Universidad Andrés Bello®
Conectar · Innovar · Liderar

FUNDAMENTOS DE PROGRAMACIÓN EN JAVA

Teoría de Conjuntos y Lógica Proposicional

En el área de las matemáticas, la teoría de conjuntos se entiende como el estudio de grupos de elementos u objetos especificados en tal forma que se puede afirmar si unobjeto pertenece o no a una agrupación. Comprender la teoría de conjuntos permiteutilizar los conjuntos como herramienta para analizar, clasificar y ordenar los conocimientos adquiridos.

Existen cuatro formas de definir los conjuntos. A continuación, se mencionará y dará ejemplos de cada una de ellas.

Extensión o enumeración

Sus elementos son encerrados entre llaves y separados por comas. Cada conjunto describe un listado de todos sus elementos. Un conjunto se determina por extensión cuando se enumeran o se nombran los elementos del conjunto. Cuando el conjunto es finito se escriben entre llaves, separados por comas. Cuando el conjunto es infinito se escriben entre llaves algunos elementos y se ponen puntos suspensivos.

Ejemplo:

```
A = {a,e,i,o,u}
B= {5,6,7,8,9}
```

Comprensión

Sus elementos se determinan a través de una condición que se establece entre llaves.Un conjunto se determina por comprensión enunciando la propiedad o cualidad que distingue a los elementos. Para tal fin se utiliza lo siguiente: {x/x cumple la propiedad}, que se lee: el conjunto de las x tal que x cumple la propiedad. A continuación, se indicala forma de expresar el ejemplo anterior como conjuntos definidos por comprensión.

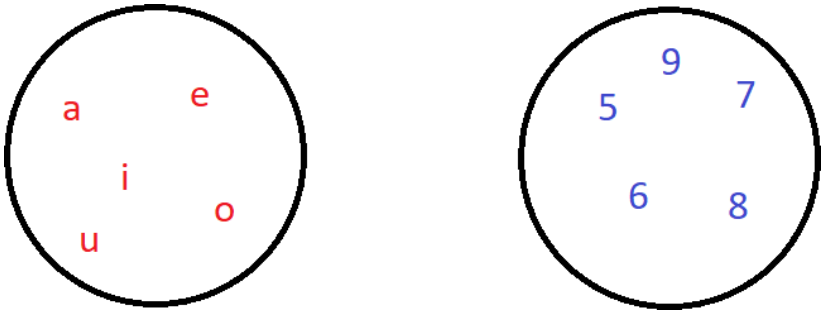
Ejemplo:

```
A = {x/x es una vocal}
B= {x e N / 5 <= x <= 9}
```

Diagramas de Venn

Regiones cerradas que nos permiten visualizar las relaciones entre los conjuntos. Consisten en figuras geométricas planas y cerradas; dentro de cada figura se ponen los elementos que le corresponden.

Ejemplo:



Descripción verbal

Se trata de un enunciado que describe una característica común a todos los elementosdel conjunto.

Ejemplo:

```
El conjunto A está compuesto por todas las letras del abecedario
clásico que son vocales.

El conjunto B está compuesto por todos los números naturales mayores
o iguales a 5, y menores o iguales a 9.
```

Lógica proposicional

La lógica es una disciplina filosófica que trata sobre la validez de los razonamientos; de acuerdo con su historia, la palabra lógica deriva del concepto griego λόγος, logos, que es traducido por *"pensamiento o razón"*. La lógica nace en el seno de la filosofía griega con el objetivo de ordenar las leyes del razonamiento; fue el filósofo Aristóteles(IV a.C.) su primer sistematizador, su tratado constituye un modelo de referencia hastael siglo XIX en el que se matematiza gracias al uso de un lenguaje simbólico.

La lógica proposicional es una rama de la lógica clásica que estudia las variables proposicionales o sentencias lógicas, sus posibles implicaciones, evaluaciones deverdad y en algunos casos su nivel absoluto de verdad. Algunos autores también la identifican con la lógica matemática o la lógica simbólica, ya que utiliza una serie de símbolos especiales que lo acercan al lenguaje matemático.

Las lógicas proposicionales carecen de cuantificadores o variables de individuo, pero tienen variables proposicionales (es decir, que se pueden interpretar comoproposiciones con un valor de verdad definido), de ahí el nombre proposicional. Los sistemas de lógica proposicional incluyen además conectivas lógicas, por lo que dentro de este tipo de lógica se puede analizar la inferencia lógica de proposiciones a partir de proposiciones, pero sin tener en cuenta la estructura interna de las proposiciones más simples.

Se ha visto anteriormente la notación simbólica empleada por la lógica proposicional.Ahora solo resta convertir expresiones del lenguaje natural en otras formalizadas.

Para ello se considera cada proposición (sujeto, predicado) como un todo. Se simbolizarán las proposiciones como letras a las que se denominarán "variablesproposicionales". Una letra cualquiera podrá simbolizar cualquier proposición que le sea asignada.

Posteriormente, se anota el significado de cada variable proposicional, y a continuación se enlazan mediante conectores. Por ejemplo, el enunciado Juan dedicasu tiempo libre a hacer deporte e ir al cine lo formalizaremos así: p Λ q, expresión en la que p = Juan dedica su tiempo libre a hacer deporte y q = Juan dedica su tiempo libre a ir al cine.

Al argumentar en el lenguaje cotidiano las personas no se ciñen estrictamente a los esquemas generales de razonamiento. Por ello, los razonamientos comunes requierenen ocasiones un esfuerzo de interpretación para determinar su estructura lógica. Es necesario entonces reconocer, más allá de la forma en que se expresan los enunciadosen el lenguaje natural, la relación lógica que se establece entre unas proposiciones y otras. Para lograr este propósito, existen recomendaciones para cada tipo de conector:

- **Conjunción Λ :** suele expresarse con la expresión "y", sin embargo, se puede nombrar de modos muy distintos ya sea por estilo o por denotar matices psicológicos: (comas, pero, sin embargo…), Se utilizará para añadir enunciados,por ejemplo:
    - Vino, cogió el dinero, se largó = p Λ q Λ r
    - Dice que es vegetariana, pero le encanta el pescado = p Λ q
    - Aunque dice que le gusta leer, no tiene ningún libro en casa = p Λ q

- **Disyunción V :** en el lenguaje natural puede ser débil o fuerte, inclusiva o exclusiva, esto es, puede indicar que cualquiera de las dos opciones vale o quevale sólo una de las dos. El símbolo V representa una disyunción inclusiva, de este tipo:
    - Se requiere una persona con conocimientos en inglés o en francés = p V q (¡no vamos a excluir a alguien por conocer los dos idiomas!)

- **Disyunción exclusiva W :** será verdadera cuando lo sea solo una de las proposiciones.
    - Estás en clase o en el patio = p W q (una u otra pero no las dos a la vez)

- **Condicional** □ : expresa oraciones tipo "si… entonces"; sin embargo esa misma implicación entre dos elementos, uno antecedente y otro consecuente, se puede

expresar de maneras distintas. Todas estas expresiones se simbolizarían como p → q:

- Si haces los ejercicios, (entonces) lo dominarás
- Siempre que queda con sus amigos vuelve de madrugada (Si queda consus amigos entonces vuelve de madrugada)
- Enséñame la entrada, entonces te acompañaré
- Haz clic sobre el icono del altavoz y sonará la canción (¡Ojo! Si presionasel icono, entonces sonará la canción)

- **Bicondicional ↔:** ambos extremos se condicionan mutuamente, lo emplearemos para expresiones del tipo "sólo si…" "si y sólo si…" o "cuando y solamente cuando". Si el condicional expresa una condición suficiente pero nonecesaria, el bicondicional expresa una condición suficiente y necesaria, la equivalencia entre dos proposiciones. Una condición es verdadera cuando sus dos extremos comparten el mismo valor de verdad y falsa en el caso contrario.
  - Sobrevivirá sólo si se somete a tratamiento quirúrgico: p ↔ q
  - Solo cuando pulso el interruptor se pone en funcionamiento el sistema eléctrico
- **Negador ¬ :** se usa tanto en negaciones explícitas como implícitas. Si en un argumento p representa ser simpático, su negación se podría presentar de estas dos formas: p = es simpático / ¬ p = no es simpático / ¬ p = es antipático.

- También se puede valorar la extensión de la negación en una oración, por ejemplo:
  - No luce el sol ni hace calor ¬ p Λ ¬ q
  - No es cierto que luzca el sol y haga calor ¬ (p Λ q)
  - No luce el sol, pero hace calor ¬ p Λ q

Con lo anterior, es posible formalizar el lenguaje natural simbolizando su estructura lógica mediante los símbolos del lenguaje lógico. Formalizar el lenguaje natural consiste en:

1. Determinar las proposiciones que intervienen para simbolizarlas con letras o variables proposicionales.
2. Establecer la estructura lógica existente entre las proposiciones parasimbolizarla mediante las conectivas lógicas.

A continuación, se indicarán algunos casos ejemplo.

Ejemplo 1: Formalice la siguiente oración "Si los elefantes volaran o supieran tocar elacordeón pensaría que estoy loco de remate y dejaría que me internaran en un psiquiátrico."

```
P : Los elefantes vuelan
Q : Los elefantes tocan el acordeón
R : Estoy loco de remate
S : Debo dejar que me internen en un psiquiátrico

Respuesta: (P v Q) → (R ^ S)
```

Ejemplo 2: Formalice la siguiente oración "Si no apruebas o no resuelves este problema, entonces es falso que hayas estudiado o domines la deducción lógica. Perono dominas la deducción lógica aunque has estudiado."

```
P : Apruebo
Q : Resuelvo el problema
R : He estudiado
```

```
S : Domino la deducción lógica

Respuesta: [(¬P v ¬Q) →¬(R v S)] ^ ¬S ^ R
```

Expresiones lógicas

Sirven para plantear condiciones o comparaciones y dan como resultado un valor booleano verdadero o falso, es decir, se cumple o no se cumple la condición. Se pueden clasificar en simples y complejas: las simples son las que usan operadores relacionales y las complejas las que usan operadores lógicos.

Las expresiones lógicas se forman combinando expresiones relacionales simples con operadores lógicos. El operador logico "Y" (AND o &&) y el operador lógico "O" (OR o ||) son ambos operadores binarios. Ambos operandos deben ser expresiones relacionales simples o expresiones lógicas que evalúen como verdadero o falso. El operador lógico de negación (NOT) requiere solo un operando simple.

La siguiente lista contiene descripciones de cada uno de los operadores lógicos y de algunas expresiones lógicas.

<<	Lenguaje natural	Ejemplo
Negación	No	No está lloviendo
Conjunción	Y	Está lloviendo y está nublado
Disyunción	O	Está lloviendo o está soleado
Condicional material	Si … entonces	Si está soleado, entonces es de día
Bicondicional	Si y sólo si	Está nublado si y sólo si hay nubes visibles
Disyunción opuesta	Ni … ni	Ni está soleado ni está nublado
Disyunción exclusiva	O bien … o bien	O bien está soleado, o bien está nublado

Ejemplo 3: Usando el operador lógico AND

"Una escuela aplica dos exámenes a sus aspirantes, por lo que cada uno de ellos obtiene dos

calificaciones denotadas como C1 y C2. El aspirante que obtenga calificaciones mayores que 80 en ambos exámenes es aceptado; en caso contrario esrechazado."

En este ejemplo se dan las condiciones siguientes:

```
Si (C1 >= 80) Y (C2 >= 80) entonces
    Escribir ("Aceptado")
SiNo
    Escribir ("Rechazado")
Fin Si
```

Se debe considerar que también se utilizan operadores relacionales. Por lo general cuando hay operadores lógicos, éstos van acompañados de operadores relacionales.

Ejemplo 4: usando el operador lógico OR
"Una escuela aplica dos exámenes a sus aspirantes, por lo que cada uno de ellos obtiene dos calificaciones denotadas como C1 y C2. El aspirante que obtenga una calificación mayor que 90 en cualquiera de los exámenes es aceptado; en casocontrario es rechazado."

En este caso se dan las condiciones siguientes:

```
Si (C1 >=90) O (C2 >=90) Entonces
    Escribir ("aceptado")
SiNo
    Escribir ("rechazado")
Fin Si
```

La instrucción equivale a OR ya que indica que puede ser en cualquiera de losexámenes, no necesariamente en los dos. En el ejemplo 1 la palabra "ambos" equivalía a seleccionar la instrucción AND.

Si la instrucción dijera "que obtenga una nota en cualquiera de los exámenes pero noen ambos", estaría indicando una instrucción XOR que es un tipo de OR, pero exclusivo. Es decir, no se puede considerar el caso en que tenga la misma nota en los dos exámenes, solo en uno de los dos.

Ejemplo 5: usando el operador lógico NOT

"Una escuela está procesando las notas de fin de año de sus alumnos. Cada promediodefine si el alumno aprobó o no el año escolar; además, se sabe de cada alumno si eshonorario (más de tres años en la escuela) o no. Si el alumno aprueba el año y es honorario, recibe el diploma de excelencia honoraria; si aprueba pero no es honorario, recibe el diploma de alumno destacado, y si no aprueba se le envía una carta de motivación para que al otro año le vaya mejor."

En este caso se dan las condiciones siguientes:

```
aprueba = Verdadero
honorario = No(Verdadero)

Si (aprueba = Verdadero) Y (honorario = Verdadero) Entonces
    Escribir ("Recibe diploma de excelencia honoraria")
SiNo
    Si (aprueba = Verdadero) Y (honorario = Falso) Entonces
        Escribir ("Recibe diploma de alumno destacado")
    SiNo
        Escribir ("Recibe carta de motivación")
    Fin Si
Fin Si
```

Es necesario recordar que el operador "NO" es unario; esto significa que solo permitirá transformar un valor de verdad al caso contrario. Por lo mismo, en vez de usar la opción "No(Verdadero)", se pudo haber utilizado la opción "Falso", obteniendo el mismo resultado.

En la clase anterior se analizaron las tablas de verdad de la conjunción, la disyunción y la negación. En esta ocasión indicaremos las tablas de verdad restantes, mencionadas en los párrafos anteriores.

Condicional material (implicancia)		
X	**Y**	**X → Y**
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Falso
Falso	Verdadero	Verdadero
Falso	Falso	Verdadero

Bicondicional		
X	**Y**	**X ↔ Y**
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Falso
Falso	Verdadero	Falso
Falso	Falso	Verdadero

Disyunción opuesta		
X	**Y**	**X ↓ Y**
Verdadero	Verdadero	Falso
Verdadero	Falso	Falso
Falso	Verdadero	Falso
Falso	Falso	Verdadero

Disyunción exclusiva		
X	**Y**	**X <-/-> Y**
Verdadero	Verdadero	Falso
Verdadero	Falso	Verdadero
Falso	Verdadero	Verdadero
Falso	Falso	Falso

Es los lenguajes de programación comunes, generalmente no existen operadores creados que permitan reemplazar la funcionalidad aplicable en estas funciones. Sin embargo, siempre es posible crear funciones o rutinas que entreguen un valor de verdad como resultado de la evaluación de dos casos específicos. Además, no se debeolvidar que las evaluaciones de los casos anteriores se derivan o generan a partir desde los operadores lógicos de conjunción, disyunción y negación.

Evaluación de expresiones lógicas

Toda expresión regresa un valor. Si hay más de un operador, se evalúan primero los operadores de mayor precedencia y, en caso de empate, se aplica la regla de asociatividad (indica que, cuando existen tres o más cifras en estas operaciones, el resultado no depende de la manera en la que se agrupan los términos). Para evaluar una expresión no hay que hacer nada del otro mundo, puesto que se deben aplicar las operaciones aritméticas o lógicas respectivas, y reemplazar dicha sentencia por el valor de verdad correspondiente. Después se evalúa la proposición siguiente, hasta obtener un valor de verdad de la sentencia completa.

Hay tres reglas de prioridad a seguir para evaluar una expresión:

- Primero, los paréntesis (si tiene)
- Después, seguir el orden de prioridad de operadores
- Por último, si aparecen dos o más operadores iguales, se evalúan de izquierda a derecha.

Es importante recordar que se puede construir una expresión válida por medio de:

1. Una sola constante o variable, la cual puede estar precedida por un signo + ó – .
2. Una secuencia de términos (constantes, variables, funciones) separados por operadores.

Debe considerarse, por cierto, que toda variable utilizada en una expresión debe tenerun valor almacenado para que la expresión, al ser evaluada, dé como resultado un valor. Además, cualquier constante o variable puede ser reemplazada por una llamadaa una función.

Para los ejemplos siguientes, de forma similar a como sucede en las expresiones matemáticas, una expresión lógica se evalúa de acuerdo a la precedencia de operadores.

Ejemplo 6: ¿Qué resultado tendrá la siguiente expresión?

```
Resultado = No(((5>4) O (3<6)) Y (8==5))
Escribir(Resultado)
```

Para evaluar lo anterior, se debe hacer siempre desde adentro hacia afuera. Se comienza

evaluando si la disyunción entre 5>4 y 3<6, lo cual arroja un valor Verdadero. Se sabe además que 8 no es igual a 5, por tanto la conjunción entre esto último y cualquier cosa será Falso. Finalmente, la negación de algo Falso seráVerdadero, lo cual corresponde al valor de la variable Resultado al final del proceso.

Estructuras de control repetitivas (mientras, repetir, para)

Las estructuras de control repetitivas están diseñadas para que una expresión sea evaluada muchas veces, sin tener que escribirlas cada vez.

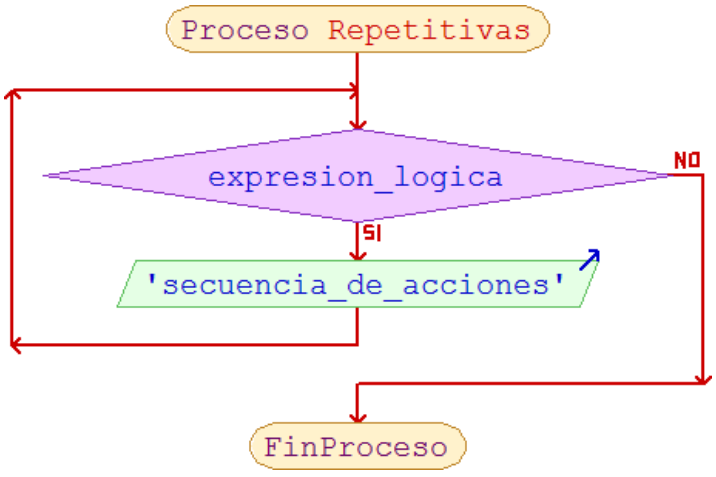
Estas estructuras también conocidas como bucles o ciclos, permiten la ejecución de una secuencia de sentencias o instrucciones una cierta cantidad de veces, esto dependerá del tipo de solución que se deba implementar.

Existen 3 estructuras repetitivas:

- Mientras
- Repetir … Hasta
- Para

Las tres instrucciones tienen el mismo fin, y difieren únicamente en su sintaxis, siendoposible sustituir una solución en la que se utiliza "Mientras", por una en la que se utiliza "Repetir … Hasta" o "Para".

En los diagramas de flujo, un ciclo se representa de la siguiente manera:



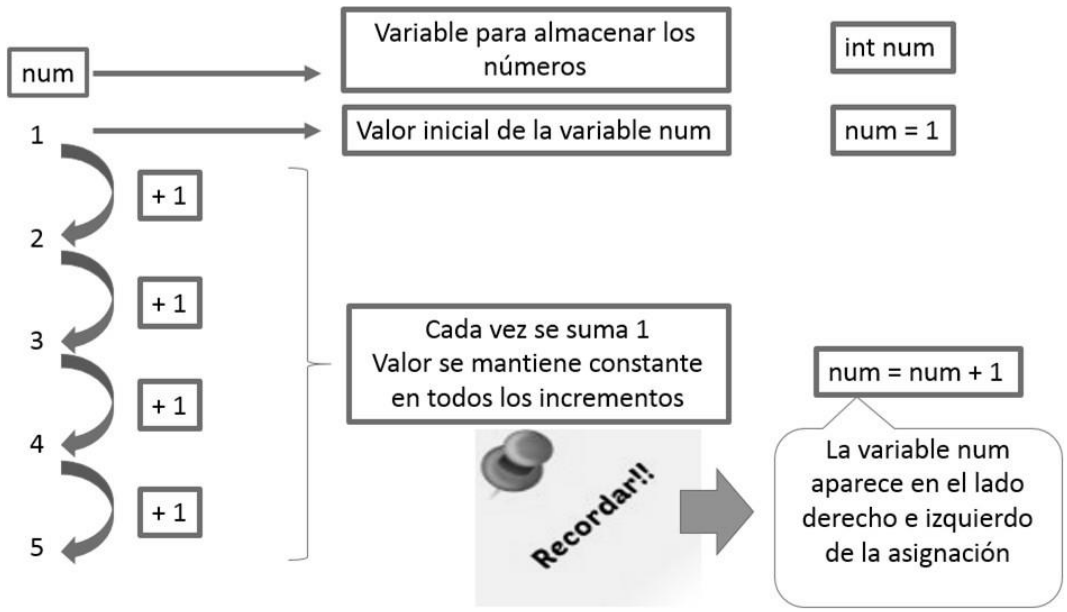
En la imagen anterior se puede observar que las líneas de flujo indican el orden a seguir y, según el valor de la condición, continuará ejecutándose el mismo conjunto de instrucciones o saldrá del ciclo. Entre las tres instrucciones hay pequeñas variaciones de representación gráfica que serán detalladas en la explicación de uso decada una de ellas.

Las estructuras de control repetitivas utilizan dos tipos de variables: Contadores y Acumuladores.

Contadores

Un contador es una variable de tipo entero, que incrementa o decrementa su valor deforma constante y requiere ser inicializada generalmente en 0 o 1, aunque en realidaddepende del problema que se está resolviendo. Como su nombre lo indica se utilizan en la mayoría de veces para contar el número de ocasiones que se ejecuta una acción,o para contar el número de veces que se cumple una condición (expresión relacional/lógica).

Por ejemplo, si se desea sumar los números del 1 al 5, se necesitará una variable quegenere esos números, es decir que empiece en 1 y llegue hasta el 5.



La variable que cumple el rol de contador, aparece tanto a la izquierda como a la derecha, por la propiedad destructiva de la asignación; así tomará el valor anterior, leadicionará o reducirá el valor constante y asignará el nuevo valor.

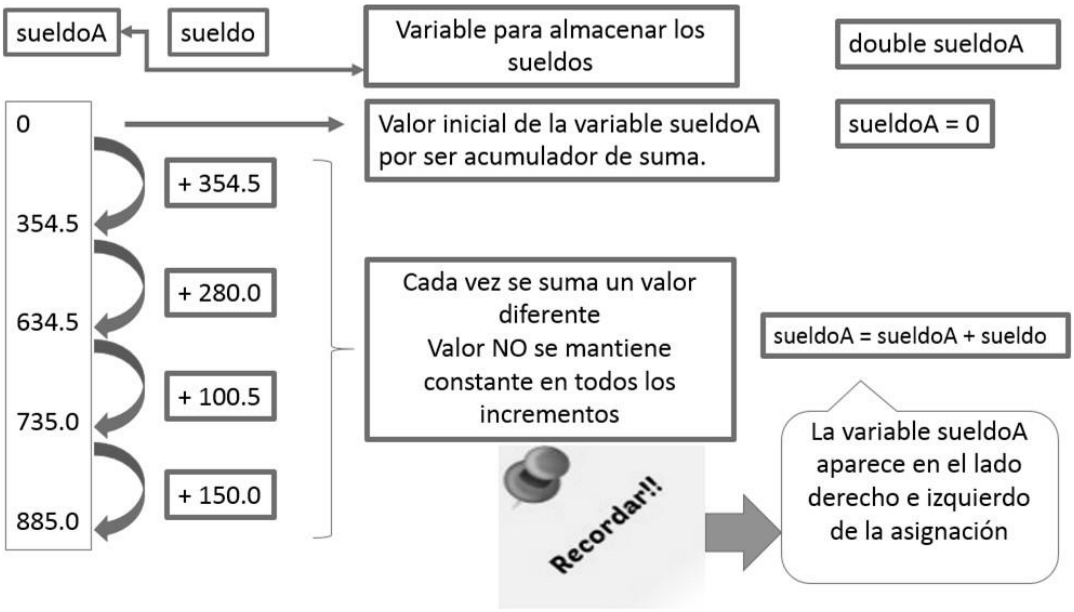
Acumuladores

Un acumulador es una variable numérica, que incrementa o decrementa su valor de forma no constante y requiere ser inicializada. Como su nombre lo indica se utilizan para acumular valores en una sola variable, ya sea de suma o producto. Por lo tanto existen dos modos de inicialización:

- Para Suma: Inicializar en 0
- Para Producto: Inicializar en 1

Esto con el objetivo de no alterar los valores de las respectivas operaciones.

Por ejemplo, si se desea conocer el acumulado de los pagos realizados a un grupo de empleados, se necesitará una variable que vaya sumando los sueldos de cada empleado, junto con una variable que permita calcular el acumulado.



Para identificar qué instrucciones van dentro de una instrucción repetitiva debe preguntarse lo siguiente:

- De las acciones requeridas para la solución de este problema ¿cuáles serealizan VARIAS VECES?
- Todas las respuestas a esta pregunta corresponden a las instrucciones que vandentro de la sentencia repetitiva

Bucle "Mientras"

Esta estructura repite el conjunto de instrucciones que se encuentra dentro del ciclo,mientras la condición evaluada sea verdadera. Si la condición es falsa, entonces se termina el ciclo. En este caso, primero se evalúa la condición y luego se ejecutan las acciones que se encuentran dentro del ciclo.

La sintaxis de esta estructura es la siguiente:

```
Mientras (condición) Hacer
    Instrucciones
Fin Mientras
```

Si la condición de término es verdadera, entonces las instrucciones que están dentro del ciclo se ejecutan. Si la condición es falsa, el ciclo termina y el flujo continúa con elalgoritmo. En este tipo de ciclo, las instrucciones se ejecutan como mínimo cero veces.

Ejemplo 1: Realice un algoritmo que permita mostrar por pantalla la suma de los 20 primeros números naturales.

```
acumulador <- 0
contador <- 1
Mientras contador <= 20 Hacer
      acumulador <- acumulador + contador
      contador <- contador + 1
Fin Mientras

Escribir "El resultado final es: ", acumulador
```

Bucle "Repetir"

Esta estructura repite un conjunto de instrucciones hasta que la condición lógica resulta ser verdadera. Por lo tanto, las instrucciones se volverán a ejecutar sólo si la condición es falsa.

La sintaxis de esta estructura es la siguiente:

```
Repetir
    Instrucciones
Hasta Que (condición)
```

Si la condición de término es falsa, entonces el flujo vuelve al principio del ciclo y éstese continúa ejecutando. Si la condición es verdadera, el ciclo termina y el flujocontinúa con el algoritmo. En este tipo de ciclo, las instrucciones se ejecutan como mínimo una vez.

Ejemplo 2: Realice un algoritmo que permita mostrar por pantalla la suma de los 20 primeros números naturales. Esta vez utilice una estructura "Repetir … Hasta"

```
acumulador <- 0
contador <- 1
Repetir
      acumulador <- acumulador + contador
      contador <- contador + 1
Hasta Que contador > 20

Escribir "El resultado final es: ", acumulador
```

Bucle "Para"

Este tipo de ciclo repite las instrucciones una cantidad determinada de veces. Esta cantidad está dada por un valor de finalización de la variable de control del ciclo y el valor del paso o incremento.

La sintáxis de la estructura repetitiva “Para” es la siguiente:

```
Para (variable = vinicial Hasta vfinal Con Paso) Hacer
    Instrucciones
Fin Para
```

Una variable de control se inicializa con un determinado valor inicial, luego va aumentando de acuerdo al paso cada vez que se ejecuta el ciclo. La ejecución finalizacuando la variable de control contiene el valor final que se ha determinado. En este tipo de ciclo, las instrucciones se ejecutan como mínimo cero veces.

Ejemplo 3: Realice un algoritmo que permita mostrar por pantalla la suma de los 20 primeros números naturales. Esta vez utilice una estructura “Repetir … Hasta”

```
acumulador <- 0

Para i<-1 Hasta 20 Con Paso 1 Hacer
      acumulador <- acumulador + i
Fin Para

Escribir "El resultado final es: ", acumulador
```

Patrones Comunes

Un **patrón** se puede definir como una solución aplicable a un problema que ocurre a menudo. Si bien existen muchos problemas diferentes que se pueden resolver, existen situaciones que son comunes a cada problema desarrollable bajo un algoritmo. Dichode otro modo, existen formas de abordar un problema que se pueden replicar en diversos contextos, haciendo las adaptaciones necesarias según lo solicite el problema.

Estos patrones son los siguientes:

Sumar cosas

Ejemplo: Escriba un programa que reciba como entrada un número entero. El programa debe mostrar el resultado de la suma de los números al cuadrado desde el1 hasta el valor ingresado.

```
Escribir "Ingrese n: "
Leer n
suma <- 0
cont <- 1

Mientras cont <= n Hacer
      d <- cont^2
      suma <- suma + d
      cont <- cont + 1
Fin Mientras

Escribir "La suma de los numeros al cuadrado es:", suma
```

Multiplicar cosas

Ejemplo: Escriba una programa que calcule el factorial de un número n ingresadacomo entrada: 3!=1·2·3

```
Escribir "Ingrese n: "
Leer n

Si n < 0 Entonces
      Escribir "El factorial está definido sólo para números
naturales mayores o igual que 0."
SiNo
      prod = 1
      cont = 1
FinSi

Mientras cont <= n Hacer
      prod <- prod * cont
      cont <- cont + 1
FinMientras

Escribir "El factorial de ", n, " es: ", prod
```

Contar cosas

Ejemplo: Escriba un programa que solicite el ingreso de n números, luego entregue la cantidad de números pares ingresados.

```
Escribir "Ingrese n: "
Leer n

pares <- 0
cont <- 0

Mientras cont < n Hacer
      Escribir "Ingrese numero: "
      Leer num

      Si num MOD 2 = 0 Entonces
            pares <- pares + 1
      FinSi

      Escribir "pares =", pares
      cont <- cont + 1
FinMientras

Escribir "La cantidad de pares ingresados es:", pares
```

Encontrar el máximo

Ejemplo: Escribir un programa que solicite n números y luego muestre el númeromayor que haya sido ingresado.

Opción 1: usando un número muy pequeño para comparar.

```
Escribir "Ingrese n: "
Leer n

i <- 1
//Se establece un numero muy pequeño
para comparar nummayor = -999999

Mientras i <= n Hacer
    Escribir "Ingrese
    numero: "Leer num

    Si nummayor < num
        Entonces
        nummayor <-
        num
    FinSi

    Escribir "mayor temp
    =", nummayor i <- i + 1
FinMientras

Escribir "El numero mayor es:", nummayor
```

Opción 2: sin usar número muy pequeño para comparar.

```
Escribir "Ingrese n: "
Leer n

i <- 1

Mientras i <= n Hacer
      Escribir "Ingrese numero: "
      Leer num

      Si i = 1 Entonces
            nummayor = num
      SiNo
            Si nummayor < num Entonces
                  nummayor = num
            FinSi
      FinSi
      i <- i + 1
FinMientras

Escribir "El numero mayor es ", nummayor
```

Encontrar el mínimo

Ejemplo: ¿Cómo cambia el patrón anterior si ahora se quiere encontrar el mínimo?

```
Escribir "Ingrese n: "
Leer n

i <- 1
nummenor <- 9999999

Mientras i <= n Hacer
      Escribir "Ingrese numero: "
      Leer num

      Si nummenor > num Entonces
            nummenor <- num
      FinSi
      i <- i + 1
FinMientras

Escribir "El numero menor es:", nummenor
```

Generar pares de cosas

Ejemplo: Escribir un programa que muestre todas las combinaciones posibles al lanzar2 dados.

```
n <- 6
m <- 6
i <- 0
Mientras i < n Hacer
      j <- 0
      i <- i + 1
      Mientras j < m Hacer
            j <- j + 1
            Escribir i , " , ", j
      FinMientras
FinMientras
```

Funciones

En programación, una función es un conjunto de líneas de código que realizan una tarea específica y puede retornar un valor. Una función puede ser invocada o llamadadesde otras partes del programa tantas veces como se desee. Las funciones pueden tomar parámetros que modifiquen su funcionamiento. Las funciones son utilizadas para descomponer grandes problemas en tareas simples y para implementar operaciones que son comúnmente utilizadas durante un programa y de esta manera reducir la cantidad de código. Cuando una función es invocada se le pasa el control a la misma, una vez que esta finalizó con su tarea el control es devuelto al punto desdeel cual la función fue llamada.

La sintaxis de una función es la siguiente:

```
SubAlgoritmo variable_retorno <- Nombre ( Argumentos)
    //Instrucciones
FinSubAlgoritmo

Algoritmo ejemplo
    //Instrucciones y llamada a SubAlgoritmo
FinAlgoritmo
```

Los componentes indicados en la sintaxis anterior se explican de la siguiente manera:

- **variable_retorno:** si el subproceso o función, calcula y devuelve un valor, se debe colocar el nombre de la variable que se utilizará para almacenar ese valor; si el subproceso o función no devuelve nada se puede eliminar la variable con su flecha de asignación.
- **Nombre:** es el nombre que recibirá el subproceso, función o subalgoritmo.

- **Argumentos:** los argumentos son variables que requiere la función, utilizandocomas para separarlos; si el subproceso o función no requiere argumentos puede dejarse en blanco, y de manera opcional omitir los paréntesis.

Por ejemplo, si debemos calcular el promedio de un alumno, podemos crear una función llamada Promedio, la cual estará encargada de recibir las 5 calificaciones del semestre, sumarlas y dividirlas para conseguir el promedio, el cual será retornado por la función.
Ejemplo: Creación de una función que permita obtener el promedio de 5 calificaciones

```
SubProceso promedio <- CalcularPromedio
      Definir n1, n2, n3, n4, n5 Como Entero;
      Definir promedio Como Real;
      Escribir "Ingrese nota 1: ";
      Leer n1;
      Escribir "Ingrese nota 2: ";
      Leer n2;
      Escribir "Ingrese nota 3: ";
      Leer n3;
      Escribir "Ingrese nota 4: ";
      Leer n4;
      Escribir "Ingrese nota 5: ";
      Leer n5;
      promedio<-((n1+n2+n3+n4+n5)/5);
FinSubProceso

Proceso Ejemplo1
      Definir prom Como Real;
      Definir x1, x2, x3, x4, x5 Como Entero;
      prom<-CalcularPromedio;
      Escribir "El promedio obtenido es: ", prom;
FinProceso
```

El ejemplo anterior corresponde a un subproceso que no recibe parámetros, siendo necesario solicitar el ingreso de cada nota de forma manual. Una alternativa a este modelo es entregar todas las notas como parte del subproceso. El ejemplo siguiente da cuenta de este caso.

Ejemplo: Creación de una función que permita obtener el promedio de 5calificaciones, con recepción de parámetros

```
Funcion prom <- CalcularPromedio ( num1, num2, num3, num4, num5 )
      Definir prom Como Real;
      prom<-((num1+num2+num3+num4+num5)/5);
FinFuncion

Proceso Ejemplo2
      Definir promedio Como Real;
```

```
      Definir x1, x2, x3, x4, x5 Como Entero;

      Escribir "Calculo de promedio";
      Escribir "Ingrese nota 1: ";
      Leer x1;
      Escribir "Ingrese nota 2: ";
      Leer x2;
      Escribir "Ingrese nota 3: ";
      Leer x3;
      Escribir "Ingrese nota 4: ";
      Leer x4;
      Escribir "Ingrese nota 5: ";
      Leer x5;

      promedio<-CalcularPromedio(x1,x2,x3,x4,x5);
      Escribir "El promedio obtenido es: ", promedio;
FinProceso
```

Una tercera opción para definir y acceder a funciones, es a través de sub algoritmos. En el ejemplo siguiente se usará la misma idea de ejemplos anteriores, pero sin entregar un valor de retorno.

Ejemplo: Creación de una función que permita obtener el promedio de 5calificaciones, con recepción de parámetros

```
SubAlgoritmo CalcularPromedio (num1, num2, num3, num4, num5)
      Definir prom Como Real;
      prom<-((num1+num2+num3+num4+num5)/5);
      Escribir "El promedio calculado es: ", prom;
FinSubAlgoritmo

Proceso Ejemplo3
      Definir x1, x2, x3, x4, x5 Como Entero;
      Escribir "Calculo de promedio";
      Escribir "Ingrese nota 1: ";
      Leer x1;
      Escribir "Ingrese nota 2: ";
      Leer x2;
      Escribir "Ingrese nota 3: ";
      Leer x3;
      Escribir "Ingrese nota 4: ";
      Leer x4;
      Escribir "Ingrese nota 5: ";
      Leer x5;

      CalcularPromedio(x1,x2,x3,x4,x5);
FinProceso
```

Los subprocesos sin parámetros se llaman desde el proceso principal simplemente por su nombre sin más argumentos; se pueden abrir y cerrarparéntesis, pero esto es

opcional. En cambio, si el subproceso contiene parámetros, estos si debenespecificarse.

Ejemplo: Desarrolle un algoritmo en pseudocódigo que retorne todos los númerosprimos entre el 1 y el 30.

```
SubAlgoritmo resultado <- Primo ( num )
    Definir cantidadDivisores, cont Como Entero;
    Definir resultado Como Logico;
    cantidadDivisores <- 0;
    Para cont <- 1 Hasta num Hacer
        Si num % cont = 0 Entonces
            cantidadDivisores <- cantidadDivisores + 1;
        FinSi
    FinPara
    Si cantidadDivisores <= 2 Entonces
        resultado <- verdadero;
    Sino
        resultado <- falso;
    FinSi
FinSubAlgoritmo

Algoritmo PrimosDel1Al30
    Definir n Como Entero;
    Para n <- 1 Hasta 30 Hacer
        Si Primo(n) Entonces
            Escribir n;
        FinSi
    FinPara
FinAlgoritmo
```

Recursividad

Las funciones recursivas son aquellas que se llaman a sí mismas durante su propia ejecución. Ellas funcionan de forma similar a las iteraciones, pero es necesario de planificar el momento en que dejan de llamarse a sí mismas o se tendrá una función recursiva infinita.

Estas funciones se estilan utilizar para dividir una tarea en sub-tareas más simples deforma que sea más fácil abordar el problema y solucionarlo.

El siguiente ejemplo da cuenta de una función recursiva para el cálculo del factorial de un número entero.

Ejemplo: Desarrolle una función recursiva que calcule el factorial de un número.

```
SubProceso fact <- Factorial ( N )
      Definir fact Como Entero;
      Si N = 0 O N = 1 Entonces
            fact <- 1;
      SiNo
            fact <- N * Factorial(N-1);
      FinSi
FinSubProceso

Proceso Ejemplo5
      Definir num Como Entero;
      Escribir "Ingrese un número: ";
      Leer num;
      Escribir "El factorial de ",num," es ",Factorial(num);
FinProceso
```

El ejemplo anterior corresponde a una función recursiva con retorno. También es posible usar este mismo concepto sin necesidad de entregar un valor como resultado.Esto se refleja en el ejemplo siguiente.

Ejemplo: Desarrolle una función recursiva que simule una cuenta regresiva.

```
Funcion cuentaRegresiva (contador)
      Si contador > 0 Entonces
            Escribir contador;
            cuentaRegresiva(contador-1);
      SiNo
            Escribir "Lanzamiento!";
      FinSi
FinFuncion

Proceso Ejemplo6
      cuentaRegresiva(10);
FinProceso
```

Anexo: Referencias

1.- Lógica proposicional
Referencia: https://www.monografias.com/trabajos-pdf5/la-logica-proposicional/la- logica-proposicional.shtml

2.- Expresiones lógicas
Referencia:
https://ocw.unican.es/pluginfile.php/266/course/section/171/capitulo4.pdf

3.- Propiedad asociativa
Referencia: https://definicion.de/propiedad-asociativa/#:~:text=La%20propiedad%20asociativa%20aparece%20en,que%20se%20agrupan%20los%20t%C3%A9rminos.

4.- Ejercicios resueltos algoritmos – Estructuras repetitivas
Referencia: https://www.aprenderaprogramar.pro/2017/07/ejercicios-algoritmos.html

5.- Estructura repetitiva: Mientras
Referencia:
https://www.aprendeaprogramar.com/cursos/verApartado.php?id=2006

6.- Estructura repetitiva: Repetir … Hasta
Referencia:
https://www.aprendeaprogramar.com/cursos/verApartado.php?id=2007

7.- Estructura repetitiva: Para
Referencia:
https://www.aprendeaprogramar.com/cursos/verApartado.php?id=2008

8.- Funciones / Subprocesos en PSeInt
Referencia: https://victomanolo.wordpress.com/funciones-subprocesos-en-pseint/#:~:text=Las%20funciones%20tambi%C3%A9n%20llamadas%20Subproceso,que%20se%20pueden%20invocar%20(ejecutar)

9.- Creación de funciones y procedimientos
Referencia:
https://www.aprendeaprogramar.com/cursos/verApartado.php?id=2012

10.- Funciones recursivas
Referencia: https://plataforma.josedomingo.org/pledin/cursos/programacion/curso/u32/