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

Para los siguientes ejercicios: Diseñar los algoritmos, realizar traza e implementar en Java

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

```
ALGORITMO problema1() RETORNA ∅
(*Este algoritmo permite calcular los primeros 10 múltiplos de un número*)
        ENTERO i,numUsuario,resultado;
                ESCRIBIR (“Ingrese un número”)
                LEER numUsuario

                (*puede tener un SI que verifique si el número es 0. De ser 0 dar un mensaje de que los primeros 10 múltiplos de 0 son 0*)
                PARA i ← 1 HASTA 10 PASO 1 HACER
                        resultado ←numUsuario*i
                        ESCRIBIR (resultado + “ “)
                FIN PARA
FIN ALGORITMO
```

2. Diseñar un algoritmo en pseudocódigo que dado un valor entero de 3 cifras (debe verificarlo) descomponga el número para mostrarlo a la inversa utilizando las operaciones MOD y DIV.
Por ejemplo si N=1234 debe mostrar 4321

```
ALGORITMO invertirNumero RETORNA ∅
(*Dado un número entero se muestra por pantalla invertido.*)
        ENTERO numeroInvetir
        REPETIR
                ESCRIBIR (“Ingrese un número entero de tres dígitos:”)
                LEER numeroInvertir
        HASTA (numeroAInvertir < 100 OR numeroInvertir >= 1000)

        ESCRIBIR (“ El numero invertido es” invierteNumero(numeroInvertir) )

FIN PROGRAMA

MODULO invierteNumero (ENTERO numero) RETORNA TEXTO
(*recibido un numero entero por parametro, realiza los calculos necesarios para invertir el orden de los números*)
        ENTERO cifra
        TEXTO numeroInvertido
        MIENTRAS numero > 0
                cifra ← numero MOD 10
                numeroInvertido ← numeroInvertido + cifra
                numero ← numero DIV 10
        FIN MIENTRAS
        RETORNA numeroInvertido
FIN MODULO
```

3. Diseñar en pseudocódigo un algoritmo que permita leer números positivos hasta ingresar un valor negativo, y en base a ellos muestre:
a) Cantidad de números pares
b) Mayor número leído

```
ALGORITMO numeros RETORNA ∅
(*Dado muchos numeros X nos dice los pares y el mayor*)
        REAL numeroIngresado, numeroMasGrande
        TEXTO almacenDeNumerosPares

        REPETIR
                ESCRIBIR “Ingrese un número:”
                LEER numeroIngresado
                SI numeroIngresado > 0 ENTONCES
                        almacenDeNumerosPares ← esPar(numeroIngresado) + “ “

                        SI (numeroIngresado > numeroMasGrande) ENTONCES
                                numeroMasGrande ← numeroIngresado
                        FIN SI
                FIN SI
        HASTA numeroIngresado < 0

        ESCRIBIR (“Los números pares son:” almacenDeNumerosPares “ y el mayor número ingresado
        fue “ numeroMasGrande)
FIN ALGORITMO

MODULO esPar (REAL numero) RETORNA TEXTO
(*Si el numero es Par devuelve el numero par, si no devuelve vacio*)
        TEXTO numeroPar ←“ ”

                SI (numero MOD 2 == 0) ENTONCES
                        numeroPar ← numero
                FIN SI
        RETORNA numeroPar
FIN MODULO
```

4. Diseñar en pseudocódigo un algoritmo que permita ingresar una frase por teclado, y en base a ella busque y muestre la palabra más larga. Suponga que entre las palabras hay un único espacio en blanco.
¿Cómo debería modificar el algoritmo para considerar que entre las palabras puede haber más de un espacio en blanco?

```
ALGORITMO palabraMasLarga RETORNA ∅
(*Dada una frase por el usuario, muestra por pantalla la palabra más larga de la frase*)
        TEXTO frase
        ESCRIBIR (“Ingrese una frase: ”)
        LEER frase
        ESCRIBIR (“La palabra más larga es” buscaPalabraMasLarga(frase))
FIN ALGORITMO

MODULO buscaPalabraMasLarga (TEXTO fraseNueva) RETORNA TEXTO
(*Dada una frase nueva busca la palabra mas larga*)
        TEXTO palabrasMasLarga ← “”, palabra ← “”
        ENTERO i
        fraseNueva ← fraseNueva + “ ”
        PARA i ← 0 HASTA i< logitud(fraseNueva) PASO 1 HACER
                SI posicion(fraseNueva,i) != “ “
                        palabra ← palabra + posicion(fraseNueva,i)
                SINO
                        SI (longitud(palabra) > longitud(palabraMasLarga)) ENTONCES
                                palabraMasLarga ← palabra
                        FIN SI
                        palabra ← “”
                FIN SI
        FIN PARA
        RETORNA palabraMasLarga
FIN MODULO
```

5. Diseñar un módulo en pseudocódigo que dada una frase y un caracter, verifique si el caracter ingresado se encuentra en la frase.
a. Implemente solo usando únicamente las operaciones longitud (length) y posicion (charAt)
b. Implemente usando las operaciones de String más adecuadas
Implementar el algoritmo llamador que invoque al módulo.

```
ALGORITMO texto RETORNA ∅
(*Permite verificar si el caracter se encuentra en la oración*)
      TEXTO oración
      CARACTER caracter
      LOGICO buscador

      (*Solicitud de datos*)
      ESCRIBIR ("Ingrese una oración")
      LEER oracion
      ESCRIBIR ("Ingrese un caracter a buscar en la oracion")
      LEER caracter

      (*Invocacion del modulo*)
      buscador ← caracterBuscado(oracion, caracter)

      (*Verificación*)
      SI buscador = verdadero ENTONCES
         ESCRIBIR ("El caracter que busca si esta en su oracion.")
      SINO
         ESCRIBIR ("El caracter que busca no esta en su oracion.")
   FIN SI

   (*Modulo que encuentra el caracter en una oracion*)
   MODULO caracterBuscado(TEXTO texto, CARACTER caracter) RETORNA LOGICO
      ENTERO i
      LOGICO letra ← falso

      PARA i ← 0 HASTA i< longitud(texto) PASO 1 HACER
         SI posicion(texto, i) = caracter ENTONCES
            letra ← verdadero
         FIN SI
      FIN PARA
      RETORNA  letra
   FIN MÓDULO
FIN ALGORITMO
```

6. Diseñar un módulo en pseudocódigo que dada una frase y una palabra, cuente cuántas veces aparece la palabra en la frase.
Piense una implementación usando solo las operaciones longitud (length) y posición (charAt) y otra usando todas las operaciones de String (las que usted crea convenientes).
Implemente la mejor solución y realice traza.

NO EFICIENTE

```
ALGORITMO cuentaAparicionDePalabra RETORNA ∅
(*Dada una frase y una palabra a buscar, indica la cantidad de veces que aparece la palabra en la frase*)
        TEXTO frase, palabraBuscar
        ESCRIBIR (“Ingrese una frase: ”)
        LEER frase
        ESCRIBIR (“Ingrese la palabra a buscar: ”)
        LEER palabraBuscar
        ESCRIBIR (“La palabra aparece ” contarPalabra (frase, palabra)“ veces.” )
FIN ALGORITMO

MODULO contarPalabra (TEXTO frase, TEXTO palabraBuscar) RETORNO ENTERO
(*Dada una frase por parametro y una palabra, cuenta cuantas veces aparece y devuelve la cantidad de veces que aparece en la frase*)
        ENTERO longitudPalabra, resultado ← 0,i,n,contador ← 0,longitudFrase;
        TEXTO palabra ← "";

        longitudPalabra ← longitud(palabraBuscar)
        frase ← frase+" "
        longitudFrase ← largo(frase)
        PARA n ← 0 HASTA n < longitud(frase) PASO 1 HACER
                SI (posicion(frase, n) != ‘ ’) ENTONCES
                        palabra ← palabra + posicion(frase, n)
                SINO
                        SI (longitud(palabraBuscar) = longitud(palabra)) ENTONCES
                                PARA i←0 HASTA i<longitudPalabra PASO 1 HACER
                                        SI(posicion(palabraBuscar, i) = posicion(palabra.i)) ENTONCES
                                                contador← contador +1
                                        FIN SI
                                        SI (contador==longitudPalabra) {
                                                i← longitudPalabra-1
                                                resultado← resultado +1
                                        FIN SI
                                FIN PARA
                                contador ← 0
                                palabra ← ""
                        SINO
                                palabra ← "";
                                contador ← 0;
```

```
                    FIN SI
                FIN SI
        FIN PARA
        RETORNA resultado
FIN MODULO
```

EFICIENTE

```
ALGORITMO cuentaAparicionDePalabra RETORNA ∅
(*Dada una frase y una palabra a buscar, indica la cantidad de veces que aparece la palabra en la frase*)
        TEXTO frase, palabraBuscar
        ESCRIBIR (“Ingrese una frase: ”)
        LEER frase
        ESCRIBIR (“Ingrese la palabra a buscar: ”)
        LEER palabraBuscar
        ESCRIBIR (“La palabra aparece ” cuentaAparicionPalabra (frase, palabra) “ veces.” )
FIN ALGORITMO

MODULO contarPalabra (TEXTO frase, TEXTO palabraBuscar) RETORNO
(*Dada una frase por parametro y una palabra, cuenta cuantas veces aparece y devuelve la cantidad de veces que aparece en la frase*)
        TEXTO palabra
        ENTERO i, contador ← 0

        frase ← frase+" "
        PARA i ← 0 HASTA i < longitud(frase) PASO 1 HACER
                SI (posicion(frase, i) != ‘ ’) ENTONCES
                        palabra ← palabra + posicion(frase, i)
                SINO
                        SI (igual(palabra, palabraBuscar)) ENTONCES
                                contador← contador +1
                        FIN SI
                        palabra ← “”
                FIN SI
        FIN PARA
        RETORNA contador
FIN MODULO
```

7. Diseñar en pseudocódigo un algoritmo que permita ingresar una cierta cantidad de palabras (N) y en base a ellas muestre:
a) Si existen palabras capicúas
b) La cantidad de palabras con más de dos vocales
c) La palabra de mayor longitud.

```
ALGORITMO analisisDePalabras RETORNA ∅
(*Analiza de una frase ingresada si existen palabras capicuas, cantidad de palabtas con mas de dos vocales y la palabra con mayor longitud.*)
```

```
TEXTO frase
ESCRIBIR("Ingresa la frase: ")
LEER frase

ESCRIBIR("Hay palabras capicúa: "  palabrasCapicua(frase))
ESCRIBIR("La cantidad de palabras con más de dos vocales es: "  cuentaPalabrasVocales(frase))
ESCRIBIR("La palabra más larga es: "  palabraMasLarga(frase))

FIN ALGORITMO

MODULO palabrasCapicua(TEXTO textoUsuario) RETORNA LOGICO
(*Analiza si existe al menos una palabra capicúa en una cadena de texto*)
        ENTERO i,j
        TEXTO palabra, cadenaInvertida
        LOGICO esCapicua ← falso
        textoUsuario ← textoUsuario + " "

        PARA i ← 0 HASTA longitud(textoUsuario) PASO 1 HACER
                SI posicion (textoUsuario, i) != " " ENTONCES
                        palabra ← palabra + posicion(textoUsuario, i)
                SINO
                        palabra ← aMinuscula(palabra)
                        PARA j← 0 HASTA j < longitud(palabra) PASO 1 HACER
                                cadenaInvertida ← posicion(palabra, j) + cadenaInvertida
                        FIN PARA
                        SI (igual(palabra, cadenaInvertida)) ENTONCES
                                esCapicua ← verdadero
                        FIN SI
                        cadenaInvertida ← " "
                        palabra ← " "
        FIN PARA
        RETORNA esCapicua
FIN MODULO

MODULO cuentaPalabrasVocales (TEXTO textoUsuario) RETORNA ENTERO
(*cuenta cuantas palabras tienen mas de 2 vocales en una cadena de texto*)
        TEXTO palabra = ""
        ENTERO i, contador, palabrasVocales
        textoUsuario ← aMinuscula(textoUsuario)
        PARA i← 0 HASTA i< longitud(textoUsuario) PASO 1 HACER
                SI posicion(textoUsuario, i) != ' ' ENTONCES
                        palabra = palabra + posicion(textoUsuario, i)
                SINO
                        PARA j ← 0 HASTA j< largo(palabra) PASO 1 HACER
                                SEGUN (posicion(palabra, j)) HACER
                                        caso 'a': contador+1
                                        caso 'e': contador+1
                                        caso 'i': contador+1
```

```
                                caso 'o': contador+1
                                caso 'u': contador+1
                                default : contador ← contador +0
                        FIN SEGUN
                FIN PARA
                SI (contador > 2) ENTONCES
                        palabrasVocales ← palabrasVocales +1
                FIN SI
                contador ← 0
                palabra ← ""
        FIN SI
    FIN PARA
    RETORNA palabrasVocales;
FIN MODULO

MODULO palabraMasLarga (TEXTO fraseNueva) RETORNA TEXTO
(*Dada una frase nueva busca la palabra mas larga*)
TEXTO palabrasMasLarga, palabra
ENTERO i


PARA i ← 0 HASTA i< longitud(fraseNueva) PASO 1 HACER
        SI (posicion(fraseNueva, i) != " ")
                palabra ← palabra + posicion(fraseNueva,i)
        SINO
                palabra ← ""
        FIN SI

        SI (longitud(palabra) > longitud(palabraMasLarga)) ENTONCES
                palabraMasLarga ← palabra
        FIN SI
FIN PARA
RETORNA palabraMasLarga
FIN MODULO
```

8. Para encriptar un mensaje se cambian las vocales por los siguientes símbolos: *, /, +, - y # (correspondientes a la 'a', 'e', 'I', 'o' y 'u' respectivamente).
Diseñe un módulo que lea un mensaje (una cadena) y genere un mensaje encriptado.
Diseñe el algoritmo que luego de invocar al módulo muestre por pantalla mensaje

final.

```
ALGORITMO encriptaMensaje RETORNA ∅
(*Dada una frase, cambia las vocales por simbolos y muestra por pantalla la frase modificada*)
TEXTO frase
        ESCRIBIR (“Ingrese una frase: ”)
        LEER frase
        ESCRIBIR (“La frase encriptada es: ” mensajeEncriptado(frase) )
FIN ALGORITMO

MODULO mensajeEncriptado (TEXTO frase) RETORNA TEXTO
        TEXTO nuevaFrase
        ENTERO i
        PARA i=0 HASTA i < longitud(frase) PASO 1 HACER
                nuevaFrase ← SEGUN (posicion (frase, i)) HACER
                                        caso a: nuevaFrase + '*'
                                        caso A: nuevaFrase + '*'
                                        caso e: nuevaFrase + '/';
                                        caso E: nuevaFrase + '/';
                                        caso i: nuevaFrase + '+';
                                        caso I: nuevaFrase + '+';
                                        caso o: nuevaFrase + '-';
                                        caso O: nuevaFrase + '-';
                                        caso u: nuevaFrase + '#';
                                        caso U: nuevaFrase + '#';
                                        SINO nuevaFrase + posicion (frase, i);
                                FIN SEGUN
        FIN PARA
        RETORNA fraseNueva
FIN MODULO
```

9. En algunos sistemas informáticos solo se aceptan contraseñas que cumplan con ciertos requerimientos a fin de mejorar el nivel de seguridad de las mismas. Diseñe un algoritmo que dada una contraseña verifique si cumple con las siguientes condiciones:

a. Tiene exactamente 8 caracteres

b. Tiene al menos 1 letra

c. Tiene al menos 1 número

d. Tiene al menos 1 caracter: '*' '/' '-' '$' '%' '#'

```
ALGORITMO contraseña RETORNA ∅
(*Este algoritmo nos permite saber si nuestra contraseña es segura*)
*Declaracion de variables*
TEXTO contraseña
LOGICO contraseñaSegura
*Solicitud de datos*
REPETIR
        REPETIR
                ESCRIBIR (“Ingrese una contraseña: \nNota: Debe contener exactamente 8 caracteres\n
                Al menos 1 numero\n Al menos 1 letra\n Al menos 1 caracter de tipo: '*' '/' '-' '$' '%' '#'”)
                LEER contraseña
        HASTA contreseña.isEmpty()

        *Llamo al modulo*
        contraseñaSegura ← checker(contraseña)

        Si contraseñaSegura ← verdadero ENTONCES
                ESCRIBIR (“Su contraseña es segura.”)
        SINO
                ESCRIBIR (“Su contraseña no es segura. \Intentelo de nuevo”
        FIN SI
HASTA contraseñaSegura = falso
FIN ALGORITMO

(*Modulo que identifica si la contraseña cumple con las condiciones*)
MODULO checker (TEXTO cadena) RETORNA LOGICO
LOGICO safePass, numero = falso, letra = falso, caracter =falso
ENTERO i

(*Primera condicion: Si son 8 caracteres exactos*)
        SI longitud(cadena) = 8 ENTONCES
                (Recorro la cadena para buscar las demás condiciones)
                PARA i ← 0 HASTA i< longitud(cadena) PASO 1 HACER

                        SI CaracterEsLetra(posicion(cadena)) ENTONCES
                        letra ← verdadero

                        SINO SI CaracterEsDigito(posicion(cadena)) ENTONCES
                        numero ← verdadero

                        SINO SI (posicion(cadena) = ‘*’) OR (posicion(cadena) = ‘/’) OR (posicion(cadena)
                        = ‘-’) OR (posicion(cadena) = ‘$’) OR (posicion(cadena) = ‘%’) OR
                        (posicion(cadena) = ‘#’) ENTONCES
                        caracter = verdadero
                        FIN SI
```

```
            FIN PARA
        FIN SI
        safePass = letra && numero && caracter;
RETORNA safePass
FIN MODULO
```