**Ejercicios iteración**

**Ejercicio 6**

1. Algoritmo: ejercicio cuenta historial

Entrada:

dinero\_ingresado: int #el dinero que se ingresa, ya sea manual o por saldo dinero\_transferido: int #el dinero que se transfiere, ya sea una pérdida o una ganancia

dinero\_sacado: int #el dinero que se saca de la cuenta

fechas\_entrada: tiempo #las fechas en las que entra en su cuenta ya haya modificaciones en el dinero o no

meses(list) #lista donde se almacenan los meses

Resultado:

historial: list(strings) # recopilamos todos los datos que se ha realizado en la cuenta durante un mes y un año

dinero\_mensual: int #dinero que tiene al final del mes

dinero\_anual: int #dinero que tiene al final del año

Precondiciones:

dinero\_ingresado ≥ 0

dinero\_sacado ≤ 0

dinero\_transferido ≥ 0 (si el dinero se transfiere a la cuenta del usuario)

dinero\_transferido ≤ 0 (si se transfiere dinero desde la cuenta del usuario)

fechas\_entrada ≥ 0

Realizacion:

Si:

dinero\_ingresado > 0

entones:

Resultado: "Ha recibido dinero en tu cuenta "

Si no:

entonces:

Resultado: "No ha recibido dinero en tu cuenta"

Si:

dinero\_sacado < 0

entones:

Resultado: "Ha retirado dinero de tu cuenta "

Si:

dinero\_transferido > 0

entones:

Resultado: "Le han tranferido dinero a su cuenta "

Si no:

dinero\_transferido < 0

entonces:

Resultado: "Has transferido dinero de tu cuenta"

Si:

fechas\_entrada > 0

entones:

Resultado: "Ha entrado en su cuenta el día x"

Si no:

fechas\_entrada < 0

entonces:

Resultado: "No ha entrado en su cuenta el día x"

Inicialización:

dinero\_mensual = dinero\_ingresado + dinero\_sacado + dinero\_transferido

i ← 1 # La variable i representará los meses del año

# A i se le sumará 1 cada vez hasta que llegue a 12

Mientras:

i ≤ 12:

# Calcularemos el dinero\_mensual de cada mes

# Añadiremos lo que hemos calculado antes a meses

i = i + 1

Resultado: meses

Postcondiciones:

Historial: Sumatorio de todos los posibles resultados anteriores

Resultado:

meses(lista) #lista del dinero restante a final de cada mes

dinero\_anual: sumatorio de elementos de meses

Entrada:

importe\_movimientos(int) #basado en los datos de CUENTA

clientes(tabla) #tabla con los clientes y sus respectivas cuentas

clientes\_resultado(list) #lista con los clientes cuya media de importes es mayor a importe\_movimientos

Resultado: clientes\_resultado

Precondiciones:

importe\_movimientos ≥ 0

clientes > 0

clientes\_resultado ≥ 0

Inicialización:

i = 1 #columna de importes de la tabla clientes

j #filas de la tabla clientes

Realización

ꓯ j ∈ (tabla):

#recorremos la tabla clientes, omitiendo la columna en la que se encuentran los nombres de usuario.

# importe\_usuario es un dato dado fijo. Es el importe de cada uno de los clientes que aparecen en la tabla clientes.

#nombre\_usuario es un dato fijo. Es el nombre de cada cliente que aparece en la tabla cliente.

Si:

importe\_usuario > importe\_movimientos:

#añadimos nombre\_usuario en clientes\_resultado

Resultado: clientes\_resultado

Postcondición:

Resultado: clientes\_resultado

**Ejercicio 8**

Algoritmo: Descomponer cadena

Entrada:

cadena (string) #cadena a descomponer

separador (string) #carácter que usaremos para separar la cadena

tabla(cadena) (list) #tabla creada a partir de los caracteres de la cadena

j (int) #indice donde se encuentra el separador

subtabla (list) # sublista donde se separarán los caracteres

i (int) #índice de la tabla

k (int) #longitud de la cadena

Precondición:

cadena > 0

separador > 0

j > 0

k > 0

i > 0

Resultado: tabla(cadena)

Inicialización:

tabla(cadena)

∀ i ∈ tabla:

# recorre la tabla en busca de separadores

Realización:

Si:

“:” está en tabla

entonces:

Resultado: subtabla # tiene los caracteres desde (j + 1) hasta i

Mientras que:

i ≠ k

#repetimos la parte de la realización

Postcondición:

Resultado: tabla de subtablas

**Ejercicio 9**

Algoritmo: Búsqueda en diccionario

Entrada:

inicial(string) # la inicial a buscar en el diccionario

diccionario(tabla) #diccionario en el que se busca

índice(int) #índice de la palabra buscada en el diccionario

Resultado(int)

Precondición:

inicial es una letra alfabética

Inicialización:

i ← I MIN

Realización:

hasta que

i ≤ I\_MAX

Postcondición:

resultado(int) #es el índice de una palabra

**Ejercicio 10**

1. Diagrama UML
2. Algoritmo: Lista de personas

Entrada:

personas(clase)

identidad(clase)

media\_edad(lista) #lista que vamos a crear con los datos

Precondiciones:

personas(clase) > 0

identidad(clase) > 0

Resultado: lista

Realización:

Si:

20 ≤ atributo\_edad ≤ 30

entonces:

Resultado: añadir atributo\_nombre a media\_edad

Postcondiciones:

Resultado: media\_edad #lista

1. ∀ atributo\_edad ∈ personas

atributo\_edad = atributo\_edad + 1

# volveríamos a repetir la parte dos pero con estas edades cambiadas

Algoritmo: Lista de personas

Entrada:

personas(clase)

identidad(clase)

huerfalista(lista) #lista que vamos a crear con los datos

Precondiciones:

personas(clase) > 0

identidad(clase) > 0

Resultado: lista

Realización:

∀ huérfano ∈ personas

Si:

edad < 15

entonces:

Resultado: añadir nombre\_huérfano a la lista huerfalista

Postcondiciones:

Resultado: huerfalista #lista

Algoritmo: Padre Jaime

Entrada:

personas(clase)

identidad(clase)

nombre\_padre(string) #es un atributo de la clase persona que tiene el nombre del padre de dicha persona

tiene\_padre(string) # Es una función de identidad que devuelve el nombre del padre de la variable.

Precondiciones:

personas(clase) > 0

identidad(clase) > 0

tiene\_padre ≥ 0

Resultado: string

Realización:

atributo\_nombre = “Jaime Martín”

si:

tiene\_padre(atributo\_nombre) = nombre\_padre

entonces:

Resultado: nombre\_padre

6

Algoritmo: Hermanos y hermanas de Jaime

Entrada:

personas(clase)

Identidad(clase)

nombre\_padre(string) #es un atributo de la clase identidad que tiene el nombre del padre de dicha persona

nombre\_madre(string) #es un atributo de la clase identidad que tiene el nombre de la madre de dicha persona

descendencia(función) #es una función que recibe dos variables en forma de nombres y añade en lista\_hijos los nombres de los hijos

lista\_hijos(lista) # es una lista vacía a la que se añaden nombres

Resultado: lista

Precondiciones:

personas(clase) > 0

identidad(clase) > 0

nombre\_madre ≥ 0

nombre\_padre ≥ 0

descendencia ≥ 0

Realización:

atributo\_nombre = “Jaime Martín”

descendencia(nombre\_padre, nombre\_madre) = lista\_hijos #es una lista en la que se añaden los hijos de las variables.

Postcondiciones:

Resultado: lista\_hijos(lista)

**Ejercicio 11.**

1. Algoritmo: mcd con Euclides:

Entrada:

numero1(int) #es el primer número que se da

numero2(int) #es el segundo número que se da

cociente(int)

resto(int)

Precondiciones:

numero1 > / < 0

numero2 > / < 0

numero1 ≥ numero2

cociente(int) > / < 0 #cociente de la división

resto(int) ≥ 0 # resto de la división

Resultado: int

Realización:

división = numero1 / numero2

numero1 = numero2 x cociente + resto

Si:

resto = 0

entonces:

Resultado: numero2

Si no:

# repetiremos división hasta obtener el resto 0

numero1 ← numero 2

numero2 ← resto

entonces:

Resultado: numero2# El nuevo que obtenemos tras realizar este procedimiento

Postcondiciones:

Resultado: siempre el numero2

1. Algoritmo: mcd con sumas o restas:

Entrada:

numero1(int) #es el primer número que se da

numero2(int) #es el segundo número que se da

cociente(int)

resto(int)

Precondiciones:

numero1 ≥ 0

numero2 >0

numero1 ≥ numero2

cociente(int) > / < 0 #cociente de la división

resto(int) ≥ 0 # resto de la división

Resultado: int

Realización:

división = numero1 / numero2

numero1 = numero2 x cociente + resto

k # parte entera del cociente

Si:

resto ≠ 0

entonces:

∀ i ∊ [0, k] # va desde 0 hasta el número entero del cociente

Resultado: numero2 +...+ numero2≤ numero1 < numero2 +…+ numero2

Si no:

entonces:

#Se hace el mismo procedimiento anteriormente

Resultado: sería el mismo

Postcondiciones:

Resultado: numero2 +...+ numero2≤ numero1 < numero2 +…+ numero2

**Ejercicio 12:**

1. Algoritmo: cuadrados perfectos lista

Entrada:

limite(int) #es el límite hasta el que se van a hacer los cuadrados perfectos

cuadrados(lista) #es la lista que va a almacenar los números con cuadrados perfectos

hacer\_cuadrado(funcion) #es una función que calcula la raíz cuadrada del número que se le envía

Precondiciones:

límite ≥ 0

Resultado: lista

Realización:

∀ n ∊ [0, limite]

hacer\_cuadrado(n) = resultado

Si:

resultado es un número entero

entonces:

añadir n a cuadrados(lista)

#repetir hasta límite

Postcondiciones:

Resultado: cuadrados

1. Algoritmo: Cálculo raíz cuadrada número entero

Entrada:

limite(int) #es el límite hasta el que se van a hacer los cuadrados perfectos

cuadrados(lista) #es la lista que va a almacenar los números con cuadrados enteros

hacer\_cuadrado(funcion) #es una función que calcula la raíz cuadrada del número que se le envía

Precondiciones:

límite ≥ 0

Resultado: lista

Realización:

∀ n ∊ [0, limite] #n son los números que nos dan

hacer\_cuadrado(n) = resultado

k = valor entero del resultado

Si:

resultado ≠ número entero

entonces:

≤ n <

Resultado: k que se añadirá a cuadrados(lista)

#repetir hasta límite

Postcondiciones:

Resultado: cuadrados