Universidad Nacional de San Antonio
Abad del Cusco

Escuela Profesional de Física

Problemas desarrollados de Programación Científica

Fred Anthony Vigoria Hualla

Código: 231820

24 de julio de 2024

Nota

El código de los problemas correspondientes así como este documento y demás relacionados se encuentra disponibles en el siguiente repositorio de GitHub: https://github.com/FredVigoria/Tarea-de-programaci-n-cientifica.git

Ejercicio 1

Ejercicio: Se tiene un arreglo A de N elementos y un arreglo B de M elementos. Escribir un módulo que determine si el arreglo A está incluido en el arreglo B.
Estado: Resuelto

Ejercicio 2

Ejercicio: Se tiene un arreglo X de N elementos y un arreglo Y de M elementos. Escribir un módulo que determine cual de los arreglos está incluido en el otro o si no hay inclusión.
Estado: Resuelto

Ejercicio 3

Ejercicio: Dado un arreglo A de N elementos, escribir un módulo que determine si el arreglo está o no ordenado ascendentemente.
Estado: Resuelto

Ejercicio 4

Ejercicio: Escribir un subprograma que sume dos polinomios.
Estado: Resuelto

Ejercicio 5

Ejercicio: Escribir un subprograma que multiplique dos polinomios.
Estado: Resuelto

Ejercicio 6

Ejercicio: Se tiene un arreglo de N notas enteras. Escribir un módulo que determine qué nota es la que más se repite.
Estado: Resuelto

Ejercicio 7

Ejercicio: Dado un arreglo ordenado A de N elementos enteros, escribir un módulo que permita insertar un número entero en el arreglo, en la posición que corresponda.
Estado: Resuelto

Ejercicio 8

Ejercicio: Dado un arreglo A de N elementos enteros, escribir un módulo que permita eliminar un elemento X del arreglo, siempre que X esté en el arreglo.
Estado: Resuelto

Ejercicio 9

Ejercicio: Escribir un módulo que permita multiplicar un escalar por un vector de N elementos.
Estado: Resuelto

Ejercicio 10

Ejercicio: Escribir un módulo que efectúe el producto escalar de dos vectores de N elementos.
Estado: Resuelto

Ejercicio 11

Ejercicio: Sea A un arreglo de N elementos enteros. Escribir un módulo que determine cuántas veces se repite el elemento X en el arreglo.
Estado: Resuelto

Ejercicio 12

Ejercicio: Sea A un arreglo ordenado ascendentemente de N elementos. Escribir un módulo que genere un arreglo B ordenado descendentemente con los elementos del arreglo A.
Estado: Resuelto

Ejercicio 13

Ejercicio: Los artículos de un almacén están codificados con números secuenciales del 1 hasta N. En un arreglo se tiene el número de unidades vendidas de cada artículo, en otro arreglo se tiene el precio unitario de cada artículo. Escribir un módulo que determine el precio total de venta de cada artículo y el gran total.
Estado: Resuelto

Ejercicio 14

Ejercicio: Se tienen cuatro arreglos paralelos, en los tres primeros se tienen las notas correspondientes a los tres exámenes parciales y en el cuarto arreglo se tiene la nota correspondiente al examen sustitutorio. Escribir un módulo que genere un quinto arreglo con los promedios finales, considerando que el examen sustitutorio reemplaza al examen parcial más bajo, siempre y cuando sea mayor a éste.
Estado: Resuelto

Ejercicio 15

Ejercicio: Sea A un arreglo de N elementos enteros. Escribir un módulo que elimine los elementos duplicados del arreglo.
Estado: Resuelto

Ejercicio 16

Ejercicio: Escribir un módulo que determine el valor numérico de un polinomio para un valor dado de X.
Estado: Resuelto

Índice

1. Problema

1.1. Enunciado

Se tiene un arreglo A de N elementos y un arreglo B de M elementos. Escribir un módulo que determine si el arreglo A está incluido en el arreglo B.

1.2. Interpretación

Queremos determinar si todos los elementos del conjunto A están presentes en el conjunto B, sin importar el orden. Esto se conoce como verificar si A es un subconjunto de B.

1.3. Solución

1.3.1. Entender

La tarea es verificar si cada elemento de A se puede encontrar en B. Si al menos uno de los elementos de A no está presente en B, A no está incluido en B.

1.3.2. Modelar

Para modelar este problema, podemos definir una función de búsqueda que verifique si un elemento está presente en un conjunto. Luego, usaremos esta función para verificar cada elemento de A en B. Los pasos son los siguientes:

- Definir una función `Buscar` que verifique si un elemento x está en el conjunto C.
- Definir una función `EstaIncluido` que use `Buscar` para verificar cada elemento de A en B.

1.3.3. Codificar

```
# Determinar si un conjunto esta incluido en otro
# Importamos nuestra libreria
import MiLibreria

# Definimos nuestra funci+on
def EstaIncluido(A, B):
    i = 0
    while i < len(A):
        if MiLibreria.Buscar(A[i], B) ==-1:
            return False
        i += 1
    return True

# ------------- Ejemplo de uso ------------------
# Leer los datos
N = MiLibreria.LeerEntero("Insertar N: ",0,9999)
A = MiLibreria.LeerLista("Insertar los elementos del conjunto A: ",
    N)
M = MiLibreria.LeerEntero("Insertar M: ",0,9999)
B = MiLibreria.LeerLista("Insertar los elementos del conjunto B: ",
    M)

# Mostrar los resultados
print("El conjunto A: ", A)
print("El conjunto B: ", B)
print("A esta incluido en B: ",EstaIncluido(A, B))
```

2. Problema

2.1. Enunciado

Se tiene un arreglo X de N elementos y un arreglo Y de M elementos. Escribir un módulo que determine cuál de los arreglos está incluido en el otro o si no hay inclusión.

2.2. Interpretación

Queremos determinar si el conjunto X está incluido en el conjunto Y, si el conjunto Y está incluido en el conjunto X o si no hay inclusión entre los conjuntos.

2.3. Solución

2.3.1. Entender

La tarea es verificar si cada elemento de X se puede encontrar en Y y viceversa. Esto se puede lograr utilizando una función de búsqueda para verificar la inclusión.

2.3.2. Modelar

Para modelar este problema, usaremos la función `Buscar` para verificar si un elemento está presente en un conjunto y definiremos una función `EstaIncluido` para verificar la inclusión entre dos conjuntos. Los pasos son los siguientes:

- Definir una función `Buscar` que verifique si un elemento x está en el conjunto C.
- Definir una función `EstaIncluido` que use `Buscar` para verificar si todos los elementos de A están en B.
- Definir una función `Inclusion` que determine cuál de los arreglos está incluido en el otro o si no hay inclusión.

2.3.3. Codificar

```
# Determinar la relacion de inclusion entre dos conjuntos
# Importamos las librerias propias
import MiLibreria

# Definimos nuestra funcion
def Inclusion(X, Y):
    if MiLibreria.EstaIncluido(X, Y) and MiLibreria.EstaIncluido(Y,
     X):
        return "Ambos arreglos son iguales"
    elif MiLibreria.EstaIncluido(X, Y):
        return "X esta incluido en Y"
    elif MiLibreria.EstaIncluido(Y, X):
        return "Y esta incluido en X"
    else:
        return "No hay inclusion"

# ------------- Ejemplo de uso ------------------
# Leer los datos
N = MiLibreria.LeerEntero("Insertar N: ",0,9999)
X = MiLibreria.LeerLista("Insertar X: ", N)
M = MiLibreria.LeerEntero("Insertar M: ",0,9999)
Y = MiLibreria.LeerLista("Insertar Y: ", M)

# Mostrar los resultados
print("Conjunto X: ", X)
print("Conjunto Y: ", Y)
print(Inclusion(X, Y))
```

3. Problema

3.1. Enunciado

Dado un arreglo A de N elementos, escribir un módulo que determine si el arreglo está o no ordenado ascendentemente.

3.2. Interpretación

Queremos determinar si el arreglo A está ordenado en orden ascendente, es decir, si cada elemento en A es menor o igual al siguiente elemento.

3.3. Solución

3.3.1. Entender

La tarea es verificar si todos los elementos del arreglo A cumplen la condición $A[i] \leq A[i+1]$ para $0 \leq i < N-1$. Si se encuentra algún par de elementos que no cumpla esta condición, el arreglo no está ordenado ascendentemente.

3.3.2. Modelar

Para modelar este problema, podemos definir un ciclo que recorra el arreglo y compare cada par de elementos consecutivos. Los pasos son los siguientes:

- Inicializar un índice i a 0.
- Recorrer el arreglo A comparando $A[i]$ con $A[i+1]$.
- Si se encuentra un par $A[i] > A[i+1]$, el arreglo no está ordenado ascendentemente.
- Si se recorre todo el arreglo sin encontrar un par que no cumpla la condición, el arreglo está ordenado ascendentemente.

3.3.3. Codificar

```
# Verificar si un conjunto esta ordenado ascendentemente
# Importamos nuestra libreria
import MiLibreria

# Definimos nuestra funcion
def EstaOrdenadoAscendente(A):
    i = 0
    N = len(A)
    while i < N - 1:
        if A[i] > A[i + 1]:
            return False
        i += 1
    return True

# ------------- Ejemplo de uso ------------------
# Leer los datos
N = MiLibreria.LeerEntero("Insertar N: ",1,9999)
A = MiLibreria.LeerLista("Insertar A: ", N)

# Mostrar resultados
print("El conjunto A: ", A)
print("El conjunto A se encuentra ordenado ascendentemente: ",
    EstaOrdenadoAscendente(A))
```

4. Problema

4.1. Enunciado

Escribir un subprograma que sume dos polinomios.

4.2. Interpretación

Queremos escribir un subprograma que tome dos polinomios representados como listas de coeficientes y devuelva su suma, también representada como una lista de coeficientes. Los coeficientes estarán ordenados de mayor a menor grado.

4.3. Solución

4.3.1. Entender

Dado dos polinomios $P(x)$ y $Q(x)$ representados como listas de coeficientes $P = [a_0, a_1, \ldots, a_n]$ y $Q = [b_0, b_1, \ldots, b_m]$, donde a_i y b_i son los coeficientes de los términos de mayor a menor grado, necesitamos sumar estos polinomios.

4.3.2. Modelar

Para sumar dos polinomios, podemos seguir los siguientes pasos:

- Determinar la longitud de los polinomios P y Q.
- Crear un nuevo polinomio S con coeficientes inicializados a 0 de la longitud máxima entre P y Q.
- Completar con ceros a la lista más pequeña para que tengan la misma longitud.
- Recorrer ambos polinomios y sumar los coeficientes correspondientes.

4.3.3. Codificar

```
# Sumar dos polinomios
# Importamos nuestra libreria
import MiLibreria

# Definimos la funcion de sumar polinomios
def SumarPolinomios(P, Q):

    # Longitud del polinomio resultante
    max_len = max(len(P),len(Q))

    # Inicializar el polinomio suma con ceros
    S = [0] * max_len

    # Completar con ceros para que tengan la misma longitud
    while len(P) < max_len:
        P = [0] + P
    while len(Q) < max_len:
        Q = [0] + Q

    # Sumar termino a termino
    i = 0
    while i < max_len:
        S[i] = P[i] + Q[i]
        i += 1
    # Devolver el polinomio suma
    return S

# ------------- Ejemplo de uso ------------------
# Leer los datos
N = MiLibreria.LeerEntero("Inserte el grado del primer polinomio: "
    ,1,99999)
P = MiLibreria.LeerLista("Inserte los coeficientes del primer
    polinomio: ",N+1)
M = MiLibreria.LeerEntero("Inserte el grado del segundo polinomio:
    ",1,99999)
Q = MiLibreria.LeerLista("Inserte los coeficientes del segundo
    polinomio: ",M+1)

# Mostrar los resultados
MiLibreria.ImprimirPolinomio(P,"P")
MiLibreria.ImprimirPolinomio(Q,"Q")
MiLibreria.ImprimirPolinomio(SumarPolinomios(P, Q),"(P+Q)")
```

5. Problema

5.1. Enunciado

Escribir un subprograma que multiplique dos polinomios.

5.2. Interpretación

Queremos escribir un subprograma que tome dos polinomios representados como listas de coeficientes y devuelva su producto, también representado como una lista de coeficientes. Los coeficientes estarán ordenados de mayor a menor grado.

5.3. Solución

5.3.1. Entender

Dado dos polinomios $P(x)$ y $Q(x)$ representados como listas de coeficientes $P = [a_0, a_1, \ldots, a_n]$ y $Q = [b_0, b_1, \ldots, b_m]$, donde a_i y b_i son los coeficientes de los términos de mayor a menor grado, necesitamos multiplicar estos polinomios.

5.3.2. Modelar

Para multiplicar dos polinomios, podemos seguir los siguientes pasos:

- Determinar la longitud de los polinomios P y Q.
- Crear un nuevo polinomio R con coeficientes inicializados a 0 de longitud $N + M - 1$, donde N y M son las longitudes de los polinomios P y Q, respectivamente.
- Recorrer ambos polinomios y multiplicar los coeficientes correspondientes, acumulando los resultados en el polinomio R.

5.3.3. Codificar

```
# Programa que multilica dos polinomios
# Importamos nuestra libreria
import MiLibreria

# Definimos nuestra funcion
def MultiplicarPolinomios(P, Q):
    # Longitud de los polinomios
    n = len(P)
    m = len(Q)

    # Longitud del polinomio resultante
    R = [0] * (n + m - 1)

    # Multiplicar termino a termino
    i = 0
    while i < n:
        j = 0
        while j < m:
            R[i + j] += P[i] * Q[j]
            j += 1
        i += 1

    return R

# ------------- Ejemplo de uso ------------------
# Leer los datos
N = MiLibreria.LeerEntero("Inserte el grado del primer polinomio: "
    ,1,99999)
P = MiLibreria.LeerLista("Inserte los coeficientes del primer
    polinomio: ",N+1)
M = MiLibreria.LeerEntero("Inserte el grado del segundo polinomio:
    ",1,99999)
Q = MiLibreria.LeerLista("Inserte los coeficientes del segundo
    polinomio: ",M+1)

# Mostrar resultados
MiLibreria.ImprimirPolinomio(P,"P")
MiLibreria.ImprimirPolinomio(Q,"Q")
MiLibreria.ImprimirPolinomio(MultiplicarPolinomios(P, Q),"(P*Q)")
```

6. Problema

6.1. Enunciado

Se tiene un arreglo de N notas enteras. Escribir un módulo que determine qué nota es la que más se repite.

6.2. Interpretación

Queremos encontrar la nota que aparece con mayor frecuencia en un arreglo de enteros. En caso de empate, se debe retornar la primera nota que alcance la máxima frecuencia.

6.3. Solución

6.3.1. Entender

Necesitamos contar la frecuencia de cada nota en la lista sin usar diccionarios. Utilizaremos dos listas: una para almacenar las notas únicas y otra para contar sus frecuencias.

6.3.2. Modelar

a. Inicialización de Listas:

- 'notas_unicas' para almacenar las notas encontradas.
- 'frecuencias' para llevar el conteo de las apariciones de cada nota.

b. Contar Frecuencias:

- Iterar sobre la lista de notas.
- Para cada nota, verificar si ya está en 'notas_unicas'. Si está, actualizar su frecuencia; si no, agregarla a 'notas_unicas' y añadir un conteo inicial en 'frecuencias'.

c. Encontrar la Nota con la Mayor Frecuencia:
Iterar sobre 'frecuencias' para encontrar la nota con el máximo valor.

6.3.3. Codificar

```
# Econtrar la notas mas frecuente de una lista de notas
# Importar nuestra libreria
import MiLibreria

# Definimos nuestra funcion
def NotaMasFrecuente(notas):
    notas_unicas = []
    frecuencias = []
    i = 0
    while i < len(notas):
        nota = notas[i]
        # Usar la funcion Buscar para verificar si la nota ya esta en notas_unicas
        indice = MiLibreria.Buscar(nota, notas_unicas)
        if indice != -1:
            frecuencias[indice] += 1
        else:
            notas_unicas.append(nota)
            frecuencias.append(1)
        i += 1
    # Encontrar la nota con la mayor frecuencia
    max_freq = -1
    nota_mas_frecuente = None
    k = 0
    while k < len(frecuencias):
        if frecuencias[k] > max_freq:
            max_freq = frecuencias[k]
            nota_mas_frecuente = notas_unicas[k]
        k += 1

    return nota_mas_frecuente

# ------------- Ejemplo de uso ------------------
# Leer los datos
N = MiLibreria.LeerEntero("Inserte el numero de notas: ", 1, 99999)
notas = MiLibreria.LeerLista("Inserte las notas: ", N)

# Mostrar los resultados
print(f"La nota que mas se repite es: {NotaMasFrecuente(notas)}")
```

7. Problema

7.1. Enunciado

Dado un arreglo ordenado A de N elementos enteros, escribir un módulo que permita insertar un número entero en el arreglo, en la posición que corresponda.

7.2. Interpretación

Queremos insertar un número en un arreglo que ya está ordenado, manteniendo el orden del arreglo.

7.3. Solución

7.3.1. Entender

Para insertar un número en un arreglo ordenado, debemos: - Encontrar la posición en la que el número debe ser insertado para mantener el orden. - Desplazar los elementos a partir de esa posición para hacer espacio para el nuevo número. - Insertar el número en la posición correcta.

7.3.2. Modelar

a. Encontrar la Posición de Inserción:

- Usar un bucle para encontrar la primera posición donde el número a insertar es menor o igual al elemento del arreglo.

b. Desplazar Elementos:

- Desplazar los elementos del arreglo a partir de la posición encontrada hacia la derecha para hacer espacio para el nuevo número.

c. Insertar el Número:

- Colocar el número en la posición encontrada.

7.3.3. Codificar

```
# Programa que inserta un elemento en una lista ordenada
# Importamos nuestra libreria
import MiLibreria

# Definimos nuestra funcion
def InsertarEnArregloOrdenado(A, num):
    N = len(A)

    # Encontrar la posicion de insercion
    i = 0
    while i < N and A[i] < num:
        i += 1

    # Desplazar elementos para hacer espacio
    A.append(0)  # Anadir un elemento al final para ampliar la
    lista
    j = N - 1
    while j >= i:
        A[j + 1] = A[j]
        j -= 1

    # Insertar el nuevo numero
    A[i] = num

    return A

# ------------- Ejemplo de uso ------------------
# Leer los datos
N = MiLibreria.LeerEntero("Longitud de la lista: ",1,9999)
A = MiLibreria.LeerLista("Inserte la lista: ",N)
X = MiLibreria.LeerReal("Inserte el numero a insertar: "
    ,-999999,99999)

# Mostrar resultados
print("Arreglo original:", A)
A = InsertarEnArregloOrdenado(A, X)
print("Arreglo despues de la insercion:", A)
```

8. Problema

8.1. Enunciado

Dado un arreglo A de N elementos enteros, escribir un módulo que permita eliminar un elemento X del arreglo, siempre que X esté en el arreglo.

8.2. Interpretación

Queremos eliminar una instancia de un número específico del arreglo, desplazando los elementos restantes para mantener el orden y el tamaño del arreglo ajustado.

8.3. Solución

8.3.1. Entender

Para eliminar un elemento de un arreglo: - Encontramos el primer índice del elemento a eliminar. - Desplazamos los elementos posteriores una posición a la izquierda. - Ajustamos el tamaño del arreglo.

8.3.2. Modelar

a. Encontrar el Elemento:

- Iterar sobre el arreglo para encontrar la primera aparición del elemento a eliminar.

b. Desplazar Elementos:

- Mover todos los elementos después del índice encontrado una posición hacia la izquierda.

c. Reducir Tamaño:

- Reducir el tamaño del arreglo en uno.

8.3.3. Codificar

```
# Eliminar un elemento de una lista
# Importar la libreria
import MiLibreria

# Definimos nuestra funcion
def EliminarElemento(A, X):
    # Buscar el indice del elemento a eliminar usando la funcion Buscar
    i = MiLibreria.Buscar(X, A)

    # Si el elemento fue encontrado
    if i != -1:
        N = len(A)
        # Desplazar elementos hacia la izquierda
        j = i
        while j < N - 1:
            A[j] = A[j + 1]
            j += 1
        # Eliminar el ultimo elemento duplicado
        A.pop()  # Remove the last element which is now a duplicate

    return A

# ------------- Ejemplo de uso ------------------
# Leer los datos
N = MiLibreria.LeerEntero("Longitud de la lista: ",1,9999)
A = MiLibreria.LeerLista("Inserte la lista: ",N)
X = MiLibreria.LeerEntero("Inserte el numero a eliminar: ",-999999,99999)

# Mostrar resultados
print("Arreglo original:", A)
A = EliminarElemento(A, X)
print("Arreglo despues de eliminar el elemento:", A)
```

9. Problema

9.1. Enunciado

Escribir un módulo que permita multiplicar un escalar por un vector de N elementos.

9.2. Interpretación

Queremos multiplicar cada elemento de un vector por un escalar. Es decir, dado un escalar k y un vector V de N elementos, queremos obtener un nuevo vector donde cada elemento sea el resultado de multiplicar el correspondiente elemento del vector original por el escalar.

9.3. Solución

9.3.1. Entender

Para multiplicar un escalar por un vector: - Recorrer el vector. - Multiplicar cada elemento del vector por el escalar. - Almacenar el resultado en un nuevo vector o reemplazar el vector original.

9.3.2. Modelar

a. Iterar sobre el Vector:

- Usar un bucle para recorrer el vector.

b. Multiplicar Cada Elemento:

- Multiplicar cada elemento por el escalar.

c. Actualizar el Vector:

- Guardar el resultado de la multiplicación en el mismo vector o en un nuevo vector.

9.3.3. Codificar

```
# Calcular el producto entre un escalar y un vector
# Importar nuestra libreria
import MiLibreria

# Definimos nuestra funcon
def MultiplicarEscalarPorVector(escalar, vector):
    N = len(vector)

    # Crear un nuevo vector para almacenar el resultado
    resultado = [0] * N

    # Calcular el producto
    i = 0
    while i < N:
        resultado[i] = escalar * vector[i]
        i += 1

    return resultado

# ------------- Ejemplo de uso ------------------
# Leer los datos
N = MiLibreria.LeerEntero('Dimension del vector: ',1,9999)
vector = MiLibreria.LeerLista('Inserte las componentes del vector:
    ',N)
escalar = MiLibreria.LeerReal('Inserte el escalar: ',-999999,99999)

# Mostrar resultados
print("Vector original: ", vector)
print("Escalar: ", escalar)
vector_multiplicado = MultiplicarEscalarPorVector(escalar, vector)
print("Vector despues de multiplicar por el escalar: ",
    vector_multiplicado)
```

10. Problema

10.1. Enunciado

Escribir un módulo que efectúe el producto escalar de dos vectores de N elementos.

10.2. Interpretación

Queremos calcular el producto escalar de dos vectores de tamaño N. Esto se realiza sumando el producto de cada par de elementos correspondientes en los dos vectores.

10.3. Solución

10.3.1. Entender

Para calcular el producto escalar: - Multiplicar los elementos correspondientes de los dos vectores. - Sumar todos los productos obtenidos.

10.3.2. Modelar

a. Iterar Sobre los Vectores:

- Usar un bucle para recorrer los vectores.

b. Multiplicar y Sumar:

- Multiplicar cada par de elementos correspondientes.
- Acumular la suma de estos productos.

10.3.3. Codificar

```
# Calcular el producto escalar entre dos vectores
# Imprimir la libreria
import MiLibreria

# Definimos la funcion de producto escalar
def ProductoEscalar(A, B):
    N = len(A)

    # Inicializar el producto escalar
    producto = 0

    # Calcular el producto escalar
    i = 0
    while i < N:
        producto += A[i] * B[i]
        i += 1

    return producto

# ------------- Ejemplo de uso ------------------
# Leer los datos
N = MiLibreria.LeerEntero('Dimension de los vectores: ',1,9999)
A = MiLibreria.LeerLista('Inserte las componentes del vector A: ',N
    )
B = MiLibreria.LeerLista('Inserte las componentes del vector B: ',N
    )

# Mostrar los resultados
print("Vector A:", A)
print("Vector B:", B)
print("Producto escalar:", ProductoEscalar(A, B))
```

11. Problema

11.1. Enunciado

Sea A un arreglo de N elementos enteros. Escribir un módulo que determine cuántas veces se repite el elemento X en el arreglo.

11.2. Interpretación

Queremos contar cuántas veces aparece un elemento específico X en un arreglo A. Este proceso implica recorrer el arreglo y contar cada coincidencia con el valor X.

11.3. Solución

11.3.1. Entender

Para contar las apariciones de un elemento X: - Recorrer el arreglo. - Contar las veces que el elemento actual del arreglo coincide con X.

11.3.2. Modelar

a. Inicializar el Contador:

- Comenzar con un contador en cero.

b. Iterar sobre el Arreglo:

- Para cada elemento del arreglo, verificar si es igual a X.
- Incrementar el contador si hay una coincidencia.

c. Devolver el Contador:

- Al final de la iteración, devolver el valor del contador.

11.3.3. Codificar

```
# Programa que cuenta las apariciones de un elemento en una lista
# Importar nuestra libreria
import MiLibreria

# Definir la funcion
def ContarApariciones(A, X):
    N = len(A)
    contador = 0
    i = 0
    while i < N:
        if A[i] == X:
            contador += 1
        i += 1
    return contador

# ------------- Ejemplo de uso ------------------
# Leer los datos
N = MiLibreria.LeerEntero("Longitud de la lista: ",1,9999)
A = MiLibreria.LeerLista("Inserte la lista: ",N)
X = MiLibreria.LeerEntero("Inserte el numero a buscar: "
    ,-999999,99999)

# Mostrar resultados
print("Arreglo: ", A)
print("Elemento a contar: ", X)
print("Numero de apariciones de X en el arreglo: ",
    ContarApariciones(A, X))
```

12. Problema

12.1. Enunciado

Sea A un arreglo ordenado ascendentemente de N elementos. Escribir un módulo que genere un arreglo B ordenado descendentemente con los elementos del arreglo A.

12.2. Interpretación

Queremos tomar un arreglo A que está ordenado en orden ascendente y crear un nuevo arreglo B que contenga los mismos elementos pero ordenados en orden descendente.

12.3. Solución

12.3.1. Entender

Para crear un arreglo ordenado en orden descendente a partir de un arreglo ordenado ascendentemente: - Recorremos el arreglo original desde el último elemento hasta el primero. - Añadimos cada elemento a un nuevo arreglo.

12.3.2. Modelar

a. Inicializar el Arreglo Resultante:

- Crear un nuevo arreglo vacío.

b. Recorrer el Arreglo Original al Revés:

- Iterar desde el último elemento hasta el primero.

c. Agregar Elementos al Nuevo Arreglo:

- Añadir cada elemento al nuevo arreglo.

12.3.3. Codificar

```
# Programa que reordena descendentemente una lista ascendente
# Importar nuestra libreria
import MiLibreria

# Definimos la funcion de reordenar
def GenerarDescendente(A):
    N = len(A)
    B = [0] * N

    i = N - 1
    j = 0
    while i >= 0:
        B[j] = A[i]
        i -= 1
        j += 1

    return B

# ------------- Ejemplo de uso ------------------
# Leer los datos
N = MiLibreria.LeerEntero("Longitud de la lista: ",1,9999)
A = MiLibreria.LeerLista("Inserte la lista: ",N)

# Mostrar los resultados
print("Arreglo A (ascendente):", A)
B = GenerarDescendente(A)
print("Arreglo B (descendente):", B)
```

13. Problema

13.1. Enunciado

Los artículos de un almacén están codificados con números secuenciales del 1 hasta N. En un arreglo se tiene el número de unidades vendidas de cada artículo, en otro arreglo se tiene el precio unitario de cada artículo. Escribir un módulo que determine el precio total de venta de cada artículo y el gran total.

13.2. Interpretación

Queremos calcular: 1. El precio total de venta de cada artículo. 2. El gran total de ventas sumando todos los precios totales.

13.3. Solución

13.3.1. Entender

Para cada artículo: - Calcular el precio total multiplicando las unidades vendidas por el precio unitario. - Sumar todos estos precios totales para obtener el gran total.

13.3.2. Modelar

a. Inicializar Variables:

- Crear un arreglo para almacenar el precio total de cada artículo.
- Inicializar el gran total a cero.

b. Calcular el Precio Total de Cada Artículo:

- Recorrer ambos arreglos simultáneamente.
- Multiplicar el número de unidades vendidas por el precio unitario y almacenar el resultado.

c. Calcular el Gran Total:

- Sumar todos los precios totales calculados.

13.3.3. Codificar

```
# Programa para calcular los precios
# Importar nuestra libreria
import MiLibreria

# Definir nuestra funcion
def CalcularPreciosTotales(unidades_vendidas, precios_unitarios):
    N = len(unidades_vendidas)

    precios_totales = [0] * N
    gran_total = 0

    i = 0
    while i < N:
        precios_totales[i] = unidades_vendidas[i] *
    precios_unitarios[i]
        gran_total += precios_totales[i]
        i += 1

    return precios_totales, gran_total

# ------------- Ejemplo de uso ------------------
# Leer los datos
N = MiLibreria.LeerEntero("Numero de objetos en venta: ",1,9999)
unidades_vendidas = MiLibreria.LeerLista("Inserte las unidades
    vendidas: ",N)
precios_unitarios = MiLibreria.LeerLista("Inserte los precios
    unitarios: ",N)

# Mostrar los resultados

print("Numero de unidades vendidas de articulos: ",
    unidades_vendidas)
print("Precios unitarios de los articulos: ",precios_unitarios)

precios_totales, gran_total = CalcularPreciosTotales(
    unidades_vendidas, precios_unitarios)

print("Precios totales de cada articulo:", precios_totales)
print("Gran total de ventas:", gran_total)
```

14. Problema

14.1. Enunciado

Se tienen cuatro arreglos paralelos: en los tres primeros se tienen las notas correspondientes a los tres exámenes parciales y en el cuarto arreglo se tiene la nota correspondiente al examen sustitutorio. Escribir un módulo que genere un quinto arreglo con los promedios finales, considerando que el examen sustitutorio reemplaza al examen parcial más bajo, siempre y cuando sea mayor que éste.

14.2. Interpretación

Queremos calcular el promedio final para cada estudiante de acuerdo con las siguientes reglas: 1. El examen sustitutorio puede reemplazar el examen parcial más bajo si es mayor que éste. 2. Calcular el promedio final con las notas ajustadas.

14.3. Solución

14.3.1. Entender

Para cada estudiante: - Identificar el examen parcial más bajo. - Comparar la nota del examen sustitutorio con el examen parcial más bajo. - Si la nota del sustitutorio es mayor, reemplazar el examen parcial más bajo. - Calcular el promedio final de los tres exámenes (considerando el ajuste) y almacenar el resultado en el quinto arreglo.

14.3.2. Modelar

a. Inicializar el Arreglo de Promedios Finales:

- Crear un nuevo arreglo para almacenar los promedios finales.

b. Recorrer Cada Estudiante:

- Encontrar el examen parcial más bajo.
- Comparar y reemplazar con el examen sustitutorio si es necesario.
- Calcular el promedio final y almacenarlo en el nuevo arreglo.

c. Calcular el Promedio Final:

- Sumar las notas ajustadas y dividir por 3.

14.3.3. Codificar

```python
# Programa que calcula los promedios finales
# Importamos nuestra libreria
import MiLibreria

# Definimos las funciones
def CalcularPromediosFinales(parcial1, parcial2, parcial3, sustitutorio):
    N = len(parcial1)
    promedios_finales = [0] * N

    i = 0
    while i < N:
        # Encontrar el minimo entre los tres parciales
        min_parcial = min(parcial1[i], parcial2[i], parcial3[i])

        # Determinar si el sustitutorio reemplaza al minimo
        if sustitutorio[i] > min_parcial:
            if min_parcial == parcial1[i]:
                parcial1[i] = sustitutorio[i]
            elif min_parcial == parcial2[i]:
                parcial2[i] = sustitutorio[i]
            else:
                parcial3[i] = sustitutorio[i]

        # Calcular el promedio final
        promedio = (parcial1[i] + parcial2[i] + parcial3[i]) / 3
        promedios_finales[i] = promedio

        i += 1

    return promedios_finales

# ------------- Ejemplo de uso ------------------
# Leer los datos
N = MiLibreria.LeerEntero("Numero de alumnos: ",1,9999)
parcial1 = MiLibreria.LeerLista("Inserte las notas del parcial 1: ",N)
parcial2 = MiLibreria.LeerLista("Inserte las notas del parcial 2: ",N)
parcial3 = MiLibreria.LeerLista("Inserte las notas del parcial 3: ",N)

sustitutorio = MiLibreria.LeerLista("Inserte las notas del sustitutorio: ",N)

# Mostrar los resultados
print("Notas del primer parcial: ", parcial1)
print("Notas del segundo parcial: ", parcial2)
print("Notas del tercer parcial: ", parcial3)
print("Notas del sustitutorio: ", sustitutorio)

promedios_finales = CalcularPromediosFinales(parcial1, parcial2, parcial3, sustitutorio)

print("Promedios finales: ", promedios_finales)
```

15. Problema

15.1. Enunciado

Sea A un arreglo de N elementos enteros. Escribir un módulo que elimine los elementos duplicados del arreglo.

15.2. Interpretación

Queremos generar un nuevo arreglo que contenga solo las primeras ocurrencias de cada elemento del arreglo original, eliminando así los duplicados.

15.3. Solución

15.3.1. Entender

Para eliminar duplicados de un arreglo: - Usaremos un nuevo arreglo para almacenar elementos únicos. - Iteraremos sobre el arreglo original y verificaremos si cada elemento ya está en el nuevo arreglo.

15.3.2. Modelar

a. Inicializar el Arreglo de Resultados:

- Crear un nuevo arreglo vacío para almacenar elementos únicos.

b. Recorrer el Arreglo Original:

- Para cada elemento del arreglo original, verificar si ya está en el nuevo arreglo.
- Si no está, agregarlo al nuevo arreglo.

c. Devolver el Nuevo Arreglo:

- Devolver el arreglo que contiene solo elementos únicos.

15.3.3. Codificar

```
# Programa que elimina los elementos duplicados de una lista
# Importamos nuestra libreria
import MiLibreria

# Definir nuestra funcion
def EliminarDuplicados(A):
    resultado = []
    i = 0
    N = len(A)

    while i < N:
        # Usar la funcion Buscar para verificar si el elemento actual ya esta en el arreglo resultado
        if MiLibreria.Buscar(A[i], resultado) == -1:
            resultado.append(A[i])

        i += 1

    return resultado

# ------------- Ejemplo de uso ------------------
# Leer los datos
N = MiLibreria.LeerEntero('Longitud de la lista: ',1,9999)
A = MiLibreria.LeerLista('Inserte el arreglo: ',N)


print('Arreglo original: ', A)
A_sin_duplicados = EliminarDuplicados(A)
print("Arreglo sin duplicados:", A_sin_duplicados)
```

16. Problema

16.1. Enunciado

Escribir un módulo que determine el valor numérico de un polinomio para un valor dado de X.

16.2. Interpretación

Dado un polinomio representado por un arreglo de coeficientes y un valor específico de X, necesitamos calcular el valor del polinomio cuando se evalúa en X.

16.3. Solución

16.3.1. Entender

Para calcular el valor del polinomio: - Evaluaremos el polinomio en el valor dado de X usando la fórmula:

$$P(X) = a_0 + a_1X + a_2X^2 + \cdots + a_{N-1}X^{N-1}$$

- Donde a_i son los coeficientes del polinomio.

16.3.2. Modelar

a. Inicializar el Resultado:

- Crear una variable para acumular el resultado de la evaluación del polinomio.

b. Recorrer los Coeficientes:

- Para cada coeficiente en el arreglo, calcular su contribución al valor final del polinomio.

c. Sumar la Contribución de Cada Término:

- Utilizar un bucle ‘while‘ para recorrer los coeficientes y calcular el valor del polinomio.

16.3.3. Codificar

```
# Programa que evalua un polinomio en un determinado X
# Importamos nuestra libreria
import MiLibreria

# Definimos nuestra funcion
def EvaluarPolinomio(coeficientes, X):
    N = len(coeficientes)
    resultado = 0
    i = 1

    while i <= N:
        # Calcular el valor del termino actual
        termino = coeficientes[N-i] * (X ** i)
        # Sumar el termino al resultado
        resultado += termino
        i += 1

    return resultado

# ------------- Ejemplo de uso ------------------
# Leer los datos
N = MiLibreria.LeerEntero("Grado del polinomio: ",1,9999)
coeficientes = MiLibreria.LeerLista("Inserte la lista: ",N+1)
X = MiLibreria.LeerReal("Valor de X: ",1,9999)

# Mostrar resultado
MiLibreria.ImprimirPolinomio(coeficientes,"P")
valor_polinomio = EvaluarPolinomio(coeficientes, X)

print("P(", X, ") = ", valor_polinomio)
```