Asignatura:

Diseño Estructurado de Algoritmos

Especialidad:

Ingeniería En Sistemas Computacionales.

Diseño Estructurado de Algoritmos

UNIDAD 1. CONCEPTOS BÁSICOS.

- **Introducción.**
    - **De los problemas a los programas.**
    - **Breves prácticas de programación.**

- **Definición de lenguaje.**
- **Definición de algoritmo.**
- **Algoritmos cotidianos.**
- **Definición de lenguajes algorítmicos.**
- **Historia y aplicación de los lenguajes algorítmicos.**

UNIDAD 11. METODOLOGÍA PARA LA SOLUCIÓN DE PROBLEMAS POR MEDIO DE COMPUTADORAS.

- **Definición de problema.**
- **Análisis de los datos.**
- **Diseño de la solución.**
- **Codificación.**
- **Prueba y depuración.**
- **Documentación.**
- **Mantenimiento.**
- **Introducción**

Los problemas que se plantean en la vida diaria suelen ser resueltos mediante el uso de la capacidad intelectual y la habilidad manual del ser humano. La utilización de la computadora en la resolución de problemas aporta grandes ventajas, como son la rapidez de ejecución y la confiabilidad de los resultados obtenidos.

En la actualidad los sistemas de computadoras consisten en un enorme conjunto de elementos de circuitos(hardware)y programación (software), que se han diseñado para proporcionar a la computación un ambiente productivo y agradable.

Muchos de los problemas acarrean complicados cálculos, así como la utilización de grandes cantidades de datos; esto nos causa muchos problemas ya que el riesgo de equivocarse es muy grande, y también con la utilización de grandes cantidades de datos el trabajo se convierte en pesado y rutinario. Mediante la computadora se eliminan estos problemas, pues su capacidad se basa en la rapidez, la precisión y memoria.

Sin embargo la computadora no puede hacer todo por si sola. Es preciso que se le describa con detalle y en su

lenguaje, todos los pasos que tiene que realizar para la resolución del problema.

Esta descripción es lo que se conoce como programa de computadora, este dirigirá el funcionamiento de la máquina en la resolución del problema.

La primer decisión que hay que tomar cuando se crea un programa, es contestar a las preguntas:

¿Qué es lo que el programa se supone que va a hacer?, ¿cuál es el problema que se va a resolver?,¿Qué tarea va a realizar nuestro programa?.

Cuanto más se detalle la descripción, más fácil será lograr resultados.

- ***Definición de lenguaje***

Un lenguaje de programación es aquel que se utiliza para escribir programas de computadora que puedan ser entendidos por ellas.

Estos lenguajes se clasifican en tres grandes categorías:

- **Lenguaje Máquina.**
- **Lenguaje de Bajo Nivel (Ensamblador)**
- **Lenguaje de Alto Nivel.**

Se puede definir también como cualquier lenguaje artificial que puede utilizarse para definir una secuencia de instrucciones para su procesamiento por un ordenador o computadora.

- **Definición de algoritmo**

Un algoritmo es una secuencia finita de instrucciones; cada una de estas instrucciones tiene un significado preciso y se puede ejecutar con una cantidad finita de esfuerzo en un tiempo finito.

Un algoritmo se define como un método que se realiza paso a paso para la solución de un problema que termina en un número finito de pasos.

−Las características fundamentales que debe cumplir un algoritmo son:

- Debe ser ***preciso e indicar el orden***. Diseño del algoritmo que describe la secuencia ordenada de pasos, sin ambigüedades, que conducen a la solución de un problema dado *(Análisis del problema y desarrollo del algoritmo).*
- Debe ser ***definido***. Si se sigue un algoritmo dos veces se debe obtener el mismo resultado cada vez. Expresar el algoritmo como un programa en un lenguaje de programación adecuado *(Fase de codificación).*
- Debe ser ***finito.*** Si se sigue un algoritmo, se debe terminar en algún momento; osea debe tener un número finito de pasos. Ejecución y validación del programa por la computadora.

La definición de un Algoritmo debe describir tres partes:

- **Algoritmos cotidianos**

Se refiere a todos aquellos algoritmos que nos ayudan a resolver problemas diarios, y que hacemos casi sin darnos cuenta que estamos siguiendo una metodología para resolverlos.

−Algunos ejemplos son:

Diseñar un algoritmo para cambiar una llanta de un coche:

1.− Inicio.

2.− Traer gato.

3.− Levantar el coche con el gato.

4.− Aflojar tornillos de las llantas.

5.− Sacar los tornillos de las llantas.

6.− Quitar la llanta.

7.− Poner la llanta de repuesto.

8.− Poner los tornillos.

9.− Apretar tornillos.

10.− Bajar el gato.

11.− Fin.

Un cliente ejecuta un pedido a una fábrica; la fábrica examina en su banco de datos la ficha del cliente, si el cliente es solvente entonces la empresa acepta el pedido, en caso contrarios rechazará el pedido.

Inicio

Leer pedido

Examinar ficha del cliente

Si el cliente es solvente aceptar el pedido; en caso contrario rechazar el pedido, FIN.

- **Definición de lenguajes algorítmicos**

Los algoritmos pueden describirse utilizando muchos lenguajes. Cada lenguaje permite describir los pasos con mayor o menor detalle.

La clasificación de los lenguajes algorítmicos es la siguiente:

- **Lenguaje Natural.**
- **Lenguaje de Diagrama de Flujo.**
- **Lenguaje Natural de Programación.**
- **Lenguaje de Programación de Algoritmos.**

Lenguaje Natural.− Es aquel que describe los

pasos a seguir utilizando un vocabulario cotidiano. Se le conoce como lenguaje jerga cuando se utiliza en

términos especializados de una determinada ciencia, profesión o grupo.

Lenguaje de Diagrama de Flujo.− Es aquel que se

vale de diversos símbolos para representar las ideas o acciones a desarrollar. Es útil para organizar las acciones o pasos de un algoritmo pero requiere de etapas posteriores para implementarse en un sistema de cómputo.

Lenguaje Natural de Programación.− *Son aquellos*

que están orientados a la solución de problemas que se definen de una manera precisa. Generalmente son aplicados para la elaboración de fórmulas o métodos científicos.

Tienen las siguientes características:

- **Evita la ambigüedad (algo confuso que se puede interpretar de varias maneras).**
- **Son precisos y bien definidos.**
- **Utilizan términos familiares al sentido común.**
- **Elimina instrucciones innecesarias.**

Lenguaje de programación.− Es un conjunto de

palabras, símbolos y reglas sintáticas mediante los cuales puede indicarse a la computadora los pasos a seguir para resolver un problema.

Los principales tipos de lenguajes utilizados son tres:

- **Lenguaje Máquina.**
- **Lenguaje de Bajo Nivel (Ensamblador).**
- **Lenguajes de Alto nivel.**

Lenguaje Máquina.− Este lenguaje es el que entiende directamente a la computadora, utiliza el alfabeto binario que consta de dos únicos símbolos 0 y 1, denominados bits(abreviatura inglesa de números binarios). El lenguaje máquina fue el primer lenguaje utilizado para programar computadoras, pero éste fue sustituido por su dificultad y complicación.

Lenguaje de bajo nivel (Ensamblador).− En este lenguaje cada instrucción equivale a una instrucción en lenguaje máquina, utilizando para su escritura palabras nemotécnicas en lugar de cadenas de bits.

Las instrucciones en lenguaje ensamblador son instrucciones conocidas como nemotécnicas, por ejemplo, nemotécnicos típicos de operaciones aritméticas son:(en inglés)ADD, SUB, DIV, etc; (en español) SUM, RES, DIV, etc.

Lenguaje de Alto nivel.− Estos lenguajes son los más utilizados por los programadores. Un programa escrito en un lenguaje de alto nivel es independiente de la máquina (las instrucciones no dependen del diseño del hardware o de una computadora en particular), por lo que estos programas son portables o transportables. Los programas escritos en lenguaje de alto nivel pueden ser ejecutados con poca o ninguna modificación en diferentes tipos de computadoras.

- **Historia y aplicación de los lenguajes algorítmicos**

Los lenguajes permiten expresar los programas o el conjunto de instrucciones que el operador humano desea

que la computadora ejecute. Los lenguajes de las primeras computadoras como la ENAC y la EDSAC se componían en el lenguaje real de las máquinas mismas; de esta manera limitaba drásticamente la utilidad de las mismas.

Los primeros lenguajes de programación se conocieron como lenguajes *ensambladores*. En los lenguajes ensambladores se define un código especial llamado *mnemónico* para cada una de las operaciones de la máquina y se introduce una notación especial para especificar el dato con el cual debe realizarse la operación.

En los años 60's aparecieron los primeros lenguajes de propósito general como COBOL, PASCAL, C, C++, etc., pero el desarrollo de nuevas tecnologías, tanto en la arquitectura de computadoras, como en lenguajes de programación continúa a paso acelerado. Los lenguajes de programación actuales son conocidos como *lenguajes visuales*, como por ejemplo: Visual Fox, Visual Basic, Visual C, etc.

2.1 Definición de problema

La definición del problema está dada en sí por el enunciado del problema, el cuál debe ser claro y complejo. Es importante que conozcamos exactamente *que se desea obtener al final del proceso.*

2.2 Análisis de los datos

Para que un problema se pueda definir con precisión se requiere que las especificaciones de entrada y salida sean descritas con detalle ya que esto es un requisito para lograr una solución.

Una vez que el problema se ha definido y comprendido, deben analizarse los siguientes aspectos:

- **Los resultados esperados.**
- **Los datos de entrada disponibles.**

Para facilitar esta etapa debemos de ponernos en lugar de la computadora deduciendo los elementos que se necesitarán para alcanzar el resultado.

2.3 Diseño de la solución

Las computadoras solo pueden solucionar problemas siempre y cuando se le proporcionen los pasos sucesivos que tiene que realizar, esto se refiere a un algoritmo que resuelva correctamente el problema.

−Esta etapa incluye la descripción del algoritmo resultante en un lenguaje natural de diagrama de flujo o natural de programación.

2.4 Codificación

La codificación se refiere a la obtención de un programa definitivo que pueda ser comprensible para la máquina. Incluye una etapa que se reconoce como compilación.

Programa Fuente.− Está escrito en un lenguaje de programación y es entendido por el programador.

Programa Ejecutable.− Está en lenguaje máquina y es entendido por la máquina.

2.5 Prueba y depuración

Cuando se obtiene el programa ejecutable, este se somete a una prueba con el fin de determinar si resuelve de forma satisfactoria o no el problema planteado.

Son muchas las pruebas que se le aplican al programa y por lo general dependen del tipo de problema que se está resolviendo. Generalmente se inicia la prueba de un programa introduciendo datos válidos, inválidos e incongruentes y se observa como reacciona en cada ocasión.

*La **depuración*** consiste en que si existen errores en el programa localizarlos y corregirlos.

Si no existen errores, se puede entender al proceso de depuración como la etapa de refinamiento, en la que se ajustan detalles para optimizar el programa.

2.6 Documentación

En esta etapa se procede a la utilización para resolver problemas del tipo que dio origen al diseño del programa.

Esta utilización no podrá ser siempre supervisada por el programador, por tal motivo debe crearse un manual o una guía de operación de los pasos de la utilización del programa.

2.5 Mantenimiento

Esta etapa se refiere a las actualizaciones que deban aplicase al programa cuando las circunstancias así lo requieran.

Cualquier cambio en el programa se debe reflejar en su documentación.

Los programas deben ser mantenidos mientras dure su ciclo de vida.

AQUÍ EMPÍEZAN LAS UNIDADES 6−7−8−9 DE LA MISMA MATERIA

****INDICE****

UNIDAD 6. ESTRUCTURAS ALGORÍTMICAS.

- **Secuenciales.**
- **Condicionales.**

UNIDAD 7. ARREGLOS.

- **Vectores.**
- **Matrices.**

UNIDAD 8. MANEJO DE CADENA DE CARACTERES.

- **Definición.**
- **Función.**
- **Manipulación.**

UNIDAD 9. MANEJO DE MÓDULOS.

- **Concepto y características de un modulo.**
- **Clasificación de los módulos.**
- **Operación de módulos y sus parámetros.**

9.4 Criterios de modularización.

6.1 ESTRUCTURA SECUENCIAL

Es aquélla en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso. La estructura secuencial tiene una entrada y una salida. Su representación gráfica es la siguiente:

Estructura secuencial:

Acción 1
Acción 2
Acción 3
.....

DIAGRAMA N−S DE UNA ESTRUCTURA SECUENCIAL:

Acción 1
Acción 2
...
Acción n

PSEUDOCÓDIGO DE UNA ESTRUCTURA SECUENCIAL:

Inicio

:

:

acciones

:

:

fin

Ejemplo:

Calcular el salario neto de un trabajador en función del número de horas trabajadas, precio de la hora de trabajo y considerando unos descuentos fijos al sueldo bruto en concepto de impuestos (20 por 100).

****Pseudocódigo****

- Inicio
- {cálculo salario neto}
- leer nombre, horas, precio_hora
- salario_bruto horas * precio

- impuestos 0.20 * salario_bruto
- salario_neto salario_bruto_impuestos
- escribir nombre, salario_bruto, salario_neto_bruto, salario_neto
- Fin

6.2 CONDICIONALES

La especificación formal de algoritmos tiene realmente utilidad cuando el algoritmo requiere una descripción más complicada que una lista sencilla de instrucciones. Este es el caso cuando existen un número de posibles alternativas resultantes de la evaluación de una determinada condición.

Las estructuras selectivas se utilizan para tomar decisiones lógicas; de ahí que se suelan denominar también estructuras de decisión o alternativas.

En las estructuras selectivas se evalúa una condición y en función del resultado la misma se realiza una opción u otra. Las condiciones se especifican usando expresiones lógicas. La representación de una estructura selectiva se hace con palabras en pseudocódigo (if, then, else o bien en español si, entonces, sino), con una figura geométrica en forma de rombo o bien con un triángulo en el interior de una caja rectangular.

Las estructuras selectivas o alternativas pueden ser:

- *Simples*
- *Doble*
- *Múltiples*

ALTERNATIVA SIMPLE (SI−ENTONCES/IF−THEN).

La estructura alternativa simple si−entonces (en inglés if−then o bien IF−THEN) ejecuta una determinada acción cuando se cumple una determinada condición. La selección si−entonces evalúa la condición y . .

Si la condición es *verdadera*, entonces ejecuta la acción S1 (o acciones caso de ser S1 una acción compuesta y constar de varias acciones).

Si la condición es *falsa*, entonces no hacer nada.

****PSEUDOCÓDIGO EN ESPAÑOL****

Si <condición> Entonces

<acción S1>

Fin_si

PSEUDOCÓDIGO EN INGLÉS

If <condición> then

<acción S1>

end_if

ALTERNATIVA DOBLE (SI−ENTONCES−SI_NO / IF − THEN − ELSE).

La estructura anterior es muy limitada y normalmente se necesitará una estructura que permita elegir entre dos opciones o alternativas posibles, en función del cumplimiento o no de una determinada condición.

Si la condición C es verdadera, se ejecuta la acción S1 y, si es falsa, se ejecuta la acción S2.

****PSEUDOCODIGO EN ESPAÑOL****

Si < condición > entonces

< acción S1 >

si_no

<acción S2>

fin_si

PSEUDOCODIGO EN INGLES.

If < condición > then

< acción S1 >

else

< acción S2 >

endif

ALTERNATIVAS MÚLTIPLES (SEGÚN _ SEA, CASO DE / CASE).

Cuando existen más de dos elecciones (alternativas) posibles, es cuando se presenta el caso de alternativas múltiples. Si el número de alternativas es grande puede plantear serios problemas de escritura del algoritmo y naturalmente de legibilidad.

La estructura de decisión múltiple evaluará una expresión que podrá tomar *n* valores distintos 1,2,3,4,..n . Según que elija uno de estos valores en la condición, se realizará una de las *n* acciones, o lo que es igual, el flujo del algoritmo seguirá un determinado camino entre los *n* posibles.

****PSEUDOCÓDIGO****

En inglés la estructura de decisión múltiple se representa:

Case expresión of

[e1]: acción S1

[e2]: acción S2

:

[en]: acción Sn

else

acción Sx

end_case

Ejemplo:

Se desea diseñar un algoritmo que escriba los nombres de los días de la semana en función del valor de una variable DIA introducida por teclado.

Los días de la semana son 7; por consiguiente, el rango de valores de DIA será 1..7, y caso de que DIA tome un valor fuera de este rango se deberá producir un mensaje de error advirtiendo la situación anómala.

Inicio

Leer DIA

Según_sea DIA hacer

1:escribir('Lunes')

2: escribir('Martes')

3: escribir('Miércole')

4: escribir('Jueves')

5: escribir('Viernes')

6: escribir('Sabado')

7: escribir('Domingo')

else

escribir('Error')

fin_según

fin

ESTRUCTURAS REPETITIVAS.

Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan *Bucles* y se denomina *Iteración* al hecho de repetir la ejecución de una secuencia de acciones. Entre las estructuras repetitivas se encuentran:

- **Mientras (while)**
- **Repetir (repeat)**

- **Desde (for)**

ESTRUCTURA MIENTRAS (WHILE).

La estructura repetitiva while, es aquélla en que el cuerpo del bucle se repite mientras se cumple una determinada condición.

Pseudocódigo en español Pseudocódigo en inglés

Mientras condición **hacer while** condición **do**

Acción S1 <Acciones>

Acción S2 :

: **End_while**

acción Sn

Fin_mientras

Ejemplo:

Contar los números enteros positivos introducidos por teclado. Se consideran dos variables enteras NUMERO y CONTADOR (contará el número de enteros positivos). Se supone que se leen números positivos y se detiene el bucle cuando se lee un número negativo o cero.

PSEUDOCÓDIGO

Inicio

contador 0

Leer (numero)

Mientras numero > 0 hacer

contador contador+1

Leer (numero)

Fin_Mientras

Escribir('El número de enteros positivos es : ', contador)

Fin

7.1 VECTORES

Son aquéllos de una sola dimensión, por lo que también son llamados arreglos Unidimensionales.

Ejemplo :

Un vector de una dimensión llamado CALIF, que consta de n elementos.

calif(1)	*calif(2)*		*calif(n−2)*	*calif(n−1)*	*calif(n)*

El subíndice o índice de un elemento (1, 2 n) designa su posición en la ordenación del vector. Otras posibles notaciones del vector son:

a1, a2,,an En matemáticas y algunos lenguajes(BASIC)

A(1), A(2),A(n)

A[1], A[2],A[n] En programación (Pascal)

Los vectores se almacenan en memoria central de la computadora en un orden adyacente. Así, un vector de 50 elementos denominado NUMEROS se representa gráficamente por 50 posiciones de memoria sucesivas.

Memoria

NUMEROS(1) Dirección x

NUMEROS(2) Dirección x+1

NUMEROS(3) Dirección x+2

:

:

NUMEROS(50) Dirección x+49

Cada elemento de un vector se puede procesar como si fuese una variable simple al ocupar una posición de memoria. Así :

NUMEROS(25) 75 (almacena el valor 75 en la posición 25a del vector NUMEROS y la instrucción de salida: Escribir NUMEROS(25).

DECLARACIÓN

Nom _arreglo = arreglo[liminf..limsup] de tipo

donde :

nom_arreglo : nombre válido del arreglo.

liminf..limsup : límites inferior y superior del rango del arreglo.

tipo : tipo de datos de los elementos del arreglo: entero, real, carácter.

NOMBRES: arreglo [1..10] de carácter

Significa que NOMBRES es un arreglo (array) unidimensional de 10 elementos (1 a 10) de tipo carácter.

ASIGNACIÓN

NOMBRES(8) `Ana'

Asigna el valor `Ana' al elemento 8 del vector NOMBRES

LECTURA / ESCRITURA DE DATOS

La Lectura / escritura de datos en un arreglo u operaciones de entrada / salida normalmente se realizan con estructuras repetitivas, aunque puede también hacerse con estructuras selectivas. Las instrucciones simples de lectura / escritura se representarán como:

leer A Lectura del vector A

escribir A Escritura del vector A

leer V(5) Leer el elemento V(5) del vector V

ACCESO SECUENCIAL AL VECTOR (RECORRIDO)

Se puede acceder a los elementos de un vector para introducir datos (escribir) en el o bien para visualizar su contenido (leer). Estas operaciones se realizan utilizando estructuras repetitivas, cuyas variables de control (por ejemplo I) se utilizan como subíndices del vector (por ejemplo, X(I). El incremento del contador del bucle producirá el tratamiento sucesivo de los elementos del vector.

Ejemplo :

Lectura de 15 valores enteros de un vector denominado TOTAL.

TOTAL= array [1..15] de entero

desde i1 hasta 15 hacer

leer TOTAL(i)

fin _desde

Si se cambian los limite inferior y superior, por ejemplo, 5 y 12, el bucle de lectura sería:

desde i 5 hasta 12 hacer

leer TOTAL(i)

fin _ desde

La salida o escritura de vectores se representa de un modo similar.

desde i1 hasta 15 hacer

escribir TOTAL(i)

fin _ desde

Visualiza todo el vector completo (un elemento en cada línea independiente).

ACTUALIZACIÓN DE UN VECTOR.

Puede constar de tres operaciones más elementales:

AÑADIR elementos (añade un nuevo elemento al final del vector).

Un arreglo A se ha dimensionado a 6 elementos, pero solo se han asignado 4 valores a los elementos A(1), A(2), A(3), A(4), se podrán añadir dos elementos más con una simple acción de asignación.

A(5) 15

A(6) 9

INSERTAR **elementos (introduce un elemento en el interior de un vector).**

Ejemplo :

Se tiene un arreglo NOM de 6 elementos de nombres de personas, en orden alfabético y se desea insertar un nuevo nombre.

{Calcular la posición ocupada por el elemento a insertar} P

{Inicializar contador de inserciones} i n.

mientras i >= P hacer

{transferir el elemento actual hacia abajo, a la posición i+1}

NOM(i+1) NOM(i)

{decrementar contador}

i i−1

fin _ mientras

{Insertar el elemento en la posición P}

NOM(P) `nuevo elemento'

{Actualizar el contador de elementos del vector}

n n+1

fin

BORRAR elementos (Elimina elementos de un vector).

ALGORITMO DE BORRADO

Inicio

{se utilizará una variable auxiliar AUX, que contendrá el valor del elemento que se desea borrar}

AUX NOM(i)

desde ij hasta N−1 hacer

{llevar elemento j+1 hacia arriba}

NOM(i) NOM(i+1)

fin_desde

{actualizar contador de elementos}

{ahora tendrá un elemento menos, N−1}

N N−1

Fin

REFERENCIA A UN ELEMENTO DE ARREGLO

Variable de arreglo [subíndice]

Ejemplo :

x x[3]9

escribir (x[2])

?	**9**

1 2 3

Ejercicio :

Se desea la lectura y desplegado de 5 nombres. Resuelva el problema por cada uno de los siguientes criterios:

a) Lectura y desplegado alternados.

b) Todas las lecturas, todos los desplegados.

a) variables :

string : nom

entero : x

Inicio

Para x1 hasta 5 hacer

escribir(`Nombre ? `)

leer(nom)

escribir(`La persona número', x, ` se llama : `,nom)

Fin

Nota : No se utilizaron arreglos porque no se requería de almacenamiento múltiple.

b) Variables:

nom : Arreglo[1..5] de string

x : Entero

Inicio

para x 1 **hasta** 5 **hacer**

escribir(`Dame el nombre número',x,' ?')

leer(nom[x])

para x 1 **hasta** 5 **hacer**

escribir(`La persona número `, x,' se llama : `,nom[x])

Fin

ORDENACIÓN DE ARREGLOS

Existen diversos métodos para ordenar los elementos de un arreglo. El más conocido de ellos (no el mejor) es el Método de la Burbuja.

El método consiste en hacer un recorrido por el arreglo comparando parejas de elementos; si estos no están en el orden deseado, se procede a intercambiarlos.

Al finalizar el recorrido se verifica la cantidad de intercambios, si esta es 0 se asume que el arreglo está ordenado; en caso contrario se inicia nuevamente el recorrido.

Las parejas de elementos que se comparan deben ser contiguos (elemento1 y elemento2, elemento2 y elemento3, etc). El número total de comparaciones es n−1 (donde n es la cantidad de elementos).

Ejemplo :

Se requiere la ordenación de una lista con 5 valores enteros previamente introducidos.

Variables :

LISTA : arreglo[1..5] de entero

x, Aux. : entero

cambio : boleano

Inicio

Para x 1 hasta 5 hacer

escribir(`Dame el valor',x,' :')

leer(LISTA[x])

repetir

cambio falso

para x 1 hasta 4 hacer

si LISTA[x] > LISTA[x+1] entonces

Aux. LISTA[x]

LISTA[x] LISTA[x+1]

LISTA[x+1] Aux.

cambio verdadero

fin_si_entonces

hasta cambio = falso

escribir(`Lista ordenada')

para x 1 hasta 5 hacer

escribir(`Elemento número',x,' es',LISTA[x])

Fin

7.2 MATRICES

Se puede considerar como un vector de vectores. Es, por consiguiente, un conjunto de elementos, todos del mismo tipo, en el cual el orden de los componentes es significativo y en el que se necesitan especificar dos subíndices para poder identificar a cada elemento del arreglo.

También se les llama arreglos Bidimensionales, ya que una tabla será utilizada cuando se requiere de establecer relaciones por renglones y columnas entre datos de un tipo común.

1 2 3 4 . . . J . . . N

B(I,J)

Se considera que este arreglo tiene dos dimensiones (una dimensión por cada subíndice) y necesita un valor para cada subíndice, y poder identificar un elemento individual. En notación estándar, normalmente el primer subíndice se refiere a la fila del arreglo, mientras que el segundo subíndice se refiere a la columna del arreglo. Es decir, B(I,J), es el elemento de −b que ocupa la Ia y la Ja columna como se muestra en la **figura anterior.**

Un ejemplo típico de un arreglo Bidimensional es un tablero de ajedrez. Se puede representar cada posición o casilla del tablero mediante un arreglo, en el que cada elemento es una casilla y en el que su valor será un código representativo de cada figura del juego.

Ejemplo:

Se desea registrar las edades de 4 grupos de personas, cada uno de ellos con 5 elementos. Los datos deberán ser mostrados en forma posterior.

variables:

edad: arreglo(1..4,1..5) de entero

g,p: enteros

Inicio

para g 1 hasta 4 hacer

escribir(`Grupo : `,g)

para p 1 hasta 5 hacer

escribir(`Edad de la persona `,p)

leer(edad[g,p])

para g1 hasta 4 hacer

escribir(`Grupo `,g)

para p 1 hasta 5 hacer

escribir(´persona `,p,' Tiene',edad[g,p],´años´)

Fin

Ejemplo:

Se tienen 4 fábricas cada una de ellas con 6 empleados. Se desea registrar los salarios y mostrarlos posteriormente. En forma alternada deberá leerse también el nombre de cada empleado. Durante el desplegado de los gastos se indicará el nombre y salario del empleado mejor pagado de cada fábrica así como el total de nómina (de cada fábrica).

Variables:

Nom: arreglo[1..4,1..6] de string

salario: arreglo[1..4,1..6] de entero

fab, emp, total, cont, lug, zuc: entero

Inicio

para fab 1 hasta 4 hacer

para emp1 hasta 6 hacer

escribir(`Empleado : `,emp)

escribir(´Nombre del empleado :')

leer(nom[fab,emp])

escribir(`Salarios del empleado : `)

leer(sal[fab,emp])

para fab 1 hasta 4 hacer

total 0

cont sal[fab,1]

escribir(`Fábrica', fab)

para emp 1 hasta 6 hacer

escribir(`El empleado `,emp,' se llama`,nom[fab,emp],' y gana `,salario[fab,emp])

total + salario[fab,emp]

si salario[fab,emp] >= cont entonces

cont salario[fab,emp]

lug fab

zuc emp

fin_Entonces

escribir(`El mejor pagado es `,nom[lug,zuc],´y gana `,salario[lug,zuc])

escribir(`El total es : `,total)

Fin

8.1 DEFINICIÓN

Una *cadena* **(string)** de caracteres es un conjunto de caracteres (incluido el blanco), que se almacenan en un área contigua de la memoria. Pueden ser entradas o salidas a/desde una computadora.

La longitud de una cadena es el número de caracteres que contiene. La cadena que no contiene ningún carácter se le denomina cadena vacía o nula. y su longitud es cero; no se debe confundir con una cadena compuesta de solo espacios en blanco.

La representación de las cadenas suele ser con comillas simples o dobles, las comillas actúan como separadores, algunos ejemplos:

`12 de Octubre de 1492'

`Hola, como estás?'

La cadena puede contener entre sus separadores, cualquier carácter válido del código aceptado por el lenguaje y la computadora; el blanco es uno de los caracteres más utilizado.

Una **Subcadena** es una cadena de caracteres que ha sido extraída de otra de mayor longitud.

'12 de' es una subcadena de '12 de Octubre de 1492'

`como estás' es una subcadena de `Hola, como estás?'

`XI´ es una subcadena de `MEXICO ES GRANDE'

DATOS TIPO CARÁCTER.

Anteriormente se analizaron los diferentes tipos de datos y entre ellos existían el dato tipo carácter (char) que se incorpora en diferentes lenguajes de programación, bien con este nombre, o bien como datos tipo cadena.

CONSTANTES.

Una constante tipo carácter es un conjunto de caracteres válidos encerrados entre comillas, para evitar confundirlos con nombres de variables, operadores, enteros, etc.

VARIABLES.

Una variable de cadena o tipo carácter es una variable cuyo valor es una cadena de caracteres. Las variables de tipo carácter se deben declarar en el algoritmo y según el lenguaje tendrán una notación u otra.

var **NOMBRE, DIRECCION, PAIS: carácter**

CADENAS DE LONGITUD FIJA.

Se consideran vectores de la longitud declarada, con blancos a izquierda o derecha si la cadena no tiene la longitud declarada. Así el ejemplo siguiente:

E	**S**	**T**	**A**	**C**	**A**	**S**	**A**	**E**	**S**	**U**	**N**	**A**	**R**	**U**	**I**	**N**	**A**	///			