXTO [] arregloPalabra) RETORNA ∅
(*Este módulo se encarga de cargar el arreglo*)
        ENTERO i,palabraTemporal, arrayLength
        arrayLength ← LONGITUD(almacenCaracter)-1
        PARA i←0, HASTA arrayLength, PASO 1  HACER
                ESCRIBIR("Ingrese la " + (i + 1) +"° palabra: ")
                LEER (palabraTemporal)
                almacenNumero[ i ] ← palabraTemporal
        FIN PARA
FIN MÓDULO

MODULO generarCadenaEspacios (TEXTO [] arregloPalabra) RETORNA TEXTO
(*Este módulo genera el espacio entre las cadena*)
        TEXTO cadenaFinal
        cadenaFinal←""
        ENTERO i, arrayLength
        arrayLength ← LONGITUD(almacenCaracter)-1
        PARA i←0, HASTA arrayLength, PASO 1 HACER
                cadenaFinal←cadenaFinal+" "+arregloPalabra[ i ]
        FIN PARA
        cadenaFinal ← removerEspacios(cadenaFinal)
        RETORNA cadenaFinal
FIN MODULO

MODULO generarCadenaInversaConGuion (TEXTO [] arregloPalabra) RETORNA TEXTO
(*Genera una cadena con las palabras del arreglo de forma invertida usando guión entre medio de ellas.*)
        TEXTO cadenaInvertidaFinal
        cadenaInvertidaFinal←""
        ENTERO i, arrayLength
        arrayLength ← LONGITUD(almacenCaracter)-1
        PARA i←0, HASTA arrayLength, PASO 1 HACER
                cadenaInvertidaFinal←arregloPalabra[ i ]+"-"+cadenaInvertidaFinal
        FIN PARA
(*Usamos el método subcadena para remover el guion que queda al final de la cadena*)
        cadenaInvertidaFinal←subcadena(cadenaInvertidaFinal,0,LONGITUD(arregloPalabra)-1)
        RETORNA cadenaInvertidaFinal
FIN MODULO
```

8. Diseñar un algoritmo en pseudocódigo que busque la palabra más larga almacenada en un arreglo de String (cada posición guarda exactamente 1 palabra)

```
ALGORITMO ejercicio08() RETORNA ∅
(*Muestra la palabra mas larga encontrada en un arreglo.*)
        ENTERO longitudArreglo
        longitudArreglo ←  verificarLongitudArreglo()
        TEXTO [] arregloPalabra ← CREAR TEXTO [longitudArreglo]
        cargarArreglo(arregloPalabra)
        ESCRIBIR("La palabra más larga del arreglo es: "+buscarPalabraLarga(arregloPalabra))
FIN ALGORITMO
```

```
MODULO verificarLongitudArreglo() RETORNA ENTERO
(*Verifica si el número ingresado por el usuario es un número válido para la longitud de arreglo*)
        ENTERO longitud ← 0
        REPETIR
                ESCRIBIR (“¿Cuántas palabras desea ingresar?”)
                LEER (longitud)
                SI longitud <= 0 ENTONCES
                        ESCRIBIR (“El número ingresado es incorrecto. Por favor, inténtelo nuevamente”)
                FIN SI
        HASTA (longitud <= 0)
        RETORNA longitud
FIN MODULO

MODULO cargarArreglo (TEXTO [] arregloPalabra) RETORNA ∅
(*Este módulo se encarga de cargar cualquier arreglo unidimensional*)
        ENTERO i,palabraTemporal, arrayLength
        arrayLength ← LONGITUD(almacenCaracter)-1
        PARA i←0, HASTA arrayLength, PASO 1  HACER
                ESCRIBIR(“Ingrese la ” + (i + 1) +“° palabra: ”)
                LEER (palabraTemporal)
                almacenNumero[ i ] ← palabraTemporal
        FIN PARA
FIN MÓDULO

MODULO buscarPalabraLarga (TEXTO [] arregloPalabra) RETORNA TEXTO
(*Busca la palabra más larga de un arreglo y la retorna*)
        ENTERO i, arrayLength
        TEXTO palabraMasLarga
        palabraMasLarga←” ”
        arrayLength ← LONGITUD(almacenCaracter)-1
        PARA i←0, HASTA arrayLength, PASO 1  HACER
                SI LONGITUD(arregloPalabra[ i ])>LONGITUD(palabraMasLarga) ENTONCES
                        palabraMasLarga←arregloPalabra[ i ]
                FIN SI
        FIN PARA
        RETORNA palabraMasLarga
FIN MODULO
```

9. Diseñar dos módulos en pseudocódigo que dado un arreglo de caracteres y un carácter:
a. Verifique si el carácter ingresado se encuentra en el arreglo. ¿Puede optimizar el algoritmo?
b. Cuente cuántas veces aparece el caracter en el arreglo. ¿Puede optimizar el algoritmo?
Implementar el algoritmo llamador que invoque a los módulos

```
ALGORITMO ejercicio09() RETORNA ∅
(*Muestra por pantalla si un caracter existe en un arreglo y la cantidad de veces que aparece si es que existe.*)
        ENTERO longitudArreglo
        longitudArreglo ← verificarLongitudArreglo()
        ENTERO [] almacenCaracter ← CREAR ENTERO[longitudArreglo]
        cargarArreglo(almacenCaracter)
        ESCRIBIR(“Ingrese el carácter que desea buscar y contar: ”)
        LEER (caracterBuscado)
        SI verificarExistencia(almacenCaracter, caracterBuscado) ENTONCES
                ESCRIBIR(“Si existe el carácter dentro del arreglo.”)
                ESCRIBIR(“La cantidad de veces que aparece el carácter es:”+contarAparicionDeCaracter(almacenCaracter, caracterBuscado))
        SINO
                ESCRIBIR(“No existe el carácter dentro del arreglo.”)
        FIN SI
FIN ALGORITMO

MODULO verificarLongitudArreglo() RETORNA ENTERO
(*Verifica si el número ingresado por el usuario es un número válido para la longitud de arreglo*)
        ENTERO longitud ← 0
        REPETIR
                ESCRIBIR (“¿Cuántos caracteres desea ingresar?”)
                LEER (longitud)
                SI longitud <= 0 ENTONCES
                        ESCRIBIR (“El número ingresado es incorrecto. Por favor, inténtelo nuevamente”)
                FIN SI
        HASTA (longitud <= 0)
        RETORNA longitud
FIN MODULO

MODULO cargarArreglo (CARACTER [] almacenCaracter) RETORNA ∅
(*Este módulo se encarga de cargar cualquier arreglo unidimensional*)
        ENTERO i, arrayLength
        CARACTER caracterTemporal
        arrayLength ← LONGITUD(almacenCaracter)-1
        PARA i←0, HASTA arrayLength PASO 1 HACER
                ESCRIBIR(“Ingrese el ” + (i + 1) +“° carácter: ”)
                LEER (caracterTemporal)
```

```
            almacenCaracter[ i ] ← caracterTemporal
        FIN PARA
FIN MÓDULO

MODULO verificarExistencia (CARACTER[] almacenCaracter, CARACTER caracterBuscado) RETORNA LOGICO
(*Verifica la existencia de un carácter dado.*)
        ENTERO i,j,longitudArreglo
        LOGICO existe
        existe ← FALSO
        longitudArreglo ← LONGITUD(almacenCaracter)-1
        i←0
        j←longitudArreglo
        REPETIR
            SI almacenCaracter[i]=caracterBuscado OR almacenCaracter[j]=caracterBuscado ENTONCES
                existe←VERDADERO
            FIN SI
            j←j-1
            i←i+1
        HASTA existe=FALSO AND j>=i
        SI almacenCaracter[j+1]=caracterBuscado ENTONCES
            existe←VERDADERO
        FIN SI
        RETORNA existe
FIN MODULO

MODULO contarAparicionDeCaracter (CARACTER [] almacenCaracter, CARACTER caracterBuscado) RETORNA ENTERO
(*Cuenta las apariciones de un carácter en un arreglo.*)
        ENTERO i,j,longitudArreglo, contador←0
        longitudArreglo ← LONGITUD(almacenCaracter)-1
        i←0
        j←longitudArreglo
        REPETIR
            SI almacenCaracter[j]=caracterBuscado ENTONCES
                contador←contador+1
            FIN SI
            SI almacenCaracter[i]=caracterBuscado ENTONCES
                contador← contador+1
            FIN SI
            i←i+1
            j←j-1
        HASTA j>i
        i←i-1
        j←j+1
        SI j=i AND almacenCaracter[i]=caracterBuscado ENTONCES
            contador←contador-1
        FIN SI
        RETORNA contador
FIN MODULO
```

10. Diseñar un algoritmo en pseudocódigo que dado un arreglo cargado con valores fijos genere otro arreglo con los valores invertidos.

```
ALGORITMO ejercicio10 () RETORNA ∅
(*Muestra un arreglo con los valores invertidos.*)
        ENTERO[] arregloNumero ← {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
        ESCRIBIR("El arreglo de manera invertida es asi: "+mostrarArregloInvertido(arregloNumero))
FIN ALGORITMO

MODULO mostrarArregloInvertido (ENTERO[] arregloNumero) RETORNA TEXTO
(*Crea una cadena con los valores del arreglo de forma invertida.*)
        ENTERO i
        TEXTO cadenaInvertida
        PARA i←0, HASTA LONGITUD(arregloNumero)-1,PASO 1 HACER
            cadenaInvertida←arregloNumero[i]+" "+cadenaInvertida
        FIN PARA
        cadenaInvertida←removerEspacios(cadenaInvertida)
        RETORNA cadenaInvertida
FIN MODULO
```

11. Diseñar un algoritmo en pseudocódigo que cargue dos arreglos de números y luego verifique si son iguales o no. Para ello se debe implementar un módulo que realice la verificación.

```
ALGORITMO ejercicio11() RETORNA ∅
(*Cargados dos arreglos de números muestra por pantalla si son iguales o no.*)
        ENTERO longitudArreglo

        ESCRIBIR("Primer Arreglo. ")
        longitudArreglo ← verificarLongitudArreglo()
        ENTERO [] arregloNumeroA ← CREAR ENTERO [longitudArreglo]

        ESCRIBIR("Segundo Arreglo. ")
```

```
        longitudArreglo ← verificarLongitudArreglo()
        ENTERO [] arregloNumeroB ← CREAR ENTERO [longitudArreglo]

        ESCRIBIR (“Cargue el primer arreglo: “)
        cargarArreglo(arregloNumeroA)

        ESCRIBIR (“Cargue el primer arreglo: “)
        cargarArreglo(arregloNumeroB)

        SI compararArreglos(arregloNumeroA, arregloNumeroB) ENTONCES
                ESCRIBIR(“El contenido de ambos arreglos es diferente entre sí.”)
        SINO
                ESCRIBIR(“El contenido de ambos arreglos es igual”)
        FIN SI
FIN ALGORITMO

MODULO verificarLongitudArreglo() RETORNA ENTERO
(*Verifica si el número ingresado por el usuario es un número válido para la longitud de arreglo*)
        ENTERO longitud ← 0
        REPETIR
                ESCRIBIR (“¿Cuántos numeros desea ingresar?”)
                LEER (longitud)
                SI longitud <= 0 ENTONCES
                        ESCRIBIR (“El número ingresado es incorrecto. Por favor, inténtelo nuevamente”)
                FIN SI
        HASTA (longitud <= 0)
        RETORNA longitud
FIN MODULO
```

```
MODULO cargarArreglo (ENTERO [] almacenNumero) RETORNA ∅
(*Este módulo se encarga de cargar cualquier arreglo unidimensional*)
        ENTERO i,numeroTemporal
        numeroTemporal←0
        PARA i←0, HASTA LONGITUD(almacenNumero)-1, PASO 1, HACER
                ESCRIBIR(“Ingrese el ” + (i + 1) +“° número: ”)
                LEER (numeroTemporal)
                almacenNumero[ i ] ← numeroTemporal
        FIN PARA
FIN MÓDULO

MODULO compararArreglos (ENTERO[] arregloNumeroA, ENTERO [] arregloNumeroB) RETORNA LOGICO
(*Compara los dos arreglos dados y verifica que sean iguales.*)
        LOGICO sonDiferentes←FALSO
        ENTERO i
        SI LONGITUD(arregloNumeroA) <> LONGITUD(arregloNumeroB) ENTONCES
                sonDiferentes ← VERDADERO
        SINO
                REPETIR
                        SI arregloNumeroA[i]=arregloNumeroB[i] ENTONCES
                                i←i+1
                        SINO
                                sonDiferentes←VERDADERO
                        FIN SI
                HASTA sonDiferentes = FALSO AND i<LONGITUD(arregloNumeroA)
        FIN SI
        RETORNA sonDiferentes
FIN MODULO
```

12. Diseñar un algoritmo en pseudocódigo que cargue un arreglo de caracteres y luego realice la copia de un arreglo en otro de igual tamaño (modularice).

```
ALGORITMO ejercicio12() RETORNA ∅
(*Carga un arreglo de caracteres y lo copia en otro.*)
        ENTERO longitudArreglo
        longitudArreglo ← verificarLongitudArreglo()
        CARACTER [] arregloCaracterOriginal ← CREAR CARACTER [longitudArreglo]
        CARACTER [] arregloCaracterCopia ← CREAR CARACTER [longitudArreglo]
        cargarArreglo(arregloCaracterOriginal)
        copiarContenidoArray(arregloCaracterOriginal, arregloCaracterCopia)
FIN ALGORITMO

MODULO verificarLongitudArreglo() RETORNA ENTERO
(*Verifica si el número ingresado por el usuario es un número válido para la longitud de arreglo*)
        ENTERO longitud ← 0
        REPETIR
                ESCRIBIR (“¿Cuántos caracteres desea ingresar?”)
                LEER (longitud)
                SI longitud <= 0 ENTONCES
                        ESCRIBIR (“El número ingresado es incorrecto. Por favor, inténtelo nuevamente”)
                FIN SI
```

```
        HASTA (longitud <= 0)
        RETORNA longitud
FIN MODULO

MODULO cargarArreglo (CARACTER [] arregloCaracter) RETORNA ∅
(*Este módulo se encarga de cargar cualquier arreglo unidimensional*)
        ENTERO i
        CARACTER caracterTemporal
        longitudArray←LONGITUD(arregloCaracter)-1
        PARA i←o, HASTA longitudArray, PASO 1 HACER
                ESCRIBIR(“Ingrese el ” + (i + 1) +“° caracter: ”)
                LEER (caracterTemporal)
                almacenCaracter[ i ] ← caracterTemporal
        FIN PARA
FIN MÓDULO

MODULO copiarContenidoArray (CARACTER[] charArrayA, CARACTER[] charArrayB) RETORNA ∅
(*Este módulo copia el contenido de dos arreglos, (siguen siendo diferentes porque no apuntan a la misma direc de memoria)*)
        ENTERO i, longitudArray
        longitudArray←LONGITUD(charArrayA)-1
        PARA i←0, HASTA longitudArray, PASO 1 HACER
                charArrayB[i] ← charArrayA[i]
        FIN PARA
FIN MODULO
```

13. Diseñar un algoritmo en pseudocódigo que cargue un arreglo de caracteres y luego genere otro que contenga solo las vocales que se encuentran en el arreglo original.

```
ALGORITMO ejercicio13() RETORNA ∅
(*Cargado un arreglo de caracteres, genera un nuevo arreglo con vocales.*)
        ENTERO longitudArreglo ← 0
        longitudArreglo ← verificarLongitudArreglo()
        CARACTER [] almacenCaracter ←CREAR CARACTER [longitudArreglo]
        cargarArreglo(almacenCaracter)
        SI cuentaVocales(almacenCaracter) == 0 ENTONCES
                ESCRIBIR (“El arreglo no tiene vocales.”)
        SINO
                ESCRIBIR("El nuevo arreglo de vocales es: \n ")
                leerArreglo(generarArregloVocales(almacenCaracter))
        FIN SI


        generarArregloVocales (CARACTER[] almacenCaracter)

FIN ALGORITMO

MODULO verificarLongitudArreglo() RETORNA ENTERO
(*Verifica si el número ingresado por el usuario es un número válido para la longitud de arreglo*)
        ENTERO longitud ← 0
        REPETIR
                ESCRIBIR (“¿Cuántos caracteres desea ingresar?”)
                LEER (longitud)
                SI longitud <= 0 ENTONCES
                        ESCRIBIR (“El número ingresado es incorrecto. Por favor, inténtelo nuevamente”)
                FIN SI
        HASTA (longitud <= 0)
        RETORNA longitud
FIN MODULO

MODULO cargarArreglo (CARACTER[] almacenCaracter) RETORNA ∅
(*Este módulo se encarga de cargar valores de tipo caracter a un arreglo.*)
        ENTERO i, longitudAlmacen
        CARACTER caracter← “”
        longitudAlmacen ← LONGITUD(almacenCaracter)-1,
        PARA i←0, HASTA longitudAlmacen PASO 1, HACER
                ESCRIBIR(“Ingrese el ” + (i + 1) +“° carácter: ”)
                LEER (caracter)
                almacenCaracter[ i ] ← caracter
        FIN PARA
FIN MÓDULO

MODULO cuentaVocales (CARACTER[] almacenCaracter) RETORNA ENTERO
(*Cuenta cuántas vocales hay en un arreglo. *)
        ENTERO i, contador ← 0, longitudAlmacen
        longitudAlmacen ← LONGITUD (almacenCaracter)

        PARA i← 0 HASTA longitudAlmacen-1 PASO 1 HACER
                SI (almacenCaracter[i] == ‘a’|| almacenCaracter[i] == ‘e’ || almacenCaracter[i] == ‘i’
                || almacenCaracter[i] == ‘o’ || almacenCaracter[i] == ‘u’ )ENTONCES
                        contador ←contador +1
```

```
            FIN SI
        FIN PARA
        RETORNA contador;
FIN MODULO


MODULO generarArregloVocales (CARACTER[] almacenCaracter) RETORNA CARACTER[]
(*Este módulo se encarga de crear un nuevo arreglo con las vocales del arreglo principal. *)
        ENTERO i, longitudAlmacenCaracter, longitudAlmacenVocales, contadorDeLongitud ←0
        longitudAlmacenCaracter ← LONGITUD(almacenCaracter)
        longitudAlmacenVocales ← cuentaVocales (almacenCaracter)
        CARACTER [] almacenVocales ←CREAR CARACTER [longitudAlmacenVocales]

        PARA i←0, HASTA longitudAlmacenCaracter PASO 1, HACER
                SI (almacenCaracter[i] == 'a'|| almacenCaracter[i] == 'e' || almacenCaracter[i] == 'i'
                || almacenCaracter[i] == 'o' || almacenCaracter[i] == 'u') ENTONCES
                        almacenVocales [contadorDeLongitud]←almacenCaracter[i]
                        contadorDeLongitud ← contadorDeLongitud +1
                FIN SI
        FIN PARA
        RETORNA almacenVocales
FIN MÓDULO

MODULO leerArreglo(CARACTER[] arreglo) RETORNA ∅
(*Muestra por pantalla los caracteres de un arreglo*)
        ENTERO i, longitudArreglo
        longitudArreglo← LONGITUD(arreglo)
        ESCRIBIR ("Las vocales del arreglo original son: ")
        PARA i←0 HASTA longitudArreglo-1 PASO 1 HACER
                ESCRIBIR (arreglo[i])
        FIN PARA
FIN MODULO
```

14. Diseñar un algoritmo en pseudocódigo que cargue un arreglo de String y luego genere dos nuevos arreglos, uno conteniendo las cadenas que estaban en las posiciones pares y otro conteniendo los caracteres que estaban en las posiciones impares. Modularice.

```
ALGORITMO ejercicio14() RETORNA ∅
(*Este algoritmo genera 2 nuevos arreglos mostrando los caracteres que hay en la posiciones par e impares de un arreglo padre.*)

        ENTERO longitudArreglo ← 0, longitudPar, longitudImpar
        longitudArreglo ← verificarLongitudArreglo()
        longitudImpar ←numerosImpar(longitudArreglo)
        longitudPar←(longitudArreglo-longitudImpare)

        TEXTO [] almacenTexto←CREAR TEXTO [longitudArreglo]
        cargarTexto(almacenTexto)
        TEXTO [] almacenImpares←CREAR TEXTO[longitudImpar]
        cargarImpares(almacenTexto)
        TEXTO [] almacenPares←CREAR TEXTO[longitudPar]
        cargarPares(almacenTexto)

        mostrarArreglo(almacenTexto,almacenImpares, almacenPares)
FIN ALGORITMO
MODULO verificarLongitudArreglo() RETORNA ENTERO
(*Verifica si el número ingresado por el usuario es un número válido para la longitud de arreglo*)
        ENTERO longitud ← 0
        REPETIR
                ESCRIBIR ("¿Cuántos números desea ingresar?")
                LEER (longitud)
                SI longitud <= 0 ENTONCES
                        ESCRIBIR ("El número ingresado es incorrecto. Por favor, inténtelo nuevamente")
                FIN SI
        HASTA (longitud <= 0)
        RETORNA longitud
FIN MODULO

MODULO cargarTexto(TEXTO [] almacenTexto) RETORNA ∅
(*Este módulo se encarga de cargar valores arreglo.*)
        ENTERO i, logitudTexto
        STRING textoTemporal← " "
        longitudTexto←LONGITUD(almacenTexto)
        PARA i←0, HASTA longitudTexto-1, PASO 1, HACER
                ESCRIBIR("Ingrese el " + (i + 1) +"° texto: ")
                LEER (textoTemporal)
                almacenTexto[ i ] ← textoTemporal
        FIN PARA
FIN MÓDULO

MODULO numeroImpar (ENTERO longitudArreglo) RETORNA ENTERO
(*Este módulo se encarga de saber cuántos números pares tiene un numero padre*)
        ENTERO i, numeroPar←0
```

```
        PARA i←0 HASTA longitudArreglo, PASO 1 HACER
                numeroPar←longitudArreglo DIV 2
        FIN PARA
RETORNA numeroPar
FIN MODULO

MODULO cargarImpares (TEXTO [] almacenTexto, TEXTO [] almacenImpares) RETORNA ∅
(*Este modulo se encargar de cargar el arreglo impar*)
        ENTERO i, longitudTexto
        longitudTexto←LONGITUD(almacenImpares)
                PARA i←0 HASTA longitudTexto, PASO 1 HACER
                almacenImpares[i]=almacenTexto[(i*2)+1]
        FIN PARA
FIN MODULO

MODULO cargarPares (TEXTO [] almacenTexto, TEXTO [] almacenPar) RETORNA ∅
(*Este modulo se encargar de cargar el arreglo par*)
        ENTERO i, longitudTexto
        longitudTexto←LONGITUD(almacenPar)
                PARA i←0 HASTA longitudTexto, PASO 1 HACER
                almacenPares[i]=almacenTexto[(i*2)]
        FIN PARA
FIN MODULO

MODULO mostrarArreglo(TEXTO [] almacenTexto, TEXTO [] almacenPares, TEXTO [] almacenImpares) RETORNA ∅
(*Este modulo se encargar de mostrar por pantalla los 3 arreglos*)
        ENTERO i, longitudTexto1, longitudTexto2, longitudTexto3
        longitudTexto1←LONGITUD(almacenTexto)
        longitudTexto2←LONGITUD(almacenImpares)
        longitudTexto3←LONGITUD(almacenPares)

        ESCRIBIR(“Los datos del arreglo padre son:”)
                PARA i←0 HASTA longitudTexto1, PASO 1 HACER
                        ESCRIBIR(almacenTexto[i]+ “ ”)
                FIN PARA
        ESCRIBIR(“\nLos datos del las posiciones impares son:”)
                PARA  i←0 HASTA longitudTexto2, PASO 1 HACER
                        ESCRIBIR(almacenImpares[i]+ “ ”)
                FIN PARA
        ESCRIBIR(“\nLos datos del las posiciones pares son:”)
                PARA  i←0 HASTA longitudTexto3, PASO 1 HACER
                        ESCRIBIR(almacenPares[i]+ “ ”)
                FIN PARA
FIN MODULO
```

15. Problema Número de DNI. El documento de identidad (DNI) en España, consta de 8 cifras y de una letra. La letra del DNI se obtiene siguiendo los pasos a continuación: 1) Calcula el resto de dividir el número del DNI entre 23 2) El número obtenido estará entre 0 y 22, selecciona la letra asociada al valor obtenido utilizando la siguiente tabla:

0	1	2	3	4	5	6	7	8	9	10	11
T	R	W	A	G	M	Y	F	P	D	X	B

12	13	14	15	16	17	18	19	20	21	22	
N	J	Z	S	Q	V	H	L	C	K	E	

Por ejemplo, si el número del DNI es 31415927 y el resto de dividir por 23 es 20, la letra que le corresponde según la tabla es la “C” Diseñar un algoritmo que solicite un numero de 8 cifras y devuelva el número de DNI correspondiente.

```
ALGORITMO ejercicio15() RETORNA ∅
(*Este algoritmo muestra por pantalla el documento español con su letra correspondiente.*)
        ENTERO numeroDocumento
        CARACTER letraDocumento
        numeroDocumento ← verificarIngreso()
        letraDocumento ← asignaLetra(numeroDocumento)
        ESCRIBIR (“El número de documento completo es: ”+ numeroDocumento + “ “ +
                        letraDocumento)
FIN ALGORITMO

MODULO verificaIngreso() RETORNA ENTERO
(*Verifica si el número de documento ingresado es correcto. Retorna el número entero ingresado.*)
        ENTERO numeroDocumento
        REPETIR
                ESCRIBIR (“Ingrese el número de documento: ”)
                LEER (numeroDocumento)
                SI (numeroDocumento < 10 000 000 OR numeroDocumento >= 100 000 000)ENTONCES
                        ESCRIBIR (“El número ingresado es incorrecto. Por favor, inténtelo nuevamente”)
                FIN SI
        HASTA (numeroDocumento < 10 000 000 OR numeroDocumento >= 100 000 000)
        RETORNA numeroDocumento
```

```
FIN MODULO

MODULO asignaLetra(ENTERO documento) RETORNA CARACTER
(*Dado un número de documento, calcula el resto de la división por 23 del documento y retorna la letra correspondiente.*)
        ENTERO resto
        CARACTER letraAsignada ←""
        CARACTER[] almacenLetras ← { 'T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B', 'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E'}
        resto ← documento MOD 23
        letraAsignada ← almacenLetras[resto]
        RETORNA letraAsignada
FIN MODULO
```

16. Dado un arreglo que almacena cadenas de caracteres se desea verificar que las mismas cumplan con las siguientes condiciones: tengan una longitud mínima de 5 caracteres y que contenga solo letras. En caso de que la cadena no cumpla la condición debe ser eliminada del arreglo y la cadena que está en la siguiente posición debe ocupar su lugar. Imprima por pantalla el arreglo resultante.

```
ALGORITMO ejercicio16() RETORNA ∅
(*Verifica que las cadenas de texto de un arreglo cumplan con ciertas condiciones. De no cumplirlas se reemplaza con espacio.*)
        ENTERO longitudArreglo ← 0
        longitudArreglo ← verificarLongitudArreglo()
        TEXTO [] almacenTexto←CREAR TEXTO [longitudArreglo]
        TEXTO [] almacenTexto2←CREAR TEXTO [longitudArreglo]
        cargarTexto(almacenTexto)
        verificarArreglo(almacenTexto, almacenTexto2)
        modificarArreglo(almacenTexto2)
        mostrarArreglo(almacenTexto2)
FIN ALGORITMO

MODULO verificarLongitudArreglo() RETORNA ENTERO
(*Verifica si el número ingresado por el usuario es un número válido para la longitud de arreglo*)
        ENTERO longitud ← 0
        REPETIR
                ESCRIBIR ("¿Cuántos números desea ingresar?")
                LEER (longitud)
                SI longitud <= 0 ENTONCES
                        ESCRIBIR ("El número ingresado es incorrecto. Por favor, inténtelo nuevamente")
                FIN SI
        HASTA (longitud <= 0)
        RETORNA longitud
FIN MODULO

MODULO cargarTexto(TEXTO [] almacenTexto) RETORNA ∅
(*Este módulo se encarga de cargar valores arreglo.*)
        ENTERO i, logitudTexto
        STRING textoTemporal← " "
        longitudTexto←LONGITUD(almacenTexto)-1
        PARA i←0, HASTA longitudTexto-1, PASO 1, HACER
                ESCRIBIR("Ingrese el " + (i + 1) +"° texto: ")
                LEER (textoTemporal)
        FIN PARA
FIN MÓDULO

MODULO identificarLetras(TEXTO cadena) RETORNA LOGICO
(*Este modulo indentifica si la palabra esta formada por letras*)
        ENTERO i←0 , largoCadena
        LOGICO esLetra←VERDADERO
        largoCadena←LONGITUD(cadena)
        REPETIR
                SI !Character.isLetter(cadena.charAt(i) ENTONCES
                        esLetra←FALSO
                FIN SI
                i++
        HASTA esLetra AND i<largoCadena
        RETORNA esLetra
FIN MODULO

MODULO verificarArreglo(TEXTO [] almacenTexto, TEXTO [] almacenTexto2) RETORNA ∅
(*Verifica si las palabras ingresadas tienen 5 caracteres de solo letras.*)
        ENTERO i, logitudTexto, j=0;
        LOGICO letra
        longitudTexto←LONGITUD(almacenTexto)-1
        PARA i←0, HASTA longitudTexto-1, PASO 1, HACER
                letra←identificarLetras(almacenTexto[i])
                SI LONGITUD(almacenTexto[i]) = 5 AND letra=VERDADERO ENTONCES
                        almacenTexto2[j] ← almacenTexto[i]
                        j←j+1
                FIN SI
        FIN PARA
FIN MÓDULO
```

```
MODULO modificarArreglo(TEXTO [] almacenTexto2) RETORNA ∅
(*Hace que las celdas que están en null queden con un espacio.*)
        ENTERO i, logitudTexto
        longitudTexto←LONGITUD(almacenTexto2)-1
        PARA i←0, HASTA longitudTexto-1, PASO 1, HACER
                SI almacenTexto2[i] = null ENTONCES
                        almacenTexto2[i] ← “ ”
                FIN SI
        FIN PARA
FIN MÓDULO

MODULO mostrarArreglo(TEXTO [] almacenTexto2) RETORNA ∅
(*Este módulo se encarga de mostrar por pantalla el arreglo.*)
        ENTERO i, logitudTexto
        longitudTexto←LONGITUD(almacenTexto2)-1
        ESCRIBIR(“\n”)
        ESCRIBIR("Los datos del arreglo son")
        PARA i←0, HASTA longitudTexto-1, PASO 1, HACER
                ESCRIBIR(almacenTexto2[i]+ “ ”)
        FIN PARA
                ESCRIBIR(“\n”)
FIN MÓDULO
```

18. Implementar un algoritmo que utilice dos arreglos, uno que almacena nombres de personas empleadas en una empresa y otro que almacena los sueldos de las mismas, y sabiendo que ambos arreglos se corresponden por posición, presentar un menú de opciones para realizar algunas de las siguientes acciones: a. Buscar la persona que tiene mayor sueldo, mostrar su nombre y el sueldo. b. Listar todas las personas que cobran exactamente un valor X (leítdo por teclado). c. Aumentar en un 10% los sueldos que sean inferiores a $10000. d. Buscar una persona y si se encuentra mostrar su sueldo

```
ALGORITMO ejercicio18() RETORNA ∅
(*Asumiendo existen dos arreglos existentes de empleados y sueldos realiza diferentes acciones sobre ellos.*)
        CARACTER opcionUsuario
        ENTERO valorSueldo
        TEXTO nombreCompletoEmpleado
        TEXTO[] empleados ← {"Leo Bruno", "Jeremias Herrera", "Maria Monserrat", "Axl Vidman" , “Mikaela Raid”}
        REAL[] sueldos ← {12000, 10000, 10000, 9000, 10}
        opcionUsuario ← menu()

        SEGUN (opcionUsuario) HACER
                CASO ‘a’:
                        ESCRIBIR ("El mayor sueldo es de $" + sueldos[sueldoMayor(sueldos)] + " y le pertenece a " +
                        empleados[sueldoMayor(sueldos)] + ". ")
                CASO ‘b’:
                        ESCRIBIR("Ingrese un valor (sueldo) que desee buscar: ")
                        LEER (valorSueldo)
                        sueldosIguales(sueldos, empleados, valorSueldo)
                CASO ‘c’:
                        aumentarSueldos(sueldos, empleados)
                CASO ‘d’:
                        ESCRIBIR ("Ingrese el nombre y apellido del empleado: ")
                        LEER (nombreCompletoEmpleado)
                        buscaUsuario(sueldos, empleados, nombreCompletoEmpleado)
                PREDETERMINADO
                        ESCRIBIR (“Programa cerrado.”)
        FIN SEGUN
FIN ALGORITMO
```

```
MODULO menu() RETORNA ∅
(*Imprime en pantalla un menú de opciones.*)
        ENTERO opcion

        RETORNA opcion
FIN MODULO

MODULO sueldoMayor(REAL[] sueldo) RETORNA ENTERO
(*Busca en el arreglo de sueldos el mayor sueldo y devuelve la ubicación en el arreglo.*)
        ENTERO i, posicion ← 0, longitudArreglo
        REAL sueldoMayor ← 0
        longitudArreglo ← LONGITUD(sueldo)
        PARA i ← 0 HASTA longitudArreglo-1 PASO 1 HACER
                SI (sueldo[i] > sueldoMayor) ENTONCES
                        sueldoMayor ← sueldo[i]
                        posicion ← i
                FIN SI
        FIN PARA
        RETORNA posicion
FIN MODULO

MODULO sueldosIguales(REAL[] sueldos, TEXTO[] empleados, REAL valorSueldo) RETORNA ∅
(*Dado un valor por parametro muestra por pantalla los empleados que tienen el mismo sueldo.*)
```

```
        ENTERO i, longitudArreglo
        LOGICO noHaySueldoIgual ← VERDADERO
        longitudArreglo ← LONGITUD(sueldos)
        PARA I← 0 HASTA longitudArreglo-1 PASO 1 HACER
                SI (sueldos[i] = valorSueldo) ENTONCES
                        ESCRIBIR ("Empleado/a: " + empleados[i] + ". \n")
                        noHaySueldoIgual ← FALSO
                FIN SI
        FIN PARA

        SI (noHaySueldoIgual) ENTONCES
                ESCRIBIR ("No hay empleados con el sueldo igual al ingresado. ")
        FIN SI
FIN MODULO

MODULO aumentarSueldos(REAL[] sueldos, TEXTO[] empleados) RETORNA ∅
(*Aumenta automaticamente los sueldos inferiores a 10.000 en un 10%. Informa a quienes se les aumentó y a cuanto.*)
        ENTERO i, longitudArreglo
        LOGICO sueldoSuperiores ← VERDADERO
        longitudArreglo ← LONGITUD(sueldos)
        PARA i ← 0 HASTA longitudArreglo -1 PASO 1 HACER
                SI (sueldos[i] < 10000) ENTONCES
                        sueldos[i] = sueldos[i] + ((sueldos[i] * 10) / 100)
                        ESCRIBIR ("Se aumentó el sueldo de " + empleados[i] + " a $" + sueldos[i] + ".\n")
                        sueldoSuperiores ← FALSO
                FIN SI
        FIN PARA
        SI (sueldoSuperiores) ENTONCES
                ESCRIBIR("Ningún empleado tiene un sueldo inferior a $10000.");
        FIN SI
FIN MODULO

MODULO buscaUsuario(REAL[] sueldos, TEXTO[] empleados, TEXTO empleadoNombre) RETORNA ∅
(*Dado un nombre por parámetro busca el empleado y muestra nombre y sueldo.*)
        ENTERO i ← 0, longitudArreglo
        LOGICO empleadoNoEncontrado ← VERDADERO
        longitudArreglo ← LONGITUD(sueldos)
        HACER
                SI(empleadoNombre.equalsIgnoreCase(empleados[i])) ENTONCES
                        ESCRIBIR("El empleado " + empleados[i] + " tiene un sueldo de $" + sueldos[i] + ". ")
                        empleadoNoEncontrado ← FALSO
                FIN SI
                i← i+1
        MIENTRAS (empleadoNoEncontrado AND i < longitudArreglo)
        SI (empleadoNoEncontrado) ENTONCES
                ESCRIBIR ("El nombre y apellido ingresado no pertenece a ningún empleado cargado.")
        FIN SI
FIN MODULO
```

EJERCICIOS PARA COMPARTIR CON LA CLASE.
SE PUEDE MODIFICAR HACIENDO UN SOLO MODULO QUE RETORNE UN ARREGLO NUEVO CON LOS 2 VALORES (MAYOR Y MENOR)

4. Diseñar un algoritmo en pseudocódigo que permita encontrar el valor más grande y el más pequeño almacenado en un arreglo de números.

```
ALGORITMO numerosMayorMenor() RETORNA ∅
(*Este algoritmo muestra el número más pequeño y más grande de un arreglo de números enteros.*)
        ENTERO longitudArreglo ← 0
        longitudArreglo ← verificarLongitudArreglo()
        ENTERO [] almacenNumero ←CREAR ENTERO [longitudArreglo]
        cargarArreglo (almacenNumero)
        ESCRIBIR("El número más grande es: "+numeroMayor(almacenNumero))
        ESCRIBIR("El número más pequeño es: "+numeroMenor(almacenNumero))
FIN ALGORITMO

MODULO verificarLongitudArreglo() RETORNA ENTERO
(*Verifica si el número ingresado por el usuario es un número válido para la longitud de arreglo*)
        ENTERO longitud ← 0
        REPETIR
                ESCRIBIR ("¿Cuántos números desea ingresar?")
                LEER (longitud)
                SI longitud <= 0 ENTONCES
                        ESCRIBIR ("El número ingresado es incorrecto. Por favor, inténtelo nuevamente.")
                FIN SI
        HASTA (longitud <= 0)
        RETORNA longitud
FIN MODULO

MODULO cargarArreglo (ENTERO [] almacenNumero) RETORNA ∅
(*Este módulo se encarga de cargar valores numéricos enteros a un arreglo.*)
        ENTERO i, numeroTemporal ← 0
        PARA i←0, HASTA LONGITUD(almacenNumero)-1, PASO 1, HACER
                ESCRIBIR("Ingrese el " + (i + 1) +"° número: ")
                LEER (numeroTemporal)
                almacenNumero[ i ] ← numeroTemporal
        FIN PARA
FIN MÓDULO

MODULO numeroMayor (ENTERO [] almacenNumero) RETORNA ENTERO
(*Dado un arreglo por parámetro, busca en el arreglo el número más chico del arreglo*)
        ENTERO i, numeroGrande← -10000000
        PARA i←0, HASTA LONGITUD(almacenNumero)-1 PASO 1 HACER
                SI almacenNumero [ i ] > numeroGrande ENTONCES
                        numeroGrande←almacenNumero [ i ]
                FIN SI
        FIN PARA
        RETORNA numeroGrande
FIN MODULO

MODULO numeroMenor (ENTERO [] almacenNumero) RETORNA ENTERO
(*Dado un arreglo por parámetro, busca en el arreglo el número más chico del arreglo*)
        ENTERO i, numeroChico ← 10000000
        PARA i←0 HASTA LONGITUD(almacenNumero) -1 PASO 1 HACER
                SI almacenNumero [ i ] < numeroChico ENTONCES
                        numeroChico ← almacenNumero [ i ]
                FIN SI
        FIN PARA
        RETORNA numeroChico
FIN MODULO
```

TRAZA numeroMayorMenor

longitudArreglo	SALIDA
~~0~~	
2	
	El número más grande es: 6
	El número más pequeño es: 4

TRAZA verificarLongitudArreglo

longitud	SALIDA	RETORNA
~~0~~	¿Cuántos números desea ingresar?	
2		
		2

ARREGLO almacenNumero

4	6

TRAZA cargarArreglo

i	numeroTemporal	SALIDA
	0	
~~0~~		Ingrese el 1° número:
	4	
~~1~~		Ingrese el 2° número:
	6	
~~2~~		

TRAZA numeroMayor

i	numeroGrande	RETORNA
	-10000000	
~~0~~		
	4	
~~1~~		
	6	
~~2~~		
		6

TRAZA numeroMenor

i	numeroChico	RETORNA
	10000000	
~~0~~		
	4	
~~1~~		
	6	
~~2~~		
		4

17. Dado un arreglo que almacena las notas correspondientes a un alumno, las cuales son números reales, se desea verificar si el alumno aprobó el cuatrimestre. La condición para aprobar es tener todas las notas con valores mayores o iguales a 6. Se debe implementar un algoritmo que cargue el arreglo con 10 notas y verifique si el alumno aprobó o no el cuatrimestre.

```
ALGORITMO notasAlumno() RETORNA ∅
(*Este algoritmo verifica si el alumno aprobó el cuatrimestre*)
        REAL [] arregloDeNotas←CREAR REAL [10]
        cargarNotas(arregloDeNotas)
        SI evaluarNotas (arregloDeNotas) ENTONCES
                ESCRIBIR(“El alumno aprobó el cuatrimestre.”)
        SINO
                ESCRIBIR(“El alumno reprobó el cuatrimestre.”)
        FIN SI
FIN ALGORITMO
```

(*Se podrían mejorar los módulos usando la función LONGITUD(arregloDeNotas) en lugar de i<10 para poder utilizarlo en caso de modificar el arreglo inicial a más o menos de 10 notas. Decidimos dejar el i<10 por interpretación del problema.*)

```
MODULO cargarNotas (REAL [] arregloDeNotas ) RETORNA ∅
(*Este módulo se encarga de cargar las notas de un alumno a un arreglo.*)
        ENTERO i ← 0
        REAL nota ← 0
        REPETIR
                ESCRIBIR(“Ingrese la ” + (i + 1) +“° nota de la materia: ”)
                LEER (nota)
                SI nota>0 AND nota<11 ENTONCES
                        arregloDeNotas[ i ] ← nota
                        i←i+1
                SINO
                        ESCRIBIR(“Nota invalida, ingréselo nuevamente.”)
                FIN SI
        HASTA i<10
FIN MÓDULO

MODULO evaluarNotas (REAL [] arregloDeNotas) RETORNA LOGICO
(*Evalúa si un estudiante aprobó devolviento Verdadero o Falso. Criterio 6 o más.*)
        LOGICO aprobado ← VERDADERO
        ENTERO i ← 0
        REPETIR
                SI arregloDeNotas[ i ] < 6 ENTONCES
                        aprobado←FALSO
                FIN SI
                i←i+1
        HASTA aprobado AND i<>10
        RETORNA aprobado
FIN MODULO
```

MODULO ALTERNATIVO: corta la carga de notas al encontrar una nota menor a 6. De ingresarse un numero invalido solicita el reingreso.

```
MODULO cargarNotas (REAL [] arregloDeNotas ) RETORNA ∅
(*Este módulo se encarga de cargar las notas de un alumno a un arreglo, cuando cualquiera de las notas ingresadas sea menor a 6, el algoritmo corta la carga y deja de pedir notas.*)
        ENTERO i ← 0
        REAL nota ← 0
        LOGICO bochado ← FALSO
        REPETIR
                ESCRIBIR(“Ingrese la ” + (i + 1) +“° nota de la materia: ”)
                LEER (nota)
                SI nota>0 AND nota<11 ENTONCES
                        SI nota>6 ENTONCES
                                arregloDeNotas[ i ] ← nota
                                i←i+1
                        ELSE
                                ESCRIBIR (“El alumno desaprobó, ya no se cargaran más notas.”)
                                bochado ← VERDADERO
                        FIN SI
                SINO
                        ESCRIBIR(“Nota invalida, ingréselo nuevamente.”)
                FIN SI
        HASTA i!=10 AND bochado = FALSO
FIN MÓDULO
```

JAVA EJERCICIO 4.

```
import java.util.Scanner;

public class numerosMayorMenor {

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        // Este algoritmo muestra el número más pequeño y más grande de un arreglo de números enteros.
        int longitudArreglo;
        longitudArreglo = verificarLongitudArreglo();
        int[] almacenNumero = new int[longitudArreglo];
        cargarArreglo(almacenNumero);
        System.out.println("El número más grande es:" + numeroMayor(almacenNumero));
        System.out.println("El número más chico es:" + numeroMenor(almacenNumero));
    }

    public static int verificarLongitudArreglo(){
        //Verifica si el número ingresado por el usuario es un número válido para la longitud de arreglo
        Scanner sc = new Scanner(System.in);
        int longitud;
        do {
            System.out.println("Cuántos números desea ingresar?");
            longitud = sc.nextInt();
            if (longitud <= 0) {
                System.out.println("El número ingresado es incorrecto. Por favor, inténtelo nuevamente.");
            }
        } while (longitud <= 0);
        return longitud;
    }

    public static void cargarArreglo(int[] almacenNumero) {
        //Este módulo se encarga de cargar valores numéricos enteros a un arreglo.
        int i, numeroTemporal;
        Scanner sc = new Scanner(System.in);
        for (i = 0; i < almacenNumero.length; i++) {
            System.out.println("Ingrese el " + (i + 1) + "° número: ");
            numeroTemporal = sc.nextInt();
            almacenNumero[i] = numeroTemporal;
        }
    }

    public static int numeroMayor(int[] almacenNumero) {
        //Dado un arreglo por parámetro, busca en el arreglo el número más chico del arreglo
        int i, numeroGrande = -1000000;
        for (i = 0; i < almacenNumero.length; i++) {
            if (almacenNumero[i] > numeroGrande) {
                numeroGrande = almacenNumero[i];
            }
        }
        return numeroGrande;
    }

    public static int numeroMenor(int[] almacenNumero) {
        //Dado un arreglo por parámetro, busca en el arreglo el número más chico del arreglo
        int i, numeroChico = 10000000;
        for (i = 0; i < almacenNumero.length; i++) {
            if (almacenNumero[i] < numeroChico) {
                numeroChico = almacenNumero[i];
            }
        }
        return numeroChico;
    }
}
```

JAVA EJERCICIO 17.

```
import java.util.Scanner;
public class diesisiete {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        //Este algoritmo verifica si el alumno aprobó el cuatrimestre
        Scanner sc = new Scanner(System.in);
        double[] arregloDeNotas = new double[10];
        cargarNotas(arregloDeNotas);
        if (evaluarNotas(arregloDeNotas)) {
            System.out.println("El alumno aprobó el cuatrimestre.");
        } else {
            System.out.println("El alumno reprobó el cuatrimestre.");
        }

    }

    public static void cargarNotas(double[] arregloDeNotas) {
        //Este módulo se encarga de cargar las notas de un alumno a un arreglo.
        Scanner sc = new Scanner(System.in);
        double nota = 0;
        int i = 0;
        do {
            System.out.println("Ingrese la " + (i + 1) + "° nota de la materia: ");
            nota = sc.nextDouble();
            if (nota > 0 && nota < 11) {
                arregloDeNotas[i] = nota;
                i++;
            } else {
                System.out.println("Nota invalida, ingréselo nuevamente.");
            }
        } while (i < 10);
    }

    public static boolean evaluarNotas(double[] arregloDeNotas) {
        //Evalúa si un estudiante aprobó devolviento Verdadero o Falso. Criterio 6 o más.
        boolean aprobado = true;
        int i = 0;
        do {
            if (arregloDeNotas[i] < 6) {
                aprobado = false;
            }
            i++;
        } while (aprobado && i != 10);
        return aprobado;
    }


    /*
    //Modulo alternativo de CargarNotas.
    public static void cargarNotas(double[] arregloDeNotas) {
        //Este módulo se encarga de cargar las notas de un alumno a un arreglo.
        Scanner sc = new Scanner(System.in);
        double nota = 0;
        int i = 0;
        boolean bochado = false;
        do {
            System.out.println("Ingrese la " + (i + 1) + "° nota");
            nota = sc.nextDouble();
            if (nota > 0 && nota < 11) {
                if (nota > 6) {
                    arregloDeNotas[i] = nota;
                    i++;
                } else {
                    System.out.println("Bochado");
                    bochado = true;
                }
            } else {
                System.out.println("Nota invalida, ingreselo nuevamente.");
            }
        } while (i < 10 && bochado == false);
    }*/
}
```

MaxIntValue