UNIVERSIDAD DE MEDELLIN

Unidad de Organización Curricular (UOC)
TECNOLÓGICAS

MÓDULO # 3

FUNDAMENTOS DE PROGRAMACIÓN

2009

PRESENTACIÓN DE LA ASIGNATURA

Esta asignatura tiene como objetivo “Desarrollar destrezas lógico-abstractas, para conceptualizar e implementar algoritmos computacionales en la solución matemática de problemas”.

A través de los contenidos de cada módulo usted podrá cumplir el objetivo y desarrollar las competencias de esta asignatura. Este documento de estudio, también llamado OVA (Objeto Virtual de Aprendizaje) tiene 3 módulos, organizados así:

En el módulo 1, usted comienza con una breve introducción al diseño de algoritmos, lo cual le servirá para poder identificar los operadores, expresiones, tipos de datos, variables y constantes de un algoritmo. Luego puede comenzar a reconocer las siguientes estructuras: de programación, del diseño de un algoritmo en pseudocódigo, secuencial y selectiva.

En el módulo 2, continúa con la estructura repetitiva y puede identificar los contadores y acumuladores. Con estas bases estará preparado para escribir algoritmos que requieran ciclos sencillos o ciclos anidados, estos pueden ser: “para” y “repita hasta”.

En el módulo 3, utilizará todo su aprendizaje anterior para crear arreglos, matrices y realizar programación modular.

Este documento de estudio le presenta el desarrollo temático y todos los recursos digitales necesarios para el estudio de la asignatura.

PRESENTACIÓN DEL MÓDULO

Las variables permiten almacenar la información en posiciones de memoria a las cuales se puede acceder por medio de su nombre. En muchos casos prácticos, se requiere procesar una colección de valores que tienen alguna propiedad en común, por ejemplo una lista de edades, los precios de una colección de artículos, etc. En estas situaciones, almacenar cada dato en una variable simple puede ser complejo; es más adecuado en cambio, utilizar alguna estructura en la cual se aloje la lista de valores. Por otra parte, para conseguir una buena comprensión de los problemas algorítmicos complejos se requiere subdividirlos en problemas más sencillos; la idea es efectuar subdivisiones hasta alcanzar un nivel de problemas suficientemente simples y fáciles de tratar; en esto consiste la programación modular.

Variables allow us store information at individual memory positions which can be accessed by variable names. Often it is required to deal with data collections such as an age list. In these situations, simple variables must be replaced by data structures. On the other hand, modular programming works breaking down a problem into several sub-problems until these become simple enough to be solved directly.

TEMARIO

DESARROLLO TEMÁTICO

1. ARREGLOS

Los arreglos son importantes estructuras de información que permiten almacenar varios datos del mismo tipo en un sólo grupo; en un arreglo se puede hacer referencia a cualquiera de sus elementos de manera sencilla, sin invadir los demás datos. Los arreglos pueden ser de una dimensión, a los cuales llamaremos *vectores*, o de dos dimensiones, a los cuales llamaremos *matrices*.

Concepto de arreglo

Un arreglo es una agrupación de datos del mismo tipo; también podemos decir que es una estructura de datos que almacena información de igual conformación. Cada valor almacenado está asociado de manera unívoca a una posición relativa dentro del grupo, llamada índice.

Arreglos unidimensionales o vectores

Son aquellos cuyo grupo de elementos se disponen con un mismo concepto de clasificación; es decir, los elementos distribuidos en una fila o una columna. La siguiente figura muestra un vector con siete posiciones y el contenido de cada posición.

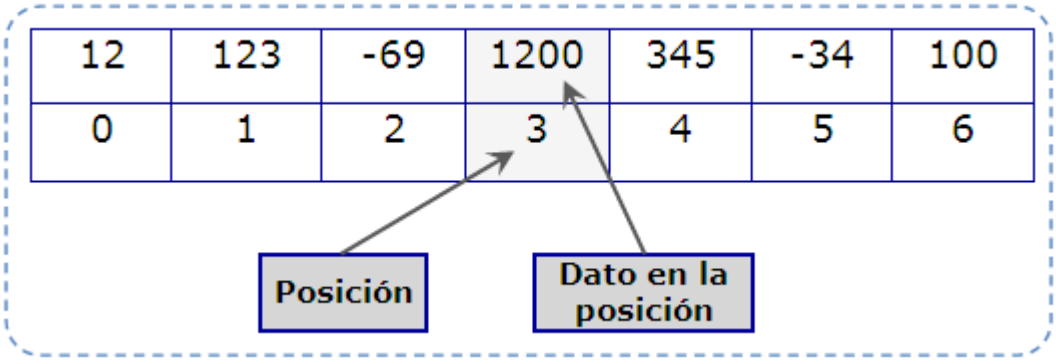


Figura 1: Representación gráfica de un vector

El valor 1200 está en la posición 3. Para hacer referencia a los elementos del arreglo se utiliza la siguiente notación: **V (3)=1200**, indicando que el elemento en la posición 3 del vector **V** es 1200.

1.1. OPERACIONES CON ARREGLOS

1.1.1. Operaciones sobre los elementos de un arreglo

Las operaciones que se pueden utilizar son: asignación, lectura (guardar) y escritura (imprimir o mostrar).

- **Asignación**

  Para asignar un valor a un elemento de un vector usamos la instrucción. **V (3) ←12**, asigna al vector V, el valor 12 en la posición 3.

- **Lectura**

  Lea (V (2)), indica que el valor que ingrese el usuario se almacenará en la posición 2 del vector V.

- **Escritura**

  Imprima (V (6)), Imprime, o muestra en pantalla el valor del vector V en la posición 6.

1.1.2. Operaciones sobre el arreglo completo

Para realizar una operación sobre un arreglo completo se utilizan los ciclos, teniendo cuidado con la variable de control del ciclo empleado. Cuando se conoce el tamaño del arreglo es conveniente utilizar un ciclo PARA, que recorra una por una las posiciones del arreglo, desde la primera hasta la última. Las operaciones que se pueden realizar sobre un vector son recorrido, búsqueda, ordenamiento, inserción y eliminación.

Ejemplo: Diseñar un algoritmo que ingrese 100 números enteros, los guarde en un vector y finalmente los imprima.

```
INICIO
  Variables
  Entero Arreglo (100), I, N
  Para I←1 Hasta 100
     Imprima ("Ingrese valor en posición" I)
    Lea (Arreglo (I))
  Fin Para
  Para I←1 Hasta 100
     Imprima (Arreglo (I))
   Fin Para
FIN
```

Como se ve en el algoritmo, el primer ciclo permite el ingreso de los cien valores del arreglo y en el segundo ciclo se imprimen.

En este documento se presentan otros ejemplos que debe estudiar para un mejor aprendizaje del tema. (Ver ejemplos en el *anexo 1* al final de este documento)

RECUERDE

Para almacenar los datos en un arreglo unidimensional es necesario tener establecido el tamaño del vector previamente, el cual se especifica en la declaración del vector.

Cantidad de Primos:

Se pretende mostrar los números primos que se encuentran en un vector de 10 posiciones. Para este propósito es necesario recorrer el arreglo y verificar si cada entrada es número primo, en caso afirmativo se muestra el número.

```
Entero I, Suma←0, cont←0, V (10)
Para I←1 Hasta 10
    Para J←1 hasta V (I)
      Si (V (I) Mod J = 0) Entonces
        Cont←Cont+1
      Fin SI
    Fin Para
    Si (Cont  = 2) Entonces
       Imprima ("El número"  V (I) "es primo")
    Fin Si
    Cont←0
 Fin Para
```

Observe que para cada posición del vector se verifica si es un número primo contando sus divisores, en caso de tener únicamente dos divisores se comprueba que es primo. Observe la utilización de los dos ciclos PARA anidados, el primero recorre el vector y el segundo verifica el número de divisores de cada uno de los elementos del vector.

RECUERDE

Utilizamos el signo "=" para hacer comparaciones y el signo "←" para hacer asignaciones. Las instrucciones Muestre e Imprima se usan indistintamente para sacar mensaje por pantalla.

Ejemplo: Diseñar un algoritmo que permita llenar un vector con los diez primeros números primos. (Ver ejemplo en el *anexo 2* al final de este documento)

1.2. ORDENAMIENTO DE UN VECTOR

Podemos ordenar un vector de datos numéricos de mayor a menor (descendentemente) o de menor a mayor (ascendentemente). El siguiente algoritmo muestra la forma de efectuar este ordenamiento. El algoritmo revisa cada elemento del vector que va a ser ordenado y lo compara con el elemento siguiente, intercambiándolos de posición si están en el orden equivocado.

Es necesario revisar varias veces todo el vector hasta que no se necesiten más intercambios, lo cual significa que está ordenado. Dado que sólo usa comparaciones para operar elementos, se considera un algoritmo de comparación. Suponemos que el vector tiene **N** posiciones y está lleno uncialmente.

```
Entero I, J, Vector (N), AUX
    Para I←1 hasta N-2
      Para J←I+1 hasta N-1
        Si (V (I)>V (J)) entonces
            AUX←V (I)
            V (I) ←V (J)
            V (J) ←AUX
        Fin SI
      Fin Para
    Fin Para
```

Tradicionalmente este algoritmo se conoce como ordenamiento BURBUJA (http://lc.fie.umich.mx/~calderon/programacion/notas/Burbuja.html), este algoritmo obtiene su nombre de la forma con la que suben por el vector los elementos durante los intercambios, como si fueran pequeñas "burbujas". También es conocido como el método del intercambio directo.

Hagamos una prueba de escritorio, para el vector

12	3	10	2	8

Prueba de escritorio

I	J	V(I)	V(J)	AUX	vector
					12, 3, 10, 2, 8
1	2	12	3	12	3, 12, 10, 2, 8
	3	3	10	3	3, 12, 10, 2, 8
	4	3	2	3	2, 12, 10, 3, 8
	5	2	8	2	2, 12, 10, 3, 8

Como vemos en la prueba, para el primer recorrido de la variable J (ciclo interno), el algoritmo lanza el menor valor del vector a la primera casilla del arreglo; veamos qué pasa cuando I=2

I	J	V(I)	V(J)	AUX	vector
					2, 12, 10, 3, 8
2	3	12	10		2, 10, 12, 3, 8
	4	10	3	10	2, 3, 12, 10, 8
	5	3	8		2, 3, 12, 10, 8

En el segundo recorrido de la variable J, el siguiente menor valor (tres, en este caso) pasó al segundo lugar; para las siguientes iteraciones se tiene:

I	J	V(I)	V(J)	AUX	vector
					2, 3, 12, 10, 8
3	4	12	10	12	2, 3, 10, 12, 8
	5	10	8	10	2, 3, 8, 12, 10
4	5	12	10	12	2, 3, 8, 10, 12

Al final el vector queda ordenado ascendentemente.

1.3. BÚSQUEDA EN UN VECTOR

El siguiente código busca un valor determinado en las posiciones de un arreglo. Recorre el vector elemento por elemento y en caso de encontrar el valor buscado, cambia el estado de una variable (Suiche) e imprime un mensaje indicando la posición en la que se encuentra. Si el elemento no está en el vector, la variable **Suiche** nunca cambia de estado (permanece Falsa) y el algoritmo indica que este elemento no está en el vector.

```
Entero F, I, Pos, V(N)
Booleano Suiche←Falso
Imprima ("Ingrese elemento a buscar")
Lea (F)
Para I←1 hasta N
      Si (V (I) = F) entonces
          Suiche←Verdadero
          Pos←I
          Imprima ("el valor "F se encuentra en la posición" Pos)
          I ←N+1
      Fin SI
Fin Para
Si (Suiche = Verdadero) Entonces
          Imprima ("el valor "F "se encuentra en la posición" Pos)
Si No
          Imprima ("el valor "F "No se encuentra en el Vector")
Fin Si
```

El algoritmo presenta dentro del ciclo **PARA** la instrucción especial **I←N,** la cual produce el rompimiento del ciclo cuando encuentra el valor buscado, esto es, dispara a la variable controladora a un valor superior al tope del ciclo, indicando al algoritmo que ya encontró lo que estaba buscando.

RECUERDE

Para hacer operaciones sobre un vector se requiere de una estructura repetitiva (regularmente el ciclo PARA), para hacer el recorrido por cada una de las posiciones del vector o arreglo unidimensional.

ACTIVIDAD
Trabajo de vectores

Resuelva la actividad y compare los resultados con otro compañero. Si hay diferencias traten de resolverlo en grupo.

Objetivo: Diseñar algoritmos que utilicen arreglos unidimensionales (vectores)

Diseñe algoritmos para:

1. Llenar un vector con los primeros 10 números impares, mayores a un número entero ingresado por el usuario.
2. En un vector se llevan las edades de los alumnos de un grupo de 10 estudiantes, en el otro se llevan los nombres de cada uno de los estudiantes, determinar la edad, el nombre del estudiante menor y el promedio de las edades.
3. Buscar en un vector de números enteros los elementos que son múltiplos de 7 o de 13.
4. Reorganizar vector, cambiar los elementos pares, ubicándolos en las posiciones iníciales, después se colocan los impares.
5. Se tiene un vector de enteros, diseñar un algoritmo para ubicar los elementos que están en posiciones pares a posiciones impares y viceversa.

2. MATRICES

Introducción a los arreglos bidimensionales

El arreglo bidimensional se puede tratar como un vector de vectores. Es un conjunto de elementos, todos del mismo tipo, en el cual el orden de los componentes es significativo y en el que se necesita especificar dos subíndices para identificar cada elemento del arreglo o array (en inglés). Se denomina *bidimensional* porque el grupo de elementos obedece simultáneamente a dos conceptos de clasificación, que se representan por filas y columnas. También se conoce como matriz.

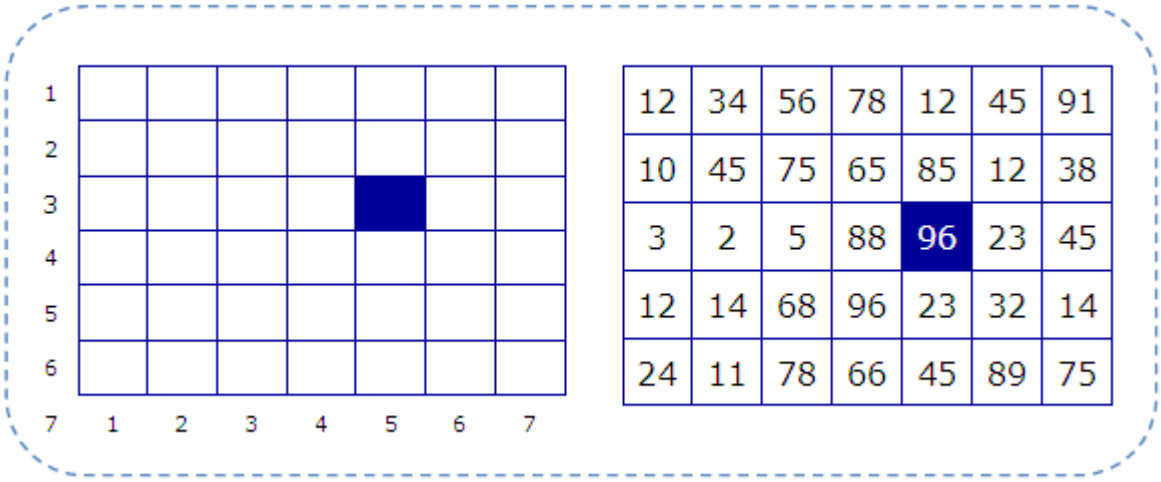


Figura 2. Representación de un arreglo bidimensional o matriz

De esta manera, al hacer referencia al elemento **M (3,5),** se está indicando el valor de la matriz **M** en la posición **(3, 5)** donde **tres** indica la fila y **cinco** la columna. Así por ejemplo, si llamamos **M** la matriz de la figura anterior, el elemento **M (2, 3) =96**. Nótese que las filas y columnas tienen como primer índice el elemento uno. Algunos lenguajes empiezan en la fila y columna cero.

La posición del dato seleccionado (destacado en negro) en la matriz anterior, se referencia como **NombreMatriz (3, 5)** donde 3 indica la fila y 5 la columna y **NombreMatriz** es el nombre del arreglo bidimensional que está declarado en el mismo lugar donde se declaran las variables.

2.1. ÍNDICES DE UNA MATRIZ

Se denomina índice a un valor entero representado por una expresión que describe la posición relativa que ocupa un elemento en un arreglo. Los índices permiten que el acceso a cada elemento se realice de manera directa, sin necesidad de recurrir a otros elementos vecinos. Por lo tanto, es fundamental controlar el valor del índice de tal manera que no sobrepase los límites del arreglo, ya que así se evita la invasión de información contenida en otros campos.

2.2. DECLARACIÓN DE MATRICES

Para declarar matrices se utiliza la siguiente sintaxis:

Tipo Nombre_Matriz (Filas, Columnas) donde tipo indica el tipo de datos que almacenara la matriz, como entero real, carácter, etc.

Ejemplo: **Entero H (3, 4)** indica matriz de valores enteros de 3 filas por 4 columnas.

Referencia a los elementos de una matriz
Para referirse a un componente o elemento particular de un arreglo, ya sea para leerlo, mostrarlo o modificarlo, se escribe el nombre del arreglo del cual hace parte, seguido de la posición que ocupa el elemento entre paréntesis **NombreMatriz (Fila, Columna).**

Ejemplo: Diseñe un algoritmo para llenar una matriz de 5 filas por 5 columnas, de modo que en cada posición quede la suma de los índices de fila y columna correspondientes a esa posición:

```
Entero I, J, M (5, 5)
Para I←1 Hasta 5
    Para J←1 hasta 5
      M (i, j) ←I+J
    Fin Para
Fin Para
```

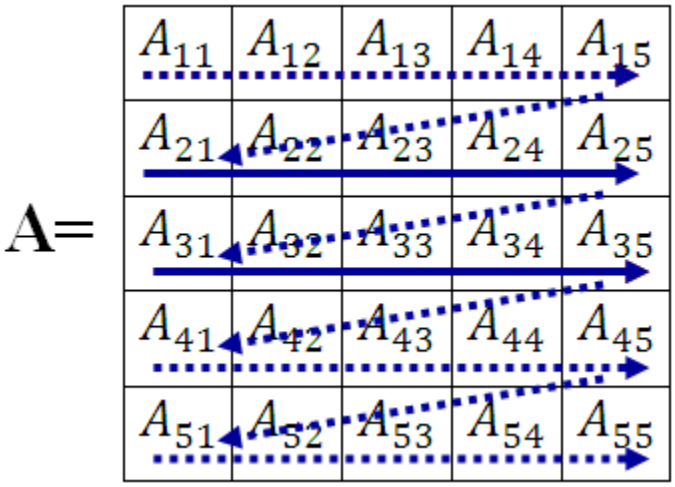
La siguiente figura muestra como quedó llena la matriz

2	3	4	5	6
3	4	5	6	7
4	5	6	7	8
5	6	7	8	9
6	7	8	9	10

En este documento se presentan otros ejemplos necesarios para el aprendizaje del tema (Ver ejemplos en el *anexo 3* al final de este documento)

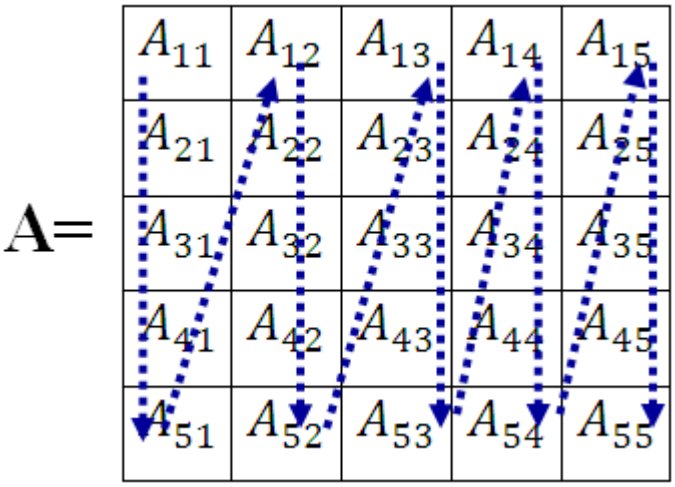
2.3. RECORRIDOS

El acceso o lectura de cada uno de los elementos de una matriz se puede hacer por filas o por columnas dependiendo de la necesidad. En el recorrido por filas primero nos paramos en la primera fila y nos desplazamos por cada uno de los elementos de la fila, una vez llegamos al último elemento de la fila, pasamos al primer elemento de la segunda fila y así sucesivamente, hasta completar las filas de la matriz, como se muestra a continuación:



Pseudocódigo recorrido por filas

```
Para I←1 Hasta Filas
    Para J←1 hasta Columnas
      Lea (A (I, J))
    Fin Para
Fin Para
```



Pseudocódigo recorrido por columnas

```
Para J←1 Hasta Columnas
    Para I←1 hasta Filas
      Lea (A (I, J))
    Fin Para
Fin Para
```

El siguiente ejemplo presenta la utilidad de cada uno de estos recorridos (Ver ejemplo en el *anexo 4* al final de este documento)

ACTIVIDAD
Trabajo con Matrices

Resuelva la actividad y compare los resultados con otro compañero. Si hay diferencias traten de resolverlo en grupo.

Objetivo: Diseñar algoritmos que utilicen arreglos bidimensionales (Matrices).

1. Cree una matriz (5X5) de 25 datos enteros, determine el promedio de las filas de la matriz.
2. Cree dos matrices de enteros A y B de (5x5)

posiciones cada una, sume las matrices y muestre el resultado de la suma.

3. Cree una matriz (5X5) de 25 datos enteros, determine la suma y el promedio de los valores que están en la diagonal principal de la matriz. La diagonal principal de una matriz son los valores para los cuales la fila tiene igual valor que la columna.
4. Cree una matriz (5X5) de 25 datos enteros, almacene en un vector los valores que están en la diagonal principal de la matriz, y determine las filas que tienen acumulado mayor al acumulado de esta diagonal.
5. Diseñe un programa que permita llenar una matriz 8x8, con números del 1 al 64, siguiendo un movimiento del caballo (ajedrez), empezando con 1 hasta 64, se empieza desde cualquier posición.
6. Se sabe que una expresión es palíndromo, si se lee igualmente al derecho que al revés. Verifique si una frase ingresada por el usuario es o no palíndromo. Ejemplos de palíndromos: 13531, nada Adán, “dabale arroz a la zorra el abad”, “anita lava la tina”
7. Diseñe un algoritmo que permita crear una matriz con elementos diferentes. Si el usuario ingresa un valor que ya está en la matriz, el algoritmo le indicará que ese elemento ya está.
8. En una matriz de tres filas se almacenan los nombres de personas (en la primera fila); en la segunda fila las edades y en la tercera las estaturas en centímetros. El programa debe mostrar el nombre de la persona con la menor edad y el nombre de la persona más alta. Muestre además los nombres de las personas más altas y los nombres de las personas más jóvenes.
9. Diseñe un algoritmo que permita determinar la moda de una matriz (Moda: elemento que más se repite)

3. PROGRAMACION MODULAR

Introducción

Una de las técnicas más utilizadas en la resolución de problemas complejos consiste en dividirlos en varios subproblemas más sencillos. Estos subproblemas se resuelven por separado y luego se integran las soluciones para obtener la solución del problema original.

La programación modular es una técnica en la que se divide un programa en partes más simples y bien diferenciadas, llamadas módulos o subprogramas, que pueden ser analizadas y programadas por separado. Cada parte puede ser una función (devuelve un valor) o un procedimiento (hace una tarea, pero no retorna valor). El algoritmo o programa principal, entrega el control a cada uno de los módulos y, una vez estos se han ejecutado, vuelve a tomar el control, continuando la ejecución del programa por donde los invocó. De tal manera, la solución del problema se lleva a cabo en apartados independientes y de manera más sencilla.

La programación modular impone algunas reglas, entre las que se destacan:

- **Regla 1:** cada módulo tiene que tener un punto de entrada y otro de salida; es decir, una vez haya realizado la tarea del módulo, debe devolver el control al programa principal o subprograma que lo invocó.
- **Regla 2:** en el programa principal, se invocan los módulos, pero también pueden ser invocados desde otros módulos.
- **Regla 3:** en lo posible, los módulos deben tener independencia, respecto al programa principal. De tal manera, los módulos realizarán una tarea genérica, para que así se pueda reutilizar su código, si fuese necesario, en otro programa, sin tener que tocar el código del módulo.
- **Regla 4:** los datos del programa principal que el módulo necesite, serán pasados desde allí. Es decir, deben evitarse las variables globales.

3.1. SUBPROGRAMAS: PROCEDIMIENTOS Y FUNCIONES

Cuando nos enfrentamos con un problema complejo, lo mejor es tratar de dividir el problema en problemas más pequeños, aplicar la frase “Divide y Vencerás”. Se trata entonces de diseñar sub-algoritmos, de tal forma que al unir estos sub-algoritmos se pueda tener una solución al problema inicial. La idea es tratar de diseñar los módulos o sub-algoritmos de acuerdo a una tarea específica. Se diseñan una vez, pero se pueden llamar muchas veces dentro del mismo algoritmo. Esta forma de proceder facilita la búsqueda de errores, el mantenimiento y sobretodo la comprensión del algoritmo.

Esta técnica también se conoce como DISEÑO DESCENDENTE, un problema complejo se divide en pequeños problemas y a su vez éstos en otros más pequeños, hasta obtener problemas fáciles de entender. Los lenguajes de programación de alto nivel están construidos pensando en esta forma de trabajo. La siguiente figura presenta un caso de subdivisión algorítmica para un caso especial.

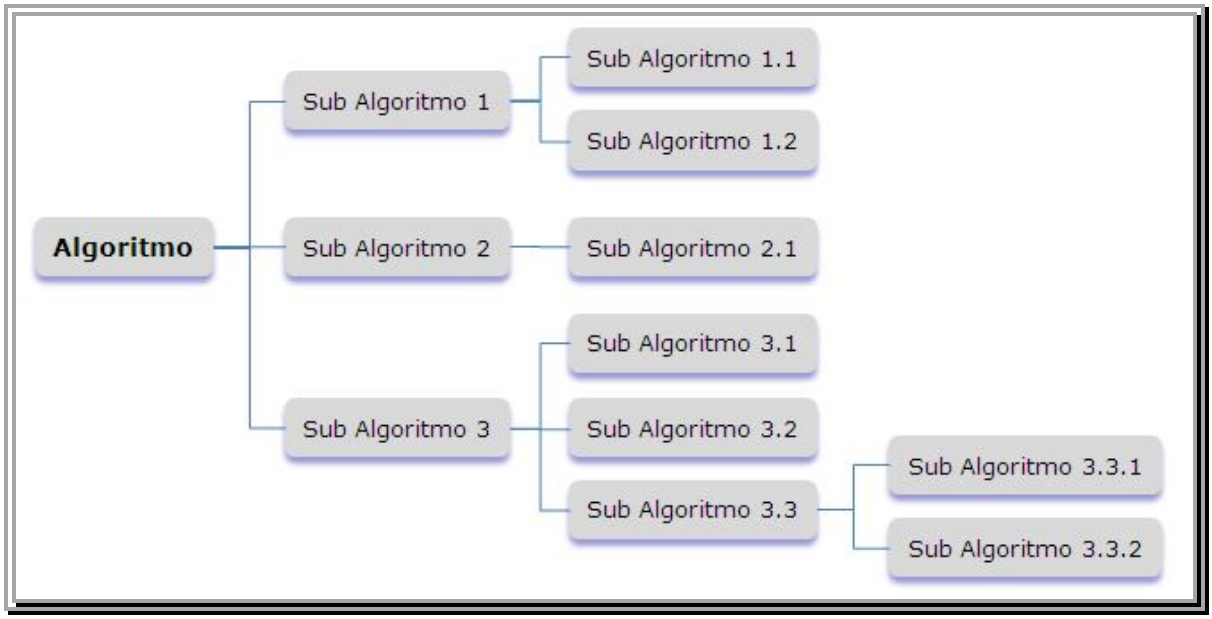


Figura 3. Diseño Descendente

Los sub-algoritmos pueden realizar las mismas funciones que un algoritmo, leer datos, realizar cálculos matemáticos y devolver los resultados; el subprograma se utiliza por el programa principal para un objetivo específico.

3.1.1. Funciones

Matemáticamente, una función $f(x)$ es una operación que toma uno o varios valores, que llamaremos argumentos y que produce un valor o resultado, que llamaremos valor de la función. Tomemos como ejemplo la siguiente función: $f(x,y) = x^2 + y$

Esta es un función con dos argumentos (x,y) el valor de la función en los valores $x = 2$, $y = 1$,
es $f(2,1) = 2\text{^}2 + 1$, o sea, $f(2,1) = 5$.

Las funciones o subprogramas se construyen de manera similar a los algoritmos; una cabecera que inicia con el valor que retorna la función, seguido de la palabra función, el nombre y los argumentos de la función, luego el cuerpo de la función:

```
Y=NombreFuncion(Lista parámetros)
Cuerpo de la función
```

Ejemplo: una función llamada *suma* que se encarga de sumar dos números se construye mediante el siguiente pseudocódigo:

```
Función y=suma(x, y)
y = x + y
```

Una función devuelve un valor o resultado según los parámetros de entrada y se invoca haciendo referencia a su nombre, con su lista de parámetros actuales (separados por comas); en la función se declara un tipo estándar, ya sea real, entero, carácter, booleano, etc.

Una función se invoca asignando a una variable el nombre de la función con sus respectivos parámetros. El programa principal o subprograma invocador lo hará con una instrucción como sigue:

```
U= NombreFuncion (a, b)
```

Ejemplo: si invocamos la función del ejemplo anterior para sumar los números 2 y 5 escribimos:

```
z = suma (2,5)
```

Prototipo de una función

El pseudocódigo completo para construir una función tiene el siguiente aspecto:

```
<Tipo_De_Resultado> Función NOMBRE_FUNCIÓN (argumentos)
Tipo y Variables Locales
INICIO
  <Instrucciones>
  Retornar <Valor>
Fin
```

Veamos un ejemplo: construyamos una función que determine el mayor de dos valores enteros

```
Entero funcion Mayor (Entero X, Entero Y)
Entero Numero_Mayor
INICIO
  Si (X > Y) Entonces
    Numero_Mayor ←X
  Si No
    Numero_Mayor ←Y
  Fin Si
  Retornar  Numero_Mayor
FIN
```

El encabezado de la función inicialmente se compone de la siguiente manera

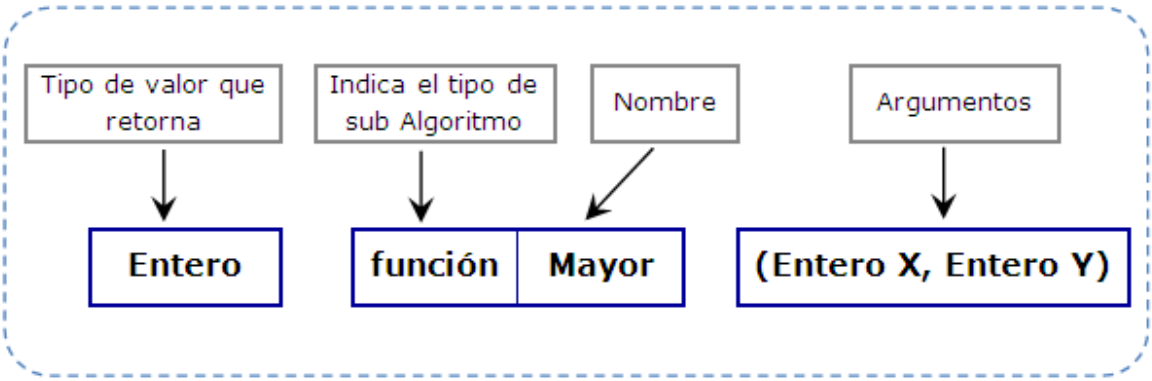


Figura 4. Prototipo encabezado función

Entero: significa que la función retorna un valor entero,
Función: indica que es una función,
Mayor: nombre asignado a la función (dado por el programador)
(Entero X, Entero Y): tipos y nombres de los argumentos de la función.

Observe que la función tiene un nombre acorde a su función y recibe dos argumentos que se pasan entre paréntesis. El cuerpo de la función se sigue con las variables locales (variables que se utilizan dentro de la función), la palabra inicio, el cuerpo y el fin de la función. En el presente caso la función tiene una variable local llamada Número_Mayor, que es la variable encargada de almacenar el mayor de los dos valores pasados como argumentos.

Para que las acciones de una función sean ejecutadas, se requiere que la función sea invocada desde el programa principal o desde otros subprogramas, con el fin de entregar los parámetros que éstos necesiten.

En este documento hay ejemplos para practicar este tema (Ver ejemplo en el *anexo 5* al final de este documento)

3.1.2. Procedimientos

Las funciones retornan un valor, pero en algunas ocasiones se requieren programas que calculen varios valores en lugar de uno solo, en estas situaciones las funciones pierden utilidad y se requiere disponer de otro tipo de subprograma que llamaremos el procedimiento. Los procedimientos no producen (devuelven) valores.

```
Procedimiento Nombre_Procedimiento (Argumentos)
  Variables
  <Instrucciones>
Fin Procedimiento
```

Vea ejemplos en el *anexo 6* al final de este documento, para ordenar un vector y otro para invocar a una función o procedimiento.

3.1.3. Las variables locales

Para realizar tareas, la funciones y los procedimientos pueden necesitar algunas variables de apoyo que llamaremos variables locales. Las variables locales se crean en el momento en que se activa la función o el procedimiento, y desaparecen en el momento en que la función o el procedimiento terminan.

Ejemplo: El siguiente ejemplo muestra la utilidad de las variables locales.

Diseñar un algoritmo que permita obtener la suma de las cifras de un número.

```
Entero Suma_Cifras (Entero X)
  Entero C, Suma←0
  Mientras (X > 0)
    C←X Mod 10
    Suma←Suma + C
    X←X DIV 10
  Fin mientras
 Retornar Suma
Fin Suma_Cifras

INICIO
   Variables
   Entero A
   Imprima ("Ingrese Número entero")
   Lea (A)
   Imprima ("la suma de Cifras de "A "Es"  Suma_Cifras (A))
FIN
```

En este caso las variables de la función **Suma_Cifras (X, C, Suma)** son variables locales.

Prueba de escritorio (suponiendo que el usuario ingresa el número 1254)

A	X	C	Suma
1254	1254		0
	125	4	4
	12	5	9
	1	2	11
		1	12

Las variables **X, C** y **Suma** son variables locales, la variable **A** es una variable local al programa principal.

La ejecución del algoritmo inicia con la declaración de la variable entera **A**, luego se invoca la función **Suma_Cifra**, lo que indica que la función toma el control del algoritmo, la instrucción **Retornar suma** le entrega de nuevo el control al programa principal.

BIBLIOGRAFÍA

[1] BECERRA SANTAMARIA, César A. Algoritmos: Conceptos básicos. Kimpres, Bogotá, 1998.

[2] BECERRA SANTAMARIA, César A. Turbo Pascal. Programación orientada a objetos. Editor César Becerra, Bogotá, 1995.

[3] Bohm, C y Jacopini, G. "Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules", Communications of the ACM, Vol.9, No.5, mayo 1966, pp.336-371.

[4] DEITEL, H.M. Y DEITEL, P.J. Cómo programar en C/C++. Segunda edición. Prentice Hall, México, 1995

[5] FORSYTHE, Alexandra y otros. Lenguajes de diagramas de flujo. Limusa, 1979.

[6] HEILEMAN, GREGORY L. Estructura de datos, Algoritmos y programación orientada a objetos. McGraw Hill. Madrid, 1997.

[7] JOYANES, Luis. Problemas de metodología de la programación. McGraw Hill, México, 1990.

[8] JOYANES, Luis. Fundamentos de programación. McGraw Hill, México, 1999.

[9] KOFFMAN Elliot B. Pascal. Introducción al lenguaje y resolución de problemas con programación estructurada. Fondo educativo interamericano, Wilmington, Estados Unidos, 1985.

[10] ORREGO VILLA, Gildardo Antonio. Diseño de algoritmos. Escuela de Ingeniería de Antioquia, Medellín, 2003.

[11] RIOS, Fabián. Soluciones secuenciales. Ciencia y tecnología. Universidad de Antioquia, Medellín, 1995.

ENLACES DE INTERÉS

Mini Diccionario Informático
Esta página presenta las definiciones más comunes en informática y presenta muchos ejemplos de algoritmos diseñados en lenguaje C.
http://www.carlospes.com/minidiccionario/variable_contador.php

Glosario de Términos Computacionales
Este recurso presenta un completo glosario de los recursos computacionales más utilizados.
http://members.fortunecity.com/miprofe/archivos/TerminosComputacionales.htm

Programación con ciclos
Este recurso presenta la sintaxis utilizada por los diferentes lenguajes de programación para las sentencias cíclicas.
http://www.magusoft.net/compuv/13-2_ciclos.html

Fundamentos de programación
Esta página presenta el desarrollo completo de un curso de fundamentos de programación, permite conocer aspectos esenciales del maravilloso mundo de la programación de computadores.
http://es.wikibooks.org/wiki/Fundamentos_de_programaci%C3%B3n/Introducci%C3%B3n

Diagramas de flujo

http://www.fundibeq.org/metodologias/herramientas/diagrama_de_flujo.pdf

Este recurso facilita la interpretación de algoritmos con diagramas de flujo, puede servir como complemento al análisis algorítmico.

Arreglos
En este recurso se encuentra una variedad de ejemplos de diferentes tipos, que permiten al estudiante comprender mejor esta estructura de programación.
http://aplicaciones.virtual.unal.edu.co/drupal/files/Arreglos%20y%20Matrices%20-%20Programacion%20de%20Computadores.pdf

Estructura de datos

En este recurso se encuentra gran cantidad de material relacionado con las estructuras de datos.

http://www.unalmed.edu.co/~fjmoreno/ed/

Ordenamiento

Este recurso le facilita la comprensión del algoritmo para ordenar arreglos

http://lc.fie.umich.mx/~calderon/programacion/notas/Burbuja.html

PREGUNTAS FRECUENTES

¿Qué son los argumentos de una función?
Una función es un módulo de código que se puede utilizar varias veces; los argumentos son los datos que aparecen en la definición de la función.

¿Se pueden almacenar valores de diferente tipo en un vector?
No. En un arreglo unidimensional (o vector) sólo se pueden almacenar valores del mismo tipo, en la declaración del vector se antepone el tipo y sólo se pueden almacenar en él valores de ese tipo. Ejemplo: si se declara un vector de la forma Entero V (10), este arreglo sólo podrá almacenar valores de tipo entero.

¿Cuántas veces se puede llamar o invocar una función o un procedimiento?
Las funciones y los procedimientos se pueden invocar desde el programa principal o desde cualquier otro módulo las veces que sea necesario.

¿Qué es el valor de retorno de una función?
Es el valor que devuelve una función después de ejecutar algunas operaciones. A diferencia de los procedimientos, las funciones deben tener un valor de retorno, ese valor de retorno se utiliza en el programa principal o en cualquier otro módulo del programa o algoritmo.

¿Se pueden construir arreglos de más de dos dimensiones?
Aunque en este módulo sólo se presentan arreglos de dos dimensiones como máximo, en algunas ocasiones es necesario diseñar algoritmos que utilizan arreglos de varias dimensiones, estos arreglos comparten las mismas características de los vectores y matrices, tanto para asignación y operación sobre él.

GLOSARIO

Arreglo: conjunto o agrupación de variables del mismo tipo cuyo acceso se realiza por medio índices.

Vector: arreglo unidimensional, utiliza un único índice para acceder a cada una de sus entradas.

Matriz: arreglo de dos dimensiones, se accede a sus elementos por medio de dos índices, el primero de ellos corresponde a la fila y el segundo a la columna.

Función: es un subprograma que puede ser llamado dentro de un programa principal, hace que el código principal se detenga y se dirija a ejecutar el código de la función, tiene la condición especial de retornar un valor; recibe también parámetros llamados argumentos de la función.

Procedimiento: acción de proceder o el método de ejecutar algunas cosas. Se trata de una serie común de pasos definidos (sub-algoritmo o subprograma) que permiten realizar una tarea. A diferencia de la función, el procedimiento no tiene un valor de retorno.

Variable local: variable a la que se le otorga un ámbito local que puede ser una función o un procedimiento. Tales variables sólo pueden accederse desde la función o bloque de instrucciones en donde se declaran.

Variable global: accesible desde todas las partes del algoritmo. El uso de este tipo de variables suele considerarse como una práctica poco adecuada, pues tiene un riesgo asociado al poder ser modificada en cualquier lugar del algoritmo, este tipo de variable tiene un poder ilimitado en el algoritmo.

ANEXO 1

Este documento hace parte del tema **"Operaciones sobre el arreglo completo"**.

Luego de estudiarlo debe regresar al documento de estudio y continuar con el tema.

EJEMPLO 1

En un arreglo se almacenan las edades de un grupo de 20 estudiantes, se quiere determinar la suma de las edades, la edad promedio, la mayor edad y la menor edad.

```
Inicio
 Variables
 Entero Edad (10), I, Mayor←0, Menor←200, suma←0
 Real Promedio
 Para I←1 Hasta 10
    Imprima ("Ingrese la edad del estudiante" I)
   Lea (Edad (I))
   Suma ← Suma + Edad (I)
   Si (Edad (I)  >  Mayor) Entonces
      Mayor ← Edad (I)
   Fin Si
   Si Edad (i) < Menor Entonces
      Menor ← Edad (I)
   Fin Si
 Fin Para
 Promedio ← Suma/10
 Imprima ("La edad Mayor es" Mayor)
 Imprima ("La edad Menor es" Menor)
 Imprima ("La suma de la edades es" Suma)
 Imprima ("El promedio de la edades es" Promedio)
FIN
```

Tenga en cuenta las siguientes observaciones referentes a este algoritmo:

1. La variable **Mayor** la inicializamos en el menor valor admisible, en este caso cero, para que se haga el cambio una vez ingrese la primera edad, observe que el condicional que permite determinar este valor es:

```
Si Edad (i) > Mayor Entonces
  Mayor ← Edad (I)
Fin Si
```

2. Igualmente, inicializamos la variable **Menor** en el máximo valor posible. Como sabemos que no hay posibilidad de una edad superior a 200 años, este es el valor seleccionado para inicializar esta variable.

```
Si Edad (i) < Menor Entonces
  Menor ← Edad (I)
Fin Si
```

3. Para determinar el acumulado total de las edades, basta con tener una variable acumuladora, en este caso llamada **Suma**, la cual inicializamos en cero.

4. Para el promedio de edades, que es un valor de tipo real, tomamos la suma total de las edades y la dividimos por la cantidad de estudiantes, o sea 100. Note que el promedio se determina por fuera del ciclo **PARA,** una vez se conoce la suma total de edades.

La siguiente prueba de escritorio nos ayuda a comprender las líneas del algoritmo

I	EDAD(I)	MAYOR	MENOR	SUMA	PROMEDIO
		0	200	0	
1	18	18	18	18	
2	15	18	15	33	
3	16	18	15	49	
4	19	19	15	68	
5	20	20	15	88	
6	14	20	14	102	
7	15	20	14	117	
8	18	20	14	135	
9	15	20	14	150	
10	21	21	14	171	
					17.1

Observe como cambian los valores de las variables **Mayor** y **Menor** en las iteraciones, por ejemplo en la segunda iteración la edad del estudiante es 15, y la variable **Menor** tenía un valor de 18; como 15<18, se cumple la condición y se hace el cambio de valor en la variable, tomando la variable **Menor** el valor 15. Igualmente pasa con la variable **Mayor** en la iteración 10, pues la edad 21 es mayor que el valor de la variable que era 20.

EJEMPLO 2

Analizar el siguiente algoritmo, determinar los valores del vector en cada posición.

```
Inicio
  Variables
  Entero Vector (10), I, Suma←0
  Para I←1 Hasta 10
    Vector (I) ←I^2 +3
  Fin Para
FIN
```

I^2: significa elevar al cuadrado el valor de la variable I.

En cada posición se almacena el valor de la posición elevado al cuadrado y se le suma tres, así para la posición uno se tiene:

1^2+3=4, para la posición 2 se tiene 2^2+3=7, y así sucesivamente, en este caso el vector se llena automáticamente, los valores serán 4, 7, 12 (3*3+3), 19 (4*4+3), 28, 39, 52, 67, 84. Supongamos ahora que deseamos llenar un vector con la siguiente información:

20	18	16	14	12	10	8	6	4	2

¿Cuáles serán las instrucciones para llenarlo de esta forma?

Como vemos en este caso, la primera posición debe tener el valor 20, y los demás valores se forman disminuyendo en dos al registro anterior, por lo tanto, una forma puede ser la que se propone a continuación:

```
Inicio
 Variables
 Entero Vector (10), I, Cont←22
 Para I←1 Hasta 10
   Vector (I) ←Cont –I*2
 Fin Para
FIN
```

ACTIVIDAD

Haga una prueba de escritorio para verificar la validez de esta propuesta.

EJEMPLO 3

Diseñe un algoritmo que permita llenar un vector de 10 posiciones con la serie de Fibonacci. La serie de Fibonacci es la siguiente: 0 1 1 2 3 5 8 13 21 34…en la cual cada número es la suma de los dos anteriores, (exceptuando, naturalmente, los dos primeros, los cuales son 0 y 1).

Solución:

```
INICIO
 Variables
 Entero Vector (10), I,
 Vector (1) ←0
 Vector (2) ←1
 Para I←3 Hasta 10
   Vector (I) ←Vector (I-1) + Vector (I-2)
 Fin Para
FIN
```

Observe entonces a partir del pseudocódigo que la tercera posición será:

Vector (3) ←Vector (3-1)+Vector (3-2), o sea,

Vector (3) ← Vector (2)+ Vector (1), la siguiente **Vector (4) ← Vector (3)+ Vector (2**), y así sucesivamente.

RECUERDE

La primera posición de un vector es la posición uno. En varios lenguajes de programación se maneja como primera posición la posición cero.

EJEMPLO 4

En una estación de monitoreo se quiere saber la temperatura promedio semanal del agua de un río; diariamente se toma la temperatura del río y se almacena en un vector. Se quiere determinar el día en que el río estuvo más cálido y durante cuántos días la temperatura del agua se mantuvo por debajo de la temperatura promedio.

Solución:

```
INICIO
 Variables
 Entero Temperatura (7), I, Suma←0, cont←0, Mayor ←-100
 Real Promedio
 Para I←1 Hasta 7
     Imprima ("ingrese la Temperatura del día", I)
     Lea (Temperatura (I))
     Suma ←Suma + Temperatura (I)
     Si (Temperatura (I) > Mayor) Entonces
       Mayor← Temperatura (I)
     Fin SI
 Fin Para
 Promedio ←Suma/7
 Para I←1 Hasta 7
     Si (Temperatura (I) > Promedio) Entonces
       Cont←Cont+1
     Fin SI
 Fin Para
```

```
Imprima ("La temperatura Mayor" mayor)
Imprima ("Promedio de Temperaturas" Promedio)
Imprima ("Temperaturas Mayores" Cont)
FIN
```

Se tiene un contador para saber la cantidad de días cuya temperatura fue superior al promedio.

Prueba de escritorio

I		1	2	3	4	5	6	7	
T		23	22	20	19	18	17	18	
Suma	0	23	55	75	94	112	125	143	
Promedio									20,42
Contador	0	1	2						
Mayor	100	23							

La prueba muestra el valor inicial de las variables en la segunda columna, en la primera fila se presentan los valores de la variable controladora del ciclo (I), la segunda fila presenta las temperaturas del río (T) en cada día de la semana, la tercera fila presenta los valores de la variable acumuladora (Suma), en la cuarta fila está la temperatura promedio (Promedio), que se calcula después de tener todas las temperaturas, en la quinta fila se presenta el contador de temperaturas superiores al promedio y por último, la temperatura mayor en la semana.

RECUERDE

Para obtener el promedio de valores, primero se debe tener la suma total y posteriormente se divide por la cantidad de datos.

Cuando se quiere llenar el vector completo utilizamos sentencias repetitivas (Para, Mientras, Haga_Mientras).

EJEMPLO 5

Llenar un vector con los primeros 10 números pares.

```
Entero Vector (10), I
Para I←1 Hasta 10
    Vector (I) ←I*2
Fin Para
```

En la línea **Vector (I) ←I*2** se hace la asignación completa del vector, aunque podría hacerse posición por posición Vector (1) ←2, Vector (2) ←4, y así sucesivamente.

RECUERDE

La lectura y escritura de datos Lea (V (2)); esta instrucción indica que lea el dato de la posición dos del vector V. las operaciones de lectura/escritura de vectores utilizan normalmente estructuras repetitivas. Imprima (V (30)) muestra el valor del vector V en la posición 30.

Nota: a las operaciones generales, sobre todos los elementos de un vector se le conoce como RECORRIDO del vector. Normalmente, estas operaciones se ejecutan utilizando variables de control o subíndices del vector.

ANEXO 2

Este documento hace parte del tema **"Operaciones sobre el arreglo completo"**.

Luego de estudiarlo debe regresar al documento de estudio y continuar con el tema.

EJEMPLO

Diseñar un algoritmo que permita llenar un vector con los diez primeros números primos.

Solución:

La idea del algoritmo es empezar con el número uno, determinar si es primo, en caso afirmativo almacenarlo en el vector, en otro caso pasar al siguiente número y repetir esta operación hasta encontrar diez números.

```
INICIO
  Entero I, Cont←0, divisores←0, Número←0, Primos (5)
  Mientras (Cont <5)
       Número←Número +1
      Para I←1 hasta Número
        Si (Número Mod I = 0) Entonces
           Divisores←Divisores +1
        Fin SI
      Fin Para
      Si (Divisores  = 2) Entonces
         Cont←Cont+1
         Primos (Cont) ←Número
     Fin Si
     Divisores←0
  Fin Para
FIN
```

Prueba de escritorio

I	Cont	Divisores	Número	Primos( )
	0	0	0	
			1	
1		1	2	
1		1		
2	1	2	3	2
		0		
1		1		
2		1		
3	2	2	4	3
1		1		
2		2		
3		2		
4		3	5	
1		1		
2		1		
3		1		
4		1		
5	3	2	6	5

Observe que los números etiquetados como primos (2, 3 y 5) tienen únicamente dos divisores (el uno y el mismo número, ejemplo: el 11 es primo, ya que sus únicos divisores son el 1 y el 11); la prueba de escritorio debe ejecutarse hasta encontrar cinco números primos, o sea que en esta lista faltan el siete y el número once.

ANEXO 3

Este documento hace parte del tema **"Declaración de matrices"**.

Luego de estudiarlo debe regresar al documento de estudio y continuar con el tema.

EJEMPLO 1

Diseñe un algoritmo que permita obtener la posición (fila y columna) del elemento de mayor valor en un matriz, suponiendo que la matriz está llena.

```
Entero I, J, M (5, 5), Mayor←0, Fila_Mayor←0, Col_Mayor←0
Para I←1 Hasta 5
   Para J←1 hasta 5
     Si (Mayor  < M (i, j)) Entonces
       Mayor← M (i, j)
       Fila_Mayor ← I
       Col_Mayor ←J
     Fin Si
   Fin Para
Fin Para
Imprima ("El elemento mayor está en la fila "Fila_Mayor" Columna: "Col_Mayor)
```

Prueba de escritorio

En el trabajo con matrices utilizamos dos ciclos anidados, normalmente utilizamos el ciclo **PARA**, pues conocemos de entrada el número de filas y columnas que será necesario recorrer, el primer Ciclo (Ciclo de la I) para hacer el recorrido por las filas, y el ciclo Interno (Ciclo de la J), controlado por J, para recorrer las columnas; el siguiente ejemplo muestra la forma en que se llena la matriz.

```
Entero I, J, M (5, 5), cont←0
Para I←1 Hasta 5
    Para J←1 hasta 5
      Cont←Cont+1
      M (I, J)) ← Cont
    Fin Para
Fin Para
```

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Nótese que antes de asignar el valor a la matriz se incrementar la variable contador.

EJEMPLO 2

Diseñe un algoritmo que llene una matriz de números enteros, los guarde en una matriz y finalmente los imprima. La dimensiones de la matriz estarán dadas por el usuario

```
Entero I, J, Filas, Columnas, M (Filas, Columnas)
Imprima ("ingrese Número de columnas")
Lea (Columnas)
Imprima ("ingrese Número de Filas")
Lea (Filas)
Entero M (Filas, Columnas)
Para I←1 Hasta Filas
    Para J←1 hasta Columnas
      LEA (M (I, J))
    Fin Para
Fin Para
Para I←1 Hasta Filas
    Para J←1 hasta Columnas
      Imprima (M (I, J))
    Fin Para
Fin Para
```

ANEXO 4

Este documento hace parte del tema **"Recorridos"**.

Luego de estudiarlo debe regresar al documento de estudio y continuar con el tema.

EJEMPLO

Objetivo: practicar con matrices.

Una cadena de almacenes (5 sucursales) guarda en una matriz las ventas semanales, y el promedio de las ventas. El siguiente cuadro presenta las ventas diarias en cada una de sus sucursales, marcadas con S1, S2, S3, S4, S5.

	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
S1	340	345	567	458	145	265	897
S2	540	457	125	104	100	745	895
S3	830	910	860	450	782	566	489
S4	120	457	411	555	888	568	471
S5	130	140	150	160	125	128	489

Según este cuadro, la sucursal 1 vendió 340 productos el día lunes. Se requiere de un algoritmo que permita determinar el día de mayor venta, la sucursal más rentable y el promedio de venta diario; note que la columna uno corresponde al lunes, dos al martes y así sucesivamente.

Solución:

En este caso se debe recorrer la matriz por filas y por columnas de acuerdo a la solicitud, plantearemos cada algoritmo en un módulo aparte. Para determinar el día de mayor venta, como vemos, los días están representados por las columnas, por lo tanto se deben sumar todos los elementos de una columna e ir comparando columna a columna. El siguiente código trata esta situación:

```
Entero Mayor←0, J, I, Suma←0
Para I←1 Hasta Filas
     Para J←1 hasta Columnas
       Suma ←Suma + M (I, J)
     Fin Para
     Si Mayor < Suma Entonces
       Mayor ← Suma
       Sucursal ← I
    Fin Si
    Suma ←0
 Fin Para
 Imprima (“La sucursal que más vende es”, Sucursal)
```

Ahora, para determinar el día de mejor rendimiento (más ventas), el código siguiente:

```
Entero Mayor_Dia←0, J, I, Suma←0, Mejor_Día
Para J←1 Hasta Columnas
     Para I←1 hasta Filas
       Suma ←Suma +M (I, J)
     Fin Para
     Si Mayor_Día < Suma Entonces
       Mayor_Día ← Suma
       Mejor_Día ← J
    Fin Si
    Suma ←0
 Fin Para
 Imprima (“El día de mas ventas es” Mejor_Día)
```

Para obtener el promedio de las ventas basta con sumar todas las entradas de la matriz y dividir este acumulado por el producto de las filas y columnas.

ANEXO 5

Este documento hace parte del tema **"Funciones"**.

Luego de estudiarlo debe regresar al documento de estudio y continuar con el tema.

EJEMPLO 1

Objetivo: Practicar el concepto de función.

Diseñar funciones para determinar el cuadrado de un número y otro que determine la cantidad de divisores de un número dado.

El cuadrado de un número

```
Entero Cuadrado (Entero X)
INICIO
  Retornar X*X
FIN
```

La cantidad de divisores de un número

```
Entero Cuadrado (Entero N)
  Entero Cont←0, I
  INICIO
  Para I←1 Hasta N
    Si (N Mod I = 0) Entonces
      Cont←Cont+1
    Fin Si
 Fin Para
  Retornar Cont
FIN
```

EJEMPLO 2

Objetivo: Comprender la utilización del programa principal y del subprograma.

Se le pide diseñar un programa que permita determinar la cantidad de números primos de dos cifras.

Solución:

Se debe tener claro que los números de dos cifras están comprendidos entre 10 y 99, diseñaremos una función que nos indique si un número es primo y esta función la llamaremos varias veces en el programa principal; el diseño de la función es el siguiente:

```
Booleano función Primo (Entero X,)
Entero Contador←0
Booleano Suiche ← Falso
INICIO
  Para I ←1 hasta X
    Si (X Mod I = 0) Entonces
      Contador←Contador+1
    Fin Si
  Fin Para
  Si (contador = 2) Entonces
    Suiche ←Verdadero
  Fin Si
  Retornar  Suiche
FIN
```

El programa principal es:

```
INICIO
 Entero I, Cont←0
 Para I←10 Hasta 99
   Si (Primo (I) = Verdadero) Entonces
       Cont ← Cont+1
    Fin SI
  Fin Para
  Imprima ("Hay ", Cont, "Números primos de dos cifras")
FIN
```

Como la función Primo retorna un valor Booleano (verdadero o falso), dentro del ciclo PARA se pregunta en cada número entre 10 y 99 si es primo, en caso de ser cierto (retornar verdadero) y se incrementará un contador. Al final se imprime la cantidad de números primos comprendidos en ese intervalo.
En la línea **Si (Primo (I) = Verdadero)** Entonces, se invoca la función y su valor se compara con el valor booleano.

En el ejemplo anterior se observa la forma de invocar una función dentro del programa principal, hay que tener en cuenta el valor de retorno para saber con qué se compara o a que arible se le debe asignar.

EJEMPLO 3

Diseñar un programa que permita contar los números capicúas presentes en un vector de 10 posiciones. Sabemos que los números capicúas son aquellos que se leen igual de izquierda a derecha, que de derecha a izquierda.

Solución:

La función para determinar si el número es capicúa es la siguiente:

```
Booleano Función Capicúa (Entero Número)
```

```
INICIO
Entero Revés←0, Auxiliar←Número, Residuo←0
Booleano Suiche ← Falso
Mientras (Número >0)
  Residuo ← Número Mod 10
  Revés← Revés*10+ Residuo
  Número ←Número Div 10
Fin Mientras
Si (auxiliar = Revés) Entonces
  Suiche ←Verdadero
Fin Si
Retorne Suiche
FIN
```

Observe que la función retorna un booleano, y recibe como argumento un número entero, tiene como variables locales Revés, Auxiliar y Suiche, se utiliza el ciclo mientras, pues no se sabe el número de veces que se tiene que ejecutar este ciclo, en cada iteración a la variable Número se le quita la última cifra; la idea es obtener el revés de un número, por ejemplo si el número es 4567, la variable revés tendrá el número 7654.

Prueba de escritorio para función capicúa

Número	Revés	Residuo	Auxiliar	Suiche
1225	0	0	1225	Falso
122	5	5		
12	52	2		
1	522	2		
0	5221	1		

Observe que en este caso, la variable **Número** va perdiendo en cada iteración la última cifra (1225, 122, 12, 1), por el contrario, la variable **Revés** va ganando la última cifra de número (5, 52, 522, 5221); si el número fuera capicúa, revés tendría el mismo valor que el número inicial que quedó almacenada en la variable auxiliar.
Veamos otro caso.

Prueba de escritorio para función capicúa

Número	Revés	Residuo	Auxiliar	Suiche
14841	0	0	14841	Falso
1484	1	1		
148	14	4		
14	148	8		
1	1484	4		
0	14841	1		Verdadero

Observe que en este caso la variable revés terminó con el mismo valor que la variable auxiliar, por lo tanto el algoritmo retorna verdadero, indicando que el número sí es capicúa.

Ahora el programa principal

```
INICIO
Variables
Entero Vector (10), I, Cont←0
  Para I ←1 hasta 10
     Imprima (“ingrese Valor en Posición”, I)
     Lea (Vector (I)
     Si (Capicúa (Vector (I))=Verdadero) Entonces
       Cont ← Cont + 1
     Fin Si
  Fin Para
  Imprima (“La cantidad de capicúas en el vector es”, Cont)
FIN
```

ANEXO 6

Este documento hace parte del tema **"Funciones"**.

Luego de estudiarlo debe regresar al documento de estudio y continuar con el tema.

EJEMPLO 1

Diseñar un procedimiento para ordenar un vector

```
Procedimiento Ordenar (Vector (N))
 Entero I, J, AUX
    Para I←1 hasta N-2
      Para J←I+1 hasta N-1
         Si (V (I)>V (J)) entonces
            AUX←V (I)
            V (I) ←V (J)
            V (J) ←AUX
         Fin SI
    Fin Para
  Fin Para
FIN Ordenar
```

En este caso, se pasa el vector a ordenar como argumento del procedimiento, se declaran variables locales al argumento **I, J, AUX**; las variables I, J son las controladoras del ciclo, la variable auxiliar desempeña el papel de burbuja.

EJEMPLO 2

Objetivo: presentar un algoritmo que invoca a una función y procedimiento.

```
Entero Máximo (Entero X, Entero Y)
  Si (X > Y) Entonces
    Retorne X
   Si No
    Retorne Y
  Fin Si

Procedimiento Escribir (Entero R)
  Imprima ("el resultado es" R)
Fin Escribir

INICIO
Variables
  Entero a, b, Max
  Imprima ("ingrese los valores")
  Lea (X, Y)
  Max←Maximo (a, b)
  Escribir (Max)
FIN
```

Como se ve, el programa principal invoca a la función máximo que retorna el mayor de dos valores y luego invoca el procedimiento Escribir, que se encarga de mostrar (o imprimir) este valor por pantalla.

CRÉDITOS

APOYO ACADÉMICO Y ADMINISTRATIVO

Néstor Hincapié Vargas
Rector

Alba Luz Muñoz Restrepo
Vicerrectora Académica

Luis Mariano González Agudelo
Decano
Facultad de Comunicación

Martha María Gil Zapata
Decana
Facultad de Ingeniería

EJECUTORES DEL PROYECTO

Bell Manrique Losada
Coordinadora académica

Sandra Isabel Arango Vásquez
Líder organizacional

Angélica Ricaurte Avendaño
Líder pedagógica

Claudia Patricia Vásquez Lopera
Líder tecnológica

Frank Piedrahita Bedoya
Líder comunicativo

Jaime Echeverri
Autor de contenidos

Equipo de producción de diseño

APOYO DE LOS GRUPOS DE INVESTIGACIÓN

ARKADIUS
Grupo de Investigación de
INGENIERÍA DE SISTEMAS

E-VIRTUAL
Grupo de Investigación
Universidad de Medellín