GUÍA DE TRABAJO - ALGORITMOS SECUENCIALES

1. INFORMACIÓN DE LA ASIGNATURA

Asignatura:	Lógica y Representación I			
Código:	2554208		**Tipo de Curso:**	Obligatorio
Plan de Estudios:	5		**Semestre:**	2
Créditos: 3	**TPS:** 6	**TIS:** 3	**TPT:** 64	**TIT:** 32

2. OBJETIVO

Unidad:	Algoritmos secuenciales
Objetivo de la asignatura:	Desarrollar en el estudiante la habilidad para plantear soluciones lógicas a problemas computacionales que deriven en la implementación de programas en un lenguaje de programación
Resultados de aprendizaje abordados:	▪ Analizar un problema identificando correctamente las entradas, el proceso y la salida asociados a su solución ▪ Escribir algoritmos y programas que solucionen problemas computacionales
Contenido temático:	▪ Definición de algoritmo ▪ Formas de representación de un algoritmo ▪ Estructura de un algoritmo en seudocódigo ▪ Elementos de un algoritmo en seudocódigo

3. DESARROLLO TEÓRICO

3.1 INTRODUCCIÓN

La primera etapa en el proceso de desarrollo de software es el *Análisis del Problema*. En esta etapa se busca comprender de manera detallada cuál es el problema que se debe resolver y al cual se le dará una solución mediante la construcción de un producto de software. Así, el objetivo principal de esta etapa es identificar y definir claramente los requisitos (o funcionalidades) que tendrá el software a fin de garantizar que el producto final cumpla con las expectativas del cliente o usuario. Igualmente, en esta etapa se deben identificar las entidades del mundo del problema que serán modeladas y las cuales representan los componentes principales del sistema que se va a construir. Así, las *Entidades del Mundo del Problema* son objetos, conceptos o cualquier elemento relevante en el dominio del problema que se deba considerar en el desarrollo del software. Estas entidades reflejan las "cosas" con las que el software interactuará, gestionará o sobre las que realizará operaciones.

Posteriormente, en la etapa de *Diseño de la Solución*, se define el *cómo* el programa realizará las funcionalidades que se han identificado. Es decir, en esta se especifica, usando un lenguaje textual o gráfico, la manera como se implementarán las funcionalidades del programa para solucionar el problema. Un elemento de la etapa de diseño es el *Diagrama de Clases*, el cual muestra las entidades del mundo del problema y sus atributos, y las relaciones que hay entre esas entidades. Otro elemento en la etapa de diseño consiste en la construcción de los *Algoritmos*, los cuales detallan el comportamiento de las entidades del mundo, pero ¿qué es un algoritmo?

3.2 Definición de Algoritmo

Un *Algoritmo* es una secuencia de pasos ordenados, bien definidos y finitos que dan solución a un problema. Esta definición encierra en si misma las características que debe tener un algoritmo.

Todo algoritmo debe ser:

- **Ordenado:** el seguimiento de los pasos en el orden estricto definido debe solucionar el problema
- **Bien definido:** cada paso debe estar claramente descrito sin ambigüedades
- **Finito:** el algoritmo debe terminar después de un número finito de pasos
- **Determinista:** la salida del algoritmo deber ser siempre la misma cuando se usan las mismas entradas
- **Legible:** cualquiera que lea el algoritmo debe ser capaz de comprenderlo

3.3 Formas para Representar un Algoritmo

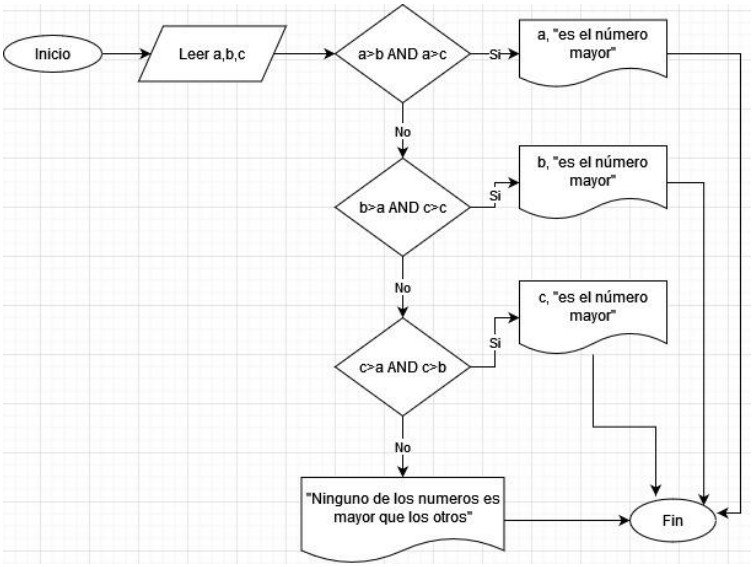
Existen diferentes formas de representar un algoritmo. La más simple es el *Lenguaje Natural*. Esta forma de representación es ampliamente utilizada para solucionar problemas cotidianos y cualitativos; es decir, problemas en los que la solución se define usando el lenguaje con el que nos comunicarnos día a día. Un ejemplo de este tipo de algoritmos son las recetas de cocina. Por ejemplo, un algoritmo que nos permitiría resolver el problema de preparar una libra de arroz es el siguiente:

```
Ingredientes:
   Una libra de arroz
   Cuatro tazas de agua
   Una cebolla picada
   Una cucharada de aceite
   Una cucharadita de sal
   Una olla mediana con tapa

Inicio
   1. Ponga la olla a fuego medio en la estufa
   2. Agregue el aceite a la olla y espere un momento a que se caliente
   3. Agregue la cebolla y sofríala por un minuto
   4. Agregue el arroz, revuélvalo y sofría por un minuto más
   5. Agregue el agua y la sal
   6. Revuelva y suba el fuego a medio-alto
   6. Cuando el agua haya casi secado por completo, baje el fuego a bajo y tape la olla
   7. Trascurridos 15 minutos apague la estufa y deje reposar el arroz por 5 minutos más
Fin
```

A pesar de su simplicidad, el lenguaje natural no es la mejor opción para escribir algoritmos porque tiende a ser ambiguo, esto sucede porque este lenguaje carece de una estructura precisa que permita representar un concepto siempre de la misma manera. Por esta razón los algoritmos en lenguaje natural no suelen usarse para solucionar problemas que conllevan a la implementación un programa de computador.

Otra forma para representar un algoritmo son los *Diagramas de Flujo*. Estos tienen una estructura más formal que limita la interpretación de las instrucciones que definen la secuencia de pasos que dan solución al problema. Los diagramas de flujo pueden usarse para escribir algoritmos cualitativos (como la receta de cocina), pero también pueden usarse para escribir algoritmos cuantitativos; es decir, aquellos algoritmos que requieren hacer cálculos matemáticos. Entre las ventajas de los diagramas de flujo está su facilidad de comprensión, esto porque que el cerebro humano interpreta más fácilmente una representación gráfica que una representación textual. No obstante, una desventaja de este tipo de representación es que su facilidad de interpretación disminuye a medida que aumenta la complejidad del problema, esto porque el diagrama crece considerablemente cuando el problema se hace más complejo. Esta es la principal razón por la cual los diagramas de flujo no suelen usarse para solucionar problemas de complejidad media que conllevan a la implementación un programa de computador. A continuación, se presenta el ejemplo de un diagrama de flujo para resolver el problema de determinar cuál es el mayor de tres números que serán ingresados por el usuario.



Otra forma para representar un algoritmo es utilizando un pseudo lenguaje llamado *Seudocódigo (o Pseudocódigo)*. La ventaja de esta forma de representación radica en su capacidad para describir la lógica de la solución de un problema computacional sin preocuparse por la sintaxis de un lenguaje de programación en particular. Al ser un pseudo lenguaje, el seudocódigo no puede ejecutarse o ser interpretado por una máquina, pero es completamente comprensible para los humanos.

Como veremos, el seudocódigo usa un conjunto de *Palabras Reservadas* en español (o inglés), las cuales, a pesar de no estar estandarizadas, siguen una notación comprensible para todos los programadores ya que comparten similitudes con los lenguajes de programación reales en términos de estructuras de control y expresiones lógicas. Vemos un fragmento de seudocódigo para dar solución al problema de determinar cuál es el mayor de tres números deben ser ingresados por un usuario.

```
Inicio
  Entero a, b, c

  Escribir ("Ingrese tres números: ")
  Leer (a, b, c)
  Si (a > b and a > c) Entonces
    Escribir(a, "es el número mayor de los números ingresados")
  Sino
    Si (b > a and b > c) Entonces
      Escribir(b, "es el número mayor de los números ingresados")
    Sino
      Si (c > a and c > b) Entonces
        Escribir(c, "es el número mayor de los números ingresados")
      Sino
        Escribir("Ninguno de los números es mayor que los otros")
      Fin_Si
    Fin_Si
  Fin_Si
Fin
```

3.4 Instrucciones y Estructura de un Algoritmo en Seudocódigo

El pseudocódigo tiene algunas reglas de sintaxis y de gramática básicas que están definidas sobre diversos elementos del seudo lenguaje. Estos elementos son los comentarios, variables y constantes, operadores, expresiones y las estructuras de control. El uso de estos elementos nos permite escribir algoritmos que puedan ser leídos e interpretados por el equipo de desarrollo. Veamos la sintaxis y gramática de estos elementos en el seudo lenguaje que usaremos en este curso.

3.4.1 COMENTARIOS

Los comentarios son fragmentos de texto que nos permite explicar, aclarar o documentar lo que hace una sección del código. En el sentido estricto, los comentarios no son instrucciones y, por tanto, la máquina los ignora cuando las encuentra. Los comentarios están destinados a los programadores para mejorar la comprensión y el mantenimiento del código.

Dependiendo del lenguaje, pueden existir dos tipos de comentarios:

- **Comentarios de una línea:** se utilizan para explicar de manera muy corta una instrucción, de ahí que ocupen una sola línea. En nuestro caso, los comentarios de una sola línea inician con el símbolo **#** y finaliza al terminar la línea donde se inició.

```
# Esto es un comentario, esta línea será ignorada
```

- **Comentarios de varias líneas:** se utilizan para explicar el funcionamiento de un bloque de código a lo largo de varias líneas. En nuestro caso, un comentario de múltiples líneas inicia con el símbolo **"""** y termina cuando se encuentra el símbolo **"""**.

```
""" Este es un comentario de varias líneas,
     es decir, todo lo que está entre estos dos símbolos es ignorado. """
```

3.4.2 VARIABLES Y CONSTANTES

Todo dato que se utiliza en un programa se almacena en la memoria RAM. Esa memoria está dividida en "contenedores", todos del mismo tamaño, en los que la información se almacena en formato binario. Cada contenedor tiene asociado una dirección única que permite a un programa (incluido el sistema operativo) acceder a la información almacenada en ese contenedor. Las direcciones de la memoria RAM están representadas, normalmente, por un código hexadecimal, como se ejemplifica a continuación.

Contenido de la RAM	Dirección Física
10000010	0x00A01
00110011	0x00A02
01110111	0x00A03

Por la complejidad inherente de manejar las direcciones de la memoria RAM, el programador usa en su lugar *identificadores* para almacenar los datos del programa en la memoria RAM, esto sin preocuparnos de las direcciones en hexadecimal. Así, cuando hablamos de una *variable* nos referimos a un *nombre simbólico* que nos permite almacenar un valor en la memoria RAM, el cual que puede cambiar durante la ejecución del programa. Por otro lado, cuando hablamos de una *constante* nos referimos a un *identificador* que nos permite almacenar en la RAM un valor que permanece constante durante la ejecución del programa.

En los lenguajes de programación tipados, antes de usar una variable o una constante esta debe ser declararla para que la máquina separe el espacio correspondiente en la memoria RAM para almacenar un valor. El espacio que se le asigna a una variable o constante depende de su tipo de dato. Veamos cuales son los tipos de datos comúnmente usados en los programas.

- **Tipos de datos numéricos.** Existen dos tipos de datos numéricos: enteros y reales.
  - **Entero:** una variable declarada como Entero sólo puede almacenar números enteros, es decir, números que no tienen decimales. Este tipo de variables suele ocupar 32 bits de la memoria RAM.

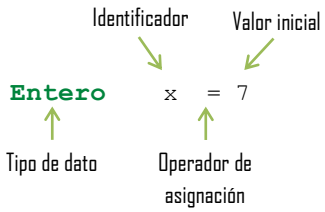
- **Real**: una variable declarada como Real puede almacenar número con coma flotante, incluyendo números enteros. Este tipo de variables tiene un rango mayor que las variables de tipo Entero puesto que suelen ocupar 64 bits.

- **Tipo de dato Lógico o Booleano:** las variables y constantes con este tipo de dato ocupan 1 bit en la memoria RAM puesto que son variables que toman uno de dos valores: `True` o `False`.
- **Tipo de dato Carácter:** las variables y constantes con este tipo de dato suelen ocupar 8 bits en la memoria RAM. Un carácter es cualquier número, letra o símbolo. Los valores de tipo Carácter se encierran en comillas simples, por ejemplo: 'A', '3', '#'.
- **Tipo de dato Cadena o Texto:** las variables y constantes de tipo Cadena no tienen un tamaño fijo definido en la memoria RAM, esto porque tienen una cantidad indeterminada de caracteres. Los valores de tipo Cadena se encierran en comillas dobles, por ejemplo: "Esto es una cadena", "Hola", "El resultado es: ".

Una vez identificados los tipos de datos, la forma general para declarar variables y constantes en pseudocódigo es como sigue:

```
<Tipo_de_dato> identificador = valor_inicial
```

Un ejemplo particular de esta forma general es la declaración de una variable entera a la que llamaremos `x` y que tiene valor inicial `7`:



Para el caso de las constantes se debe tener presente que sus identificadores se escriben en mayúscula sostenida, por ejemplo, para definir el valor de PI como constante, lo hacemos así:

```
Real PI = 3.14159
```

3.4.3 Operadores

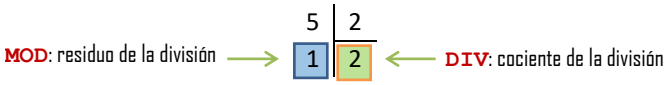
Un algoritmo se construye a partir de expresiones, y las expresiones están definidas sobre operandos y operadores. En un algoritmo existen 5 tipos de operadores: aritméticos, relacionales, lógicos, de asignación y de entrada/salida.

- *Operadores Aritméticos*: se usan para construir expresiones aritméticas que operan datos de tipo numérico. El resultado de estos operadores también es de tipo numérico. Estos son los operadores aritméticos:

Operador	Símbolo	Ejemplo	Resultado
Potencia	`**`, `^`	`2 ^ 3`	`8`
Multiplicación	`*`	`4 * 3`	`12`
División real	`/`	`5 / 2`	`2.5`
División entera	`DIV`	`5 DIV 2`	`2`
Residuo (o módulo)	`MOD`	`5 MOD 2`	`1`
Suma	`+`	`6 + 4`	`10`
Resta	`-`	`3 - 15`	`-12`

Importante: los operadores `MOD` y `DIV` se usan para calcular el residuo y el cociente de una división entera. Tenga presente que el operador `/` se usa para calcular una división de tipo real; es decir, una división cuyo resultado puede ser un número con decimales. Por el contrario, el operador `DIV` calcula una división entera y en consecuencia su resultado siempre será un número entero.

Veamos cómo se calcula el `DIV` y el `MOD`:



El orden de aplicación de los operadores aritméticos es como sigue:

- Primero se resuelven las operaciones en paréntesis
- Después se resuelven las potencias
- A continuación, se resuelven las multiplicaciones y divisiones en orden de aparición de izquierda a derecha
- Finalmente, se resuelven las sumas y las restas, en orden de aparición de izquierda a derecha

Nota: el módulo y el cociente tienen la misma precedencia de las multiplicaciones y divisiones.

- *Operadores Relacionales*: nos permiten comparar dos valores. El resultado de un operador relacional es un valor lógico (`True` o `False`). En seudocódigo los operadores relacionales son:

Operador	Símbolo	Ejemplo	Resultado
Igual que: ¿Son iguales los valores?	`==`	`"Hola" == "hola"`	`False`
Diferente: ¿Son distintos los valores?	`<>`, `!=`	`7 != -7.5`	`True`
Menor que: ¿Es un valor menor que el otro?	`<`	`4 < -1`	`False`
Mayor que: ¿Es un valor mayor que el otro?	`>`	`4 > -1`	`True`
Menor o igual que: ¿Es un valor menor o igual que el otro?	`<=`	`6 <= 6`	`True`
Mayor o igual que: ¿Es un valor mayor o igual que el otro?	`>=`	`6 >= 15`	`False`

Nota: los operadores `==` y `!=` permiten comparar valores numéricos entre sí, cadenas de texto entre sí o valores lógicos entre sí. Por otro lado, los operadores `<`, `>`, `<=` y `>=` sólo se usan para comparar valores numéricos.

El orden de aplicación de los operadores relacionales es el siguiente:

- Primero se evalúan los paréntesis
- Después se evalúan los operadores: `<`, `>`, `<=` y `>=`
- Después se evalúan los operadores: `==` y `!=`

- *Operadores Lógicos*: se utilizan para construir expresiones lógicas cuyos operandos son valores lógicos. Como resultado, este tipo de operadores genera un valor lógico. En seudocódigo se usan tres operadores lógicos:

Operador	Símbolo	Ejemplo	Resultado
Negación	`not`, `!`	`not True`	`False`
Conjunción	`and`, `&&`	`True and False`	`False`
Disyunción	`or`, `||`	`True or True`	`True`

El orden de precedencia de estos operadores es el siguiente:

- Primero se resuelven las operaciones en paréntesis
- Después se resuelven las negaciones (`not`)
- A continuación, se evalúa el operador `and`
- Por último, se evalúa el operador `or`

Cómo una expresión puede contener al mismo tiempo operadores de cualquiera de los tipos, se debe considerar la precedencia de todos los operadores: operaciones entre paréntesis, operadores aritméticos, operadores relacionales y al final los operadores lógicos. Esta precedencia se resume en la siguiente tabla.

Operador	Tipo
`( )`	Paréntesis
`^`	Potenciación
`*  /  DIV  MOD`	Multiplicativos
`+  -`	Aditivos
`<  <=  >  >=`	Relacionales
`==  !=`	Relacionales
`not`	Lógico
`and`	Lógico
`or`	Lógico
`=  +=  -=  *=  /=  %=`	Asignación

- *Operadores de Lectura y Escritura:* son los operadores que permiten a un programa interactuar con los usuarios que lo utilizan. El operador **Escribir** (también conocido como Mostrar o Imprimir) se utiliza para mostrar un mensaje en la pantalla del computador. Por otro lado, el operador **Leer** permite a los usuarios ingresar valores que serán almacenados en una variable. Veamos un ejemplo, el siguiente fragmento de seudocódigo pide al usuario su nombre, lo almacena en una variable llamada `nombre` y posteriormente lo saluda:

```
Inicio

  Cadena nombre

  Escribir("Cuál es tu nombre?")
  Leer(nombre)

  Escribir("Hola ", nombre)
  Escribir("¡Espero que tengas un fantástico día!")

Fin
```

En la instrucción **Escribir**, el texto entre las comillas dobles (" ") es el texto que se muestra al usuario, tal cual como lo ponemos en las comillas. Por otro lado, el texto por fuera de las comillas puede ser una variable o una operación encerrada en paréntesis y unida al texto usando una coma (,), la cual va por fuera de las comillas.

- Finalmente, está el *Operador de Asignación*, el cual permite guardar (o reemplazar) un valor de una variable. El operador de asignación se representa por el símbolo igual (=) e indica que en la variable que está a la izquierda del operador se almacena el valor que está a la derecha de éste.

Veamos un ejemplo, creemos tres variables enteras denominadas x, y e z. Los valores de x e y se los pedimos al usuario. Posteriormente en la variable z ASIGNAMOS el resultado de sumar x e y. Finalmente mostramos el resultado de la suma en la pantalla:

```
Inicio
 # Se declaran tres variables enteras
  Entero x, y, z

  # Se piden dos valores al usuario y se almacenan en las variables x e y
  Escribir ("Ingrese 2 números enteros: ")
  Leer (x, y)

  # Se suman los dos valores y el resultado se guarda en la variable z
  z = x + y

  # Se muestra el resultado al usuario
  Escribir ("El resultado de la suma es: ", z)

Fin
```

Esta es la instrucción que almacena en z el resultado de sumar x e y

Importante: Tenga presente que una instrucción que tenga la forma `x+y=z` NO ES CORRECTA en un algoritmo, esto porque el operador de asignación siempre espera que a la izquierda haya una sola variable y no una operación como es este caso.

3.4.4 EXPRESIONES

Una expresión se compone de operadores y operandos. Como vimos, los operadores nos permiten realizar operaciones aritméticas entre los operandos, los cuales pueden ser valores específicos, variables o constantes. En general, existen dos tipos de expresiones: las aritméticas y las booleanas. Las expresiones aritméticas son aquellas que tras su ejecución el resultado es un valor numérico, mientras que el resultado de las expresiones booleanas es un valor lógico.

Por otro lado, las expresiones aritmeticológicas son combinaciones de operadores aritméticos y lógicos que se utilizan para realizar cálculos matemáticos y tomar decisiones basadas en condiciones. Estas expresiones permiten a los programas realizar operaciones sobre datos y evaluar condiciones que determinan el flujo de ejecución del código.

Un ejemplo de una expresión aritmética surge, por ejemplo, cuando transformamos una fórmula matemática a una expresión algorítmica. Considere la ecuación matemática $x = \frac{-b+\sqrt{b^2-4ac}}{2a}$, la cual al ser transformada a una expresión algorítmica crea la siguiente expresión aritmética:

```
x = ((-1*b)+(b^2-4*a*c)^(1/2))/(2*a)
```

Por otro lado, un algoritmo que combina los dos tipos de expresiones (aritméticas y lógicas) es el siguiente:

```
Inicio

  Entero x=25, y=13, z
  Lógico p, q
  # Combinación de aritmética y lógica
  z = x + y DIV 2 * 3 - (23 MOD 20 + 5 ^ 2)
  p = x > z OR y + 20 < z
  q = p and x MOD 7 < y - 2

Fin
```

En este ejemplo, el cálculo de `z` es una expresión aritmética cuyo resultado es `25`, mientras que el cálculo de `p` y `q` cuyos valores es `False`, obedecen al resultado de expresiones booleanas.

4. EJERCICIO PROPUESTO

Considere el siguiente enunciado a partir del cual se ejemplificarán los elementos antes descritos.

Algoritmos secuenciales en seudocódigo

La empresa "***Huevos a Precio de Huevo***" comercializa 4 tipos de huevos: A, AA, AAA y Jumbo. El precio de cada tipo de huevo está definido sobre el valor de los huevos tipo AA. Haga un algoritmo que pida el valor de un huevo tipo AA y con base en este indique el precio de los otros tipos de huevo teniendo en cuenta las siguientes reglas:

- Los huevos tipo A cuestan la raíz cuadrada del precio de un huevo AA, multiplicado por 10, más el resultado de multiplicar 3 por el módulo entre el precio de un huevo AA y 100.
- El precio de los huevos AAA equivale al de un huevo AA más la raíz tercera del precio de un huevo tipo A, más un cuarto del valor de un huevo AA, sumado con el resultado de la división entera entre el valor del huevo AA con 10.
- El precio de un huevo Jumbo equivale al precio de un huevo AAA más, la raíz cuarta de un huevo AA elevado a la potencia 2. A todo se le suma un quinto del valor de un huevo A.

4.1 ANÁLISIS DEL PROBLEMA:

1. Identificamos el cliente y el usuario de la aplicación
   - **Cliente:** la empresa "Huevos a Precio de Huevo"
   - **Usuario:** los vendedores de la tienda

2. Identificamos los requerimientos funcionales:
   - **R1:** calcular el precio de los huevos tipo A, AAA y Jumbo

3. Identificamos las entidades del mundo del problema:
   - Huevo, hace referencia a un huevo del que se almacena su tipo (A, AA, AAA o Jumbo)

Al expandir el requerimiento en el formato de descripción de requerimientos, tenemos.

Identificador	**R1**
Nombre	**Calcular el precio de los huevos A, AAA y Jumbo**
Resumen	A partir del precio de un huevo AA el sistema debe calcular el precio de los huevos A, AAA y Jumbo. Los precios de cada huevo se calculan como se indica a continuación: ▪ huevoA = $\sqrt{huevoAA} * 10 + 3 * (huevoAA \text{ MOD } 100)$ ▪ huevoAAA = $huevoAA + \sqrt[3]{huevoA} + \frac{1}{4} huevoAA + (huevoAA \text{ DIV } 10)$ ▪ huevoJumbo = $huevoAAA + \sqrt[2]{huevoAA^5} + \frac{1}{5} huevoA$
Entradas	El precio de un huevo AA
Salidas o Resultados	Se muestra en la pantalla el valor de los huevos A, AA, AAA y Jumbo

4.2 ALGORITMO DE LA SOLUCIÓN

El paso siguiente en el diseño es el desarrollo de los algoritmos de la clase. En nuestro caso el algoritmo es sencillo y se presenta a continuación:

```
Inicio

  Real huevoA, huevoAA, huevoAAA, huevoJumbo

  # Pedimos los datos de entrada
  Escribir("Ingrese el precio de los huevos tipo AA: ")
  Leer(huevoAA)

  # Calculamos el precio de los huevos tipo A
  huevoA = huevoAA^(1/2) * 10 + 3 * (huevoAA MOD 100)

  # Calculamos el precio de los huevos tipo AAA
  huevoAAA = huevoAA + huevoA^(1/3) + (1/4)*huevoAA + (huevoAA DIV 10)

  # Calculamos el precio de los huevos tipo Jumbo
  huevoJumbo = huevoAAA+ huevoAA^(2/4) + (1/5)*huevoA

  # Mostramos los precios de los huevos
  Escribir("Los huevos tipo A cuestan ", huevoA, " pesos ")
  Escribir("Los huevos tipo AA cuestan ", huevoAA, " pesos ")
  Escribir("Los huevos tipo AAA cuestan ", huevoAAA, " pesos ")
  Escribir("Los huevos tipo Jumbo cuestan ", huevoJumbo, " pesos ")

Fin_clase
```

5. BIBLIOGRAFÍA

- Herrera, A., Ebratt, R., & Capacho, J. (2016). Diseño y construcción de algoritmos. Barranquilla: Universidad del Norte.
- Aguilar, L. (2020). Fundamentos de programación: algoritmos, estructuras de datos y objetos (5a.ed.). México: Mc Graw Hill.
- Thomas Mailund. Introduction to Computational Thinking: Problem Solving, Algorithms, Data Structures, and More, Apress, 2021.
- Steven S. Skiena. The Algorithm Design Manual. Third Edition, Springer, 2020.
- Samuel Tomi Klein. Basic Concepts in Algorithms, World Scientific Publishing, 2021.

Elaborado Por:	Carlos Andrés Mera Banguero
Versión:	1.1
Fecha	Agosto de 2024