FUNDAMENTOS DE PROGRAMACIÓN

ALGORITMOS Y ESTRUCTURA DE DATOS

CAPITULO I

RESOLUCION DE PROBLEMAS

Definición

Un problema es un conjunto de cuestiones que se plantean para ser resueltas.
En Informática: se busca la solución utilizando computadoras, mediante un programa (buscando el mejor resultado en tiempo y forma).

Planteo

QUE me están pidiendo;
CÓMO resolverlo;
CON QUÉ, expresar la supuesta solución (recursos, herramientas...).

Dicho de otro modo, responde al siguiente orden de planteo:
1) ESTRATÉGICO
2) LÓGICO
3) HERRAMENTAL

Enunciado

Debe estar dado en forma completa, correcta y sin ambigüedades. El problema tiene una naturaleza, su particularidad, y debo reconocerla para saber si puedo enfrentarlo, para ello se sugiere tener en cuenta:

a) CONOCIMIENTO
b) HABILIDAD
c) EXPERIENCIA

es decir:
- conocer los elementos, propiedades, leyes, **teoría**;
- poseer cualidades que me permitan alcanzar la solución, **creatividad**;
- toma de decisiones, naturaleza que me de mayor **seguridad** y garantía de resolución.

Debo sentirme en condiciones de responder, ya que no entiendo lo que no conozco.

Sugerencias ante un planteo

(pautas que me permiten adquirir una metodología, un orden que me acerca a la solución);

1) Necesito poder captar **datos importantes**. Existen datos secundarios; no agregarlos;
2) Reconocer **relaciones entre datos**; hacer explícito el dato que está oculto en una relación de datos, en el planteo, de acuerdo a la naturaleza del problema;
3) Profundizar en los **detalles**; en la práctica se aconseja leer 3 veces el problema;
4) Dividir el problema en **subproblemas**; si es de complejidad importante se reduce la misma, obteniendo una mejor performance.
-Tener en cuenta que la complejidad está dada por la cantidad de procesos que se realizan;
5) Aplicar la **experiencia** en problemas similares.

-Para tener una orientación no tengo que olvidar el orden; el planteo estratégico.

El paso siguiente consiste en construir la propuesta de solución:

datos de entrada	--------->	procesamiento de datos (algoritmo)	---(generan)--->	información, datos de salida

CAPITULO II

DATOS, ALGORITMOS Y LENGUAJES

Los sistemas de procesamiento de la información

Para los informáticos, datos e información no son sinónimos. *Datos* se refiere a la representación de algún hecho, concepto o entidad real, en cambio, *información* implica datos procesados y organizados.
Un *sistema* en general se define como un conjunto de componentes conectados e interactivos, que tiene un propósito y una unidad total. En consecuencia, sistema de procesamiento de información es un sistema que transforma datos brutos en información organizada, significativa y útil.

El conjunto de instrucciones que especifican la secuencia de operaciones a realizar para resolver un sistema específico o clase de problema se denomina *algoritmo.* En otras palabras, un algoritmo es una fórmula para la resolución de un problema.
Un *programa* se escribe en un lenguaje de programación y a la actividad de expresar un algoritmo en forma de programa se le denomina programación. Un programa consta de una secuencia de instrucciones, cada una de las cuales especifica las operaciones que debe realizar la computadora.

La resolución de problemas exige al menos los siguientes pasos:
1.- Definición o análisis del problema
2.- Diseño del *algoritmo* (Secuencia ordenada de pasos que conducen a la solución).
3.- Transformación del algoritmo en un programa (Fase de codificación).
4.- Ejecución y validación del programa.

CONCEPTO DE ALGORITMO

Para ser un programador eficaz se necesita aprender a resolver problemas de un modo riguroso y sistemático.
Los algoritmos son más importantes que los lenguajes de programación o las computadoras. Un lenguaje de programación es tan sólo un medio para expresar el algoritmo, y una computadora es sólo un procesador para ejecutarlo.

Definición

Un algoritmo es un conjunto de tareas o pasos en una cantidad finita que se ejecutan en un orden determinado, y para determinada situación inicial se resuelve el problema en un tiempo finito.

situación ---------> algoritmo ----------> solución
inicial

Características que debe cumplir

1)Debe ser **correcto**; responder a lo que me piden y resolver el problema;
2)**Eficiente** en cuanto a recursos y tiempo;
3)**Claro**;
4)**Flexible**; poder adaptarse a pequeños cambios de lógica;
5)**Preciso**; e indicar el orden de realización de cada paso;
6)**Estar definido:** si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez;
7)**Ser finito:**, se debe terminar el algún momento; debe tener un número finito de pasos;
8)**Fiable ó confiable**; estar acorde a lo anterior en cuanto a propuesta de solución.

La definición de un algoritmo debe describir tres partes: entrada, proceso y salida.

COMPONENTES DE UNA COMPUTADORA

Hardware es el conjunto de componentes físicos de una computadora, y software es el conjunto de programas que controlan el funcionamiento de una computadora.
El hardware de una computadora se compone de:

_ La Unidad Central de Procesos: es el conjunto de circuitos electrónicos capaces de ejecutar algunos cálculos sencillos. La potencia de una computadora depende completamente de la velocidad y fiabilidad de la CPU.
_ Memoria Central: La información procesada por la CPU se almacena normalmente en la memoria central hasta que se terminan los cálculos. Los programas de computadora también se almacenan en la memoria central.
_ Dispositivos de almacenamiento secundario (memoria auxiliar): Diferentes dispositivos tales como discos y cintas magnéticas se utilizan para almacenar grandes cantidades de información. Para ser procesados por la CPU, los datos se almacenan en dispositivos de almacenamiento secundario y luego deben llevarse a la memoria central.
_ Periférico o dispositivos de entrada / salida: Permiten al usuario comunicarse con la computadora.

LOS LENGUAJES DE PROGRAMACIÓN

El procesador debe ser capaz de interpretar el algoritmo, lo que significa:
- comprender las instrucciones de cada paso;
- realizar las operaciones correspondientes.

Semántica: Significado, interpretación de lo que la sintaxis está expresando.

Sintaxis: Combinación entre cierta **simbología** o signos, y **palabras clave** (que no sean ambiguas).

Los principales tipos de lenguajes utilizados en la actualidad son tres:
- Lenguaje máquina;
- Lenguaje de bajo nivel (ensamblador);
- Lenguaje de alto nivel.

Instrucciones a la computadora

El término instrucción (operación realizable) se suele referir a los lenguajes máquina y bajo nivel, reservando el término sentencia o proposición para los lenguajes de alto nivel.
Las instrucciones básicas y comunes a todos los lenguajes de programación se pueden condensar en cuatro grupos:
- **Instrucciones de entrada / salida**: instrucciones de transferencia de información entre dispositivos periféricos y la memoria central;
- **Instrucciones aritmético / lógicas**: instrucciones que ejecutan operaciones aritméticas y lógicas;
- **Instrucciones selectivas**: instrucciones que permiten la selección de tareas alternativas en función de los resultados de diferentes expresiones condicionales;
- **Instrucciones repetitivas**: Instrucciones que permiten la repetición de secuencias de instrucciones un número determinado o indeterminado de veces.

El que se lleve a cabo una instrucción se denomina ejecución.

Lenguaje máquina

Son aquellos que están escritos en lenguajes directamente inteligibles por la máquina, ya que sus instrucciones son cadenas binarias que especifican una operación, y las posiciones (dirección) de memoria implicadas en la operación se denominan instrucciones de máquina o código máquina.

Las *ventajas* de programar en lenguaje máquina es la posibilidad de cargar (transferir un programa a la memoria) sin necesidad de traducción posterior, lo que supone una velocidad de ejecución superior a cualquier otro lenguaje de programación.

Los *inconvenientes* que hacen que este lenguaje no sea recomendado son: dificultad y lentitud en la codificación, poca fiabilidad, dificultad grande de verificar y poner a punto los programas, y que los programas sólo son ejecutables en el mismo procesador.

Lenguaje de bajo nivel

Es el ensamblador. También depende de la máquina en particular. Las instrucciones en este lenguaje se conocen como *nemotécnicos*.

Requiere una fase de traducción al lenguaje máquina.

El programa original escrito en lenguaje ensamblador se denomina *programa fuente*, y el traducido en lenguaje máquina se conoce como *programa objeto*.

No se debe confundir el programa ensamblador (assembler), encargado de efectuar la traducción a lenguaje máquina del programa fuente escrito, con el lenguaje ensamblador (assembly languaje), lenguaje de programación con una estructura y gramática definidas.

Los lenguajes ensambladores presentan una *ventaja* frente a los lenguajes de máquina por su mayor facilidad de codificación y su velocidad de cálculo.

Los *inconvenientes* son su dependencia total de la máquina, lo que impide ejecutar un programa en diferentes máquinas; y la formación de los programadores, ya que exige no sólo las técnicas de programación sino también el conocimiento interior de la máquina.

Lenguaje de alto nivel

Es independiente de la máquina, las instrucciones del programa de la computadora no dependen del diseño del hardware. En consecuencia, los programas escritos en este lenguaje son portables o transportables.

Ventajas:

-El tiempo de formación de los programadores es relativamente corto comparado con otros lenguajes;

-La escritura de programas se basa en reglas sintácticas similares a los lenguajes humanos;

-Las modificaciones y puestas a punto de los programas son mas fáciles;

-Reducción del coste de los programas;

-Transportabilidad.

Inconvenientes:

-Incremento del tiempo de puesta a punto, al necesitarse traducciones del programa fuente para conseguir el programa definitivo;

-No se aprovechan los recursos internos de la máquina, que se explotan mucho mejor en lenguajes máquina y ensambladores;

-Aumento de la ocupación de la memoria;

-El tiempo de ejecución de los programas es mucho mayor.

Cada lenguaje de programación está diseñado bajo un **paradigma** o como combinación de componentes de mas de un paradigma.

Paradigma de programación

Definición

Conjunto de patrones conceptuales que moldean la forma de resolver y pensar un problema, diseñar un algoritmo y estructurar un programa. Es decir, va a ser tenido en cuenta en el armado del programa.

Se lo puede entender como un estilo de programación marcado como un conjunto de criterios para manejar el programa.

Clasificación general

-Grupo de paradigmas con efectos laterales: Imperativo, ensamblador, orientado a objetos;
-Grupo de paradigmas declarativos: funcional, lógico.

P. con efectos laterales

Se basan en la estructura interna de una pc., en particular en el modelo de Von Newmann (procesador, memoria y otros accesorios). Se basan en el cambio de estado (contenido) de entidades; el más cercano a como internamente la pc. maneja los datos es el **ensamblador**, propio del procesador; el **imperativo** trabaja los datos a través de variables; el **orientado** es mas abstracto (se aleja del estado interno de la pc.), trabaja encapsulando datos teniendo en cuenta las acciones de estos datos.

P. declarativos

Se basan en fundamentos matemáticos, no tienen en cuenta la estructura interna de la pc. (sistema binario: números/caracteres). el **fucional** se basa en que su célula no es la variable sinó la función; el **lógico** se basa en proposiciones lógicas (inteligencia artificial).

*// ¿cómo pensar un problema?**:** dentro del paradigma imperativo. //*

Traductores de lenguaje

Los traductores de lenguaje son programas que traducen a código máquina los programas fuente escritos en lenguajes de alto nivel; y se dividen en compiladores e intérpretes.
Un *intérprete* es un traductor que toma un programa fuente, lo traduce y a continuación lo ejecuta.
Un *compilador* es un programa que traduce a lenguaje máquina los programas fuente escritos en lenguajes de alto nivel.

La compilación y sus fases

La compilación es el proceso de traducción de programas fuente a programas objeto. El programa objeto obtenido de la compilación ha sido traducido normalmente a código máquina.
Para conseguir el programa máquina real se debe utilizar un programa en lenguaje máquina directamente ejecutable.

//¿cómo va a ser referenciado un problema?: a través de una variable.//

DATOS, TIPOS Y ESTRUCTURA DE DATOS, OPERACIONES PRIMITIVAS.

El primer objetivo de toda computadora es el manejo de la información o datos. Un dato es la expresión general que describe los objetos con los cuales opera una computadora.

//DATO NO ES VALOR: El dato es el concepto (Ej.: *nombre_del_dia*); valor es el contenido asociado al dato (Ej.: *martes*). *//*

Existen dos clases de tipos de datos: simples (sin estructura) y compuestos (estructurados).
Un dato es un conjunto o secuencia de bits.

Tipos de datos simlples o primitivos

_Numéricos: -Entero
-Real

_Caracter: -Cadena de caracteres: string >> "el"
-Cadena nula: " "
-Caracter en blanco: ~~b~~

_Lógico: -Verdadero
-Falso

Los tipos de datos primitivos van a tener operaciones válidas, expresadas con la sintaxis admitida:

_Numéricos

+ - * / **

(orden >> comparaciones; Ej.: entero (3,8)= 3)

_Caracter

(orden >> comparaciones; Ej.: numb1>numb2)

-Concatenación de cadenas (unión) +
(no es conmutativo)
dia:= "lunes"
tex:= "hoy~~b~~es~~b~~"
nombredia:= tex + dia

-Subcad (T.P.C)
Permite como resultado una subcadena dentro de un valor que puede estar dado por una expresión T, luego explicar con que posición P, y la cantidad de caracteres C que me interesa extraer.
leer: dia
tex:= "hoy~~b~~es~~b~~"
nombredia:= tex + dia
imprimir: subcad (nombredia, 8, 6) // Es decir, va a imprimir a partir de la posición 8 inclusive, los 6 caracteres que se encuentran a continuación: va a imprimir sólo el nombre del día. //

-Long (T)
Cantidad de caracteres que conforman T.
leer: nombredia
imprimir: subcad (nombredia, 8, long(nombredia) - 7)

_Lógicos

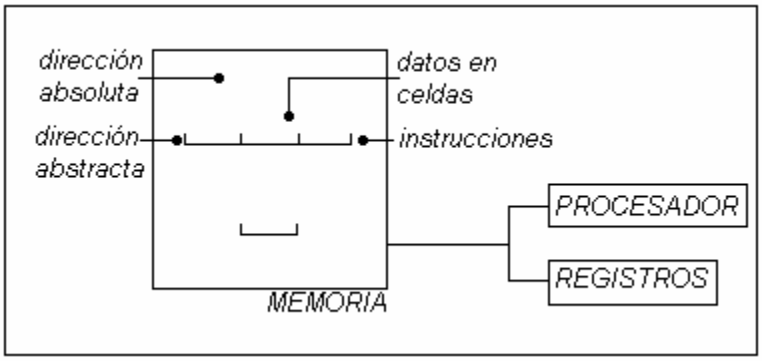
Las operaciones que se pueden realizar no son de orden, sino de igualdad / desigualdad:
d1 <> d2 (distintos)
r = x

funciones primitivas
and (y)
or (o)
not (negación)

Constantes y variables

Una *constante* es un componente que va a tener un identificador y un valor asignado en la declaración, a través del cual se conoce el tipo de dato. Es un valor constante: no se va a poder modificar durante todo el transcurso del programa.

Una *variable* es una referencia abstracta de una zona de memoria en donde se guarda / almacena un dato; su valor puede cambiar durante el desarrollo del algoritmo y / o durante la ejecución del programa.



//Memoria: conjunto de celdas o zonas referenciadas mediante direcciones absolutas, que se manejan en el paradigma ensamblador.//

Variable: -Identidad (declaración);
-Tipo (declaración);
-Valor (ejecución).

Constante: -identificador;
-Valor (tipo).

Expresiones

Las *expresiones* son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales. Una expresión consta de operandos y operadores, y según sea el tipo de objetos que manipulan, las expresiones se clasifican en aritméticas, lógicas y carácter.

Funciones internas

Las operaciones que se requieren en los programas exigen en numerosas ocasiones, además de las operaciones aritméticas básicas, un número determinado de operadores especiales que se denominan *funciones internas*, incorporadas o estándar.
Cuando se le envía una cierta cantidad de valores a una función, ésta tiene como objetivo devolver un único valor, el cuál le es asignado a una variable.

La operación de asignación: instrucción básica

La operación de asignación (dentro del Paradigma Imperativo) es el modo de asignarle valores a una variable. Esta operación se conoce como instrucción o sentencia de asignación cuando se refiere a un lenguaje de programación.
La nomenclatura en el lenguaje pseudocódigo es la siguiente:

<var>:=<exp> // nombre:=expresión //

EJ1) *Desarrollar el algoritmo en pseudocódigo que me permita calcular e informar el promedio de dos valores numéricos ingresados.*

Programa: promedio de 2 valores

Variables
(valor1, valor2, prom) real 8,2

Hacer
Imprimir: "ingrese datos"
Leer: valor1, valor2
prom:=(valor1 + valor2)/2 // << asignación.//
Imprimir: "promedio=", prom
Fin Hacer

Fin Programa

Cuando se quiere asignar un valor a una variable de determinado tipo (Ej.: *variable de tipo real*) no se puede asignar uno que no sea de ese mismo tipo.

Entrada y salida de información

Los cálculos que realizan las computadoras requieren la *entrada* de los datos necesarios para ejecutar las operaciones, que posteriormente se convertirán en resultados, es decir, *salida*.
Esta entrada se conoce como operación de *lectura.* Los datos de entrada se introducen al procesador mediante los dispositivos de entrada.
La salida puede aparecer en un dispositivo de salida. La operación de salida se denomina *escritura*.
Estas acciones se representan por los formatos siguientes:

Leer: (lista de variables de entrada)

Escribir: (lista de expresiones de salida) o, en nuestro caso:

Imprimir: (lista de expresiones de salida)

CAPITULO III

RESOLUCION DE PROBLEMAS CON COMPUTADORAS HERRAMIENTAS DE PROGRAMACIÓN

La resolución de problemas con computadoras se puede dividir en tres fases:

- Análisis del problema
- Diseño del algoritmo
- Resolución del algoritmo en la computadora

El primer paso requiere que el problema sea definido y comprendido claramente para que pueda ser analizado con todo detalle. Una vez analizado el problema, se debe desarrollar el algoritmo. Por último, para resolver el algoritmo mediante una computadora se necesita codificar el algoritmo en un lenguaje de programación.

Análisis del problema

El propósito del análisis del problema es ayudar al programador para llegar a una cierta comprensión de la naturaleza del problema. El problema debe estar bien definido si se quiere llegar a una solución satisfactoria.
Para poder definir con precisión el problema se requiere que las especificaciones de entrada y salida sean descritas en detalle.
Éstos son los requisitos mas importantes para llegar a una solución eficaz.

Diseño del algoritmo

La descomposición del problema original en subproblemas mas simples y a continuación dividir estos subproblemas en otros mas simples, que pueden ser implementados para su solución en la computadora, se denomina *diseño descendente* (top-down design). Este método se suele denominar también *divide y vencerás*.
Para problemas complejos se necesitan con frecuencia diferentes *niveles de refinamiento* antes de que se pueda obtener un algoritmo claro, preciso y completo.
Las ventajas mas importantes del diseño descendente son:

- El problema se comprende mas fácilmente al dividirse en partes mas simples, denominadas *módulos;*
- Realizarle modificaciones al algoritmo es mas fácil;
- La comprobación de la solución del problema se puede verificar fácilmente.

Estructura inicial del algoritmo:

- Las tareas deben estar seguidas de alguna secuencia definida se pasos hasta que se obtenga un resultado coherente;
- El flujo de control usual de un algoritmo es secuencial;
- Un aspecto importante a considerar es el método elegido para describir los algoritmos: empleo de justificación en la escritura de algoritmos.

Resolución del problema mediante computadoras

Esta fase se descompone en las siguientes subfases:

- Comprobación del algoritmo en un programa
- Ejecución del programa
- Comprobación del programa

La fase de conversión del algoritmo en un lenguaje específico de programación se denomina *codificación*, y el algoritmo resultante se denomina *código.*
Tras la codificación del programa, el mismo deberá ejecutarse en una computadora y a continuación de comprobar los resultados, pasar a la fase final de documentación.

Representación gráfica de los algoritmos

Se debe utilizar algún método que permita independizar el algoritmo del lenguaje de programación elegido. Ello permitirá que un algoritmo pueda ser codificado indistintamente en cualquier lenguaje.
Los métodos usuales para representar un algoritmo son:

- Diagrama de flujo
- Diagrama N-S (Nassi-Schneiderman)
- Lenguaje de especificación de algoritmos: Pseudocódigo
- Lenguaje español
- Fórmulas

Un **diagrama de flujo** (flowchart) es un diagrama que utiliza símbolos (cajas) estándar y que tiene los pasos del algoritmo escritos en esas cajas, unidas por flechas denominadas *líneas de flujo*, que indican la secuencia en que se deben ejecutar.
Los símbolos estándar normalizados por ANSI (American National Standars Institute) son muy variados.

El **diagrama N-S**, también conocido como *diagrama de chaplín*, es como un diagrama de flujo en el que se omiten las líneas de flujo y las cajas son contiguas. Las acciones sucesivas se escriben en cajas sucesivas y, como en los diagramas de flujo, se pueden escribir diferentes acciones en una caja.

El **pseudocódigo** es un lenguaje de especificación (descripción) de algoritmos. El uso de tal lenguaje hace el paso de codificación final relativamente fácil.
Se considera como un primer borrador, dado que el pseudocódigo tiene que traducirsa posteriormente a un lenguaje de programación.
La ventaja del pseudocódigo es que, en la palnificación de un programa, el programador se puede concentrar en la lógica y en las estructuras de control, y no preocuparse por las reglas de un lenguaje específico. Además, es fácil de modificar si se encuentran errores.
Es muy importante añadir *comentarios* durante la utilización de éste o cualquier otro lenguaje. El comentario brinda información acerca de las acciones que realiza el programa, y no realiza ninguna instrucción ejecutable, sólo tiene efecto de documentación interna del programa.

CAPITULO IV

ESTRUCTURA GENERAL DE UN PROGRAMA

CONCEPTO DE PROGRAMA

Un programa es un conjunto de instrucciones (órdenes) que producirán la ejecución de una determinada tarea; es la conversión del algoritmo haciendo uso de los constructores formales de un lenguaje de programación para ser ejecutado en una pc.

Atributos, características

-debe estar **definido**: único punto de entrada/salida, que se comprenda la secuencia de las acciones;
-debe ser **confiable**;
-**eficiente**, en cuanto al uso de los recursos de la mejor manera posible;

// Recursos:
-tiempo (análisis de cantidad de acciones/operaciones);
-espacio (asociado a la cantidad de datos a considerar en el algoritmo). //

-debe ser **claro**, tener una **documentación**;
-debe ser **portable**: permitir ser ejecutado en distintas pcs con un mínimo de cambios.

El proceso de programación es un proceso de solución de problemas, y el desarrollo de un programa requiere las siguientes fases:
1-Definición y análisis del problema.
2-Diseño de algoritmos:
-diagrama de flujo
-diagrama N-S
-pseudocódigo
3-Codificación del programa.
4-Depuración y verificación del programa.
5-Documentación externa.
6-Mantenimiento.

En todos los casos anteriores, llevar a cabo simultáneamente la adecuada *documentación interna*.

PARTES CONSTITUTIVAS DE UN PROGRAMA

Tras la decisión de desarrollar un programa, el programador debe establecer el conjunto de especificaciones que debe contener el programa: entrada, salida y algoritmos de resolución, que incluirá las técnicas para obtener las salidas a partid de las entradas.

Instrucciones y tipos de instrucciones

Un programa puede ser lineal o no lineal:

- Es *lineal* si las instrucciones se ejecutan **secuencialmente**, sin bifurcaciones, decisiones ni comparaciones;
- Es no lineal cuando se interrumpe la secuencia mediante instrucciones de bifurcación.

La clasificación mas usual es:

- Instrucciones de inicio / fin.
- Instrucciones de asignación: dar valores a una variable.
- Instrucciones de lectura: leer datos de un dispositivo de entrada.
- Instrucciones de escritura.

- Instrucciones de bifurcación.

Instrucciones de bifurcación

Pueden ser hacia delante o hacia atrás, y pueden realizarse de manera condicional o incondicional:
-*Bifurcación incondicional*: se realiza siempre que el flujo del programa pase por la instrucción sin necesidad del cumplimiento de ninguna condición.
-*Bifurcación condicional*: que se ejecute o no, depende del cumplimiento de una determinada condición.

Elementos básicos de un programa

Los lenguajes de programación tienen elementos básicos que se utilizan como bloques constructivos, así como reglas para que estos elementos se combinen; esas reglas se denominan sintaxis del lenguaje. Solamente las instrucciones sintácticamente correctas pueden ser interpretadas por la computadora, y los programas que contengan errores de sintaxis serán rechazados por la máquina.

Los elementos básicos constitutivos de un programa o algoritmo son:

- palabras reservadas (“hacer”, “fin hacer”, “si” >> “entonces”...)
- identificadores (nombres de variables esencialmente)
- caracteres especiales (coma, apóstrofe...)
- constantes
- variables
- expresiones
- instrucciones

Además de estos elementos básicos, existen otros que forman parte de los programas:

- bucles.
- acumuladores.
- contadores.
- interruptores.
- **estructuras de control**: secuenciales, selectivas, repetitivas.

Bucles

Un *bucle* o *lazo* (loop) es un segmento de un algoritmo o programa, cuyas instrucciones se repiten una n cantidad de veces mientras se cumpla una determinada condición.
Se debe establecer un mecanismo para determinar las tareas repetitivas; este mecanismo es una condición que puede ser verdadera o falsa, y que se comprueba cada vez que se realiza un paso o iteración del bucle.
Un bucle consta de tres partes:
-decisión
-cuerpo
-salida
Los bucles son *anidados* cuando están dispuestos de tal modo que unos son interiores a otros, y son *independientes* cuando son externos unos de otros.

Contadores y acumuladores

VARIABLE ACUMULADORA: Su función es acumular valores, distintos o no, en forma parcial; valores resultantes de sumas sucesivas. Se debe inicializar siempre, de lo contrario el valor que esta tome va a incluir el acumulado de todas las veces que corrió el programa. Generalmente se inicializa al comienzo o casi al final del programa. El modo de acumularle valores es siempre el mismo:

<var acumuladora>:= <var acumuladora> + <var a sumar >

>> El incremento o decremento de cada suma es **variable**.

VARIABLE CONTADORA: Es una variable cuyo valor se incrementa o decrementa en una unidad, en cada iteración.

<var contadora>:= <var contadora> + <constante>

>> El incremento o decremento de cada suma es **constante**.

Interruptores

Un interruptor o conmutador (switch) – llamado a veces, indicador o bandera (flag) – es una variable que puede tomar diversos valores a lo largo de la ejecución del programa, y que permite comunicar información de una parte a otra del mismo.

Estructura de algoritmos / programas

Un algoritmo constará de dos componentes: una cabecera de programa y un bloque algoritmo.
La *cabecera de programa* es una acción simple que comienza con la palabra “algoritmo” o ”programa”. Esta palabra estará seguida por el nombre completo asignado al programa.
El *bloque algoritmo* es el resto del programa, y consta de dos componentes o secciones: las acciones de declaración y las acciones ejecutables.
Las *declaraciones* definen o declaran las variables y constantes que tengan nombre .
Las *acciones ejecutables* son las acciones que posteriormente deberá realizar la pc cuando el algoritmo convertido en programa se ejecute.

Comentarios

La documentación de un programa es el conjunto de información interna y externa al programa, que facilitará su posterior mantenimiento y puesta a punto.
La *documentación interna* es la que acompaña en el código o programa fuente y se realiza a base de comentarios significativos. Estos comentarios se representancon diferentes notaciones, según el tipo de lenguaje de programación.
La *documentación externa* se realizará con información ajena al programa, y será proporcionada por el programador.

CAPITULO V

INTRODUCCION A LA PROGRAMACION ESTRUCTURADA

Las nuevas teorías de programación se centran en las técnicas de programación modular y programación estructurada, de modo que se pueda conseguir un programa eficaz.

Programación modular

Una estrategia para la resolución de un problema complejo con computadoras, es la división o descomposición del mismo en subproblemas más pequeños. Estos subproblemas se implementan mediante módulos o subprogramas. Los subprogramas son una herramienta importante para el desarrollo de algoritmos y programas, ya que un proyecto de programación normalmente se compone de un programa principal, y de un conjunto de subprogramas, los cuales son llamados o invocados desde el programa principal.

En la programación modular, el programa se divide en módulos, cada uno de ellos independiente de los restantes, que ejecutan cada uno una única actividad o tarea. Cada uno de ellos se analiza, codifica y pone a punto por separado.
Cada programa contiene un módulo denominado programa principal que controla todo lo que sucede; de allí se transfiere el control de ejecución a submódulos de modo que ellos puedan ejecutar sus funciones o procedimientos; sin embargo, cada submódulo devuelve el control al módulo principal cuando se haya completado su tarea. Si la tarea asignada a cada submódulo es demasiado compleja, éste deberá dividirse en otros módulos mas pequeños. El proceso sucesivo de subdivisión de módulos continúa hasta que cada módulo tenga una tarea específica que ejecutar.
Los módulos son independientes en el sentido en el que ningún módulo puede tener acceso directo a cualquier otro módulo, con excepción del módulo al que llama o invoca, y sus propios submódulos. Sin embargo, los resultados producidos por un módulo pueden ser utilizados por cualquier otro módulo cuando se transfiere a ellos el control de ejecución.

Programación estructurada

El término programación estructurada se refiere a un conjunto de técnicas que aumentan la productividad del programa reduciendo el elevado tiempo requerido para escribir, verificar, depurar y mantener los programas.
La programación estructurada es el conjunto de técnicas que incorporan:

- Diseño descendente (Top-Down).
- Recursos abstractos.
- Estructuras básicas.

DISEÑO DESCENDENTE

El *diseño descendente* es el proceso mediante el cual un problema se descompone en una serie de niveles o pasos sucesivos de refinamiento (stepwise). La metodología descendente consiste en efectuar una relación entre las sucesivas etapas de estructuración, de modo que se relacionen unas con otras mediante entradas y salidas de información.

RECURSOS ABSTRACTOS

Descomponer un programa en términos de *recursos abstractos* consiste en descomponer una determinada acción compleja en función de un número de acciones mas simples, capaces de ser ejecutadas por una computadora, y que constituirán sus instrucciones.

TEOREMA DE LA PROGRAMACIÓN ESTRUCTURADA: ESTRUCTURAS BÁSICAS

Un *programa propio* puede ser escrito utilizando solamente tres tipos de estructuras de control:

- **Secuencial:** En el mismo orden en forma imperativa, no depende del contenido.
- **Repetición:** Llevar a cabo la ejecución de un conjunto de funciones una determinada cantidad de veces.
- **Decisión:** Mediante estructura lógica >> saber si se realiza una función o no.

Un programa se define como propio si cumple las siguientes características:

-Posee un solo punto de entrada y uno de salida, o fin, para control del programa;
-Existen caminos desde la entrada hasta la salida que se pueden recorrer y que pasan por todas las partes del programa.
-Todas las instrucciones son ejecutables y no existen lazos o bucles infinitos.

ESTRUCTURA DE CONTROL SECUENCIAL

La estructura secuencial es aquella en la que una acción sigue a otra en secuencia. La salida de una es la entrada de la siguiente.

EJ2) Calcular el sueldo (líquido) recibiendo el valor del básico, descuento del 10% por IPS y el 4% de IOMA.

Programa: Sueldo Líquido

```
Constantes
IOMA: 4
IPS: 10

Variables
(basico, impips, impioma, liq) real 6,2          //4 enteros, 2 decimales//

Hacer
  Imprimir: "Ingresar basico"
  Leer: basico
  impips:= basico * IPS / 100                    //cálculo de porcentajes //
  impioma:= basico * IOMA / 100
  liq:= basico - impips - impioma
  Imprimir: "sueldo líquido=", liq
Fin Hacer

Fin Programa
```

ESTRUCTURAS DE CONTROL REPETITIVAS

Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan bucles; y se llama iteración al hecho de repetir la ejecución de una secuencia de acciones.

-Repetición

-Incondicional >> fija: // no utilizar incondicional cuando la cantidad de repeticiones no será fija. //

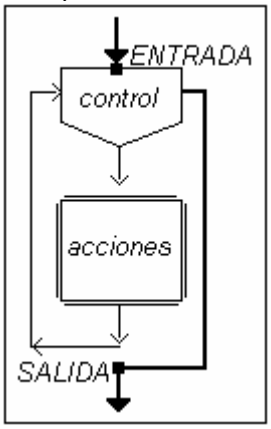
-constante (entero)

-variable (entero)

-Condicional >> indeterminada.

Formato gráfico de repetición

(ya sea condicional o incondicional)



Las estructuras se basan en 2 criterios:

1. Si la cantidad de repeticiones es **fija**;
2. Si la cantidad de repeticiones es **indeterminada**, generalmente porque depende de los datos

Dependiendo de cual de las 2 formas de **control** tenga, va a estar conformada por 2 criterios de control de datos respectivamente:

1. variable de control (rango);
2. condición (expresión lógica).

Repetición incondicional; "Repetir Para"

Esta estructura (For) ejecuta las acciones del cuerpo del bucle un número especificado de veces, y de modo automático controla la cantidad de iteraciones o pasos a través del cuerpo del bucle. En pseudocódigo:

SINTAXIS

Repetir Para : <variable de control> := <desde>, <hasta>, <paso>
...
..
..(instrucciones)
..
...
Fin Repetir Para

DESCRIPCIÓN

<variable de control> : Es una variable de tipo Entero, que nos permite controlar la repetición de la secuencia de instrucciones que se encuentra dentro de la estructura de control.

<desde> : tipo Entero; nos indica el valor inicial de la repetición, puede llegar a ser un valor constante, una variable, o una operación.

<hasta> : tipo Entero; nos indica el valor hasta el cual nosotros queremos llegar con la repetición. Puede llegar a ser un valor constante, una variable o una operación.

<paso> : tipo Entero; nos indica el paso de incremento de la variable de control, generalmente es de a uno (se le suma uno a la variable de control), pero se podrían plantear pasos de mas de uno, e inclusive pasos negativos (decrementar la variable de control).

Nota: el Paso puede ser positivo o negativo.

- *Si es positivo, Hasta debe ser mayor o igual que Desde;*
- *Si es negativo, Hasta debe ser menor o igual que Desde.*

Paso no puede ser nulo; puede omitirse, en cuyo caso se asume el valor de paso igual a 1.

Las instrucciones que se encuentran dentro del bucle se ejecutan una vez para cada valor de la variable de control, desde el valor inicial hasta el valor final, con el incremento de a pasos.

Ejemplos de control

a) I =1,100,1 >> 1,2,3...99,100 |*101*

b) P = 2,100,2 >> 2,4,6...100 |*102*

c) D = 1,10 >> 1,2...9,10 |*11* //Por defecto, el paso es 1 //

d) F = 20,1,**-1** >> 20,19,18...1 |*0*

e) X = 1, totsillas (0,99) >> -si totsillas= 5 >> 1,2,3,4,5

-si totsillas=Ø >> **?**

EJ3) Una empresa tiene 13 sucursales de las cuales se informa la cantidad de facturas confeccionadas entre 0 y 99 con hasta 99 artículos en cada una, con un importe total máximo de $10.000 dentro del cual se cobran $13 del flete.
Informar:
-cantidad total de facturas
-cantidad de artículos vendidos por sucursal
-importe a reservar para amortizar el flete.

***Análisis**:*
-13 sucursales
-c/u entre 0 y 99 facturas
- c/ factura entre 1 y 99 artículos
-máximo de importe $10.000
-flete $13

***Diseño**:*
(variables)
cantfactot //será necesario inicializarla en 0 //
recauda //inicializar //
impfac //var. de entrada: lectura//
artfac //entrada //
cantfacsuc //entrada //
cantartsuc //inicializar //
impflete //var. de salida: impresión //

(constantes)
cantsuc = 13
costoflete = 13

Codificación

Programa: ventas

Constantes
CANTSUC=13
COSTOFLETE=13

Variables
(ISUC, IFAC) entero 3 // índices: sucursal y factura //
(artfac, cantfacsuc) entero 2 // articulos x factura, cant de facturas en sucursal //
cantfactot entero 4 // cantidad de facturas en total //
(imfac, cantartsuc, impflete) entero 5 // importe factura, cant.artículos sucursal, importe flete//
recauda entero 8 // recaudación //

Hacer
cantfactot:= 0
recauda:= 0

Repetir Para: ISUC:= 1,CANTSUC
cantartsuc:= 0
Leer: CANTFACSUC

Repetir Para: IFAC:= 1,CANTFACSUC
Leer: imfac, artfac
recauda:= recauda + imfac
cantartsuc:= cantartsuc + artfac
Fin Repetir Para

cantfactot:= cantfactot + cantfacsuc
Imprimir: cantfactot, recauda
Fin Repetir Para

Fin Hacer

Fin Programa

PREDICADO

Definición
Es una **condición lógica o expresión lógica**.

-Expresión lógica
-Simple (sin conectores lógicos); // Conectores lógicos: -and
-Compuesta (uso de conectores). -or //

-Variable lógica
-Verdadero
-Falso

-Constante lógica:
-Verdadero
-Falso

Ejemplos

1. (resultado >= 4)
2. (libre = verdadero)
3. ((x >= I) and (x <= J)) and (tope < 5) //resultará verdadero o falso//
   // I <= x <= J //

Repetición condicional; "Repetir Mientras"

La estructura repetitiva mientras (while) es aquella en la cual el cuerpo del bucle se repetirá mientras se cumpla una determinada condición. En pseudocódigo:

SINTAXIS

Repetir Mientras (<condición o condiciones compuestas>**)**
...
..
..(instrucciones)
..
...
Fin Repetir Mientras

DESCRIPCIÓN

<condición o condiciones compuestas> : En este sector de la estructura, se debe expresar la condición que nos indique que mientras permanezca verdadera, se siguen repitiendo las instrucciones dentro del bucle; se observa que no hay una repetición fija, sino que está condicionada, no se sabe cuantas veces se va a repetir.

-Condicional:
-Verdadero >> repite otra vez
-Falso >> deja de repetir

Bucles infinitos

Algunos bucles no exigen fin y otros no encuentran el fin por algún error en su diseño. Un bucle que nunca termina se denomina bucle infinito. Los bucles sin fin, no intencionados, son perjudiciales para el programa, y se deben evitar siempre.

Regla práctica: Es conveniente que las comparaciones en las expresiones booleanas sean por mayor o menor, en lugar de ser de igualdad o desigualdad. En el caso de la codificación en un lenguaje de programación, esta regla debe seguirse rígidamente a la horade comparar números reales, ya que como estos valores se almacenan en cantidades aproximadas, las comparaciones de "igualdad de valores reales" normalmente plantean problemas. Siempre que realice comparaciones de números reales, utilice <, <=, > ó >=.

Terminación de bucles mediante datos de entrada

Si el programa está leyendo una lista de valores con un bucle condicional, se debe incluir algún tipo de mecanismo para terminar el bucle. Existen cuatro métodos típicos para terminar un bucle de entrada:

- Preguntar antes de la iteración;
- Finalizar la lista con su valor de entrada;
- Agotar los datos de entrada.

El primero simplemente pregunta mediante un mensaje al usuario si existen mas datos de entrada. Tal vez el método mas correcto para terminar un bucle que lee una lista de valores es con un *centinela.* Un valor centinela es un valor especial usado para indicar el fin de una lista de datos. Éste debe leerse al final del bucle, por lo que se debe dar la instrucción "Leer:" al final del mismo.
El último método es comprobar simplemente que no existen mas datos de entrada. Este sistema suele depender del tipo de lenguaje.

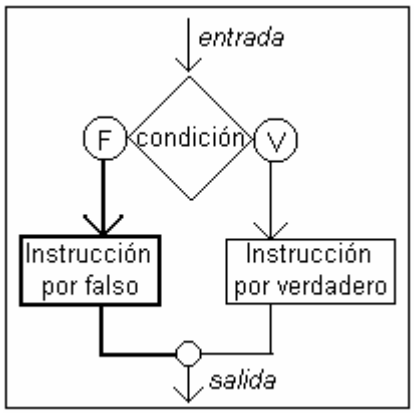
Estructuras repetitivas anidadas

Es posible insertar un bucle dentro de otro. La estructura interna debe estar incluida totalmente dentro de la externa, y no puede existir solapamiento.
Las variables de control de los bucles toman valores de modo tal que, por cada valor de la variable índice del ciclo externo, se debe ejecutar totalmente el bucle interno.

ESTRUCTURAS DE CONTROL SELECTIVAS

Las estructuras selectivas se utilizan para tomar decisiones lógicas; por eso se suelen denominar también estructuras de decisión o alternativas.
En las estructuras selectivas se evalúa una condición, y en función del resultado de la misma, se realiza una opción u otra; las condiciones se especifican usando expresiones lógicas.

Formato gráfico de decisión



Las estructuras selectivas pueden ser simples, dobles o múltiples.

Alternativa simple

Si la condición es verdadera, ejecuta una determinada acción; si es falsa, no hace nada. En pseudocódigo:

Si: <condición> Entonces
..
...(instrucciones)
..

Fin Si

Alternativa doble

Esta estructura permite elegir entre dos opciones posibles, en función del cumplimiento o no de una determinada condición. En pseudocódigo:

Si: <condición> Entonces
..
...(instrucción por verdadero)
..

Sino
..
...(instrucción por falso)
..

Fin Si

// Promedio: Σ / n°

Porcentaje: (n°1) * (n°2) / 100 //

EJ4) Se realiza una prueba en la que participan 300 alumnos; 30 alumnos por cada uno de los 10 colegios invitados. Se debe controlar la entrega de los resultados informando nota y colegio por cada uno, para interrumpir dicha recepción al terminar el décimo alumno del colegio de orden 5to. Informar:
-Promedio de notas de los alumnos del colegio de orden 5to;
-Porcentaje de alumnos registrados de otros colegios durante la recepción sobre el total de recibidos.

Análisis
10 colegios
30 alumnos x colegio } 300 // 280 tope teórico //
datos de ingreso:
-nota (de 0 a 10)
-colegio (de 1 a 10)
datos de salida:
-promedio
-porcentaje

Codificación

Programa: Prueba entre colegios

```
  Variables
  (nota, PROM) real 4,2
  colegio entero 2
  PORC real 5,2
  acumnota real 6,2    // variable acumuladora //
  cantcol5 entero 2     // variable contadora //
  cantotros entero 3   // contadora //

  Hacer
    acumnota:= 0
    cantcol5:=  0
    cantotros:= 0

    Repetir Mientras "ingresar nota y colegio"            //sin desarrollar//
      Leer: nota, colegio

      Si: (colegio = 5) Entonces
                                  cantcol5:= cantcol5 + 1
                                  acumnota:= acumnota + nota

                    Sino
                                  cantotros:= cantotros + 1
      Fin Si

    Fin Repetir Mientras

    PROM:= acumnota / cantcol5
    PORC:= cantotros / (cantotros + cantcol5) * 100
    Imprimir: "promedio=", PROM, "porcentaje=", PORC

  Fin Hacer

Fin Programa
```

Alternativa múltiple: CASO

La estructura de decisión múltiple evaluará una expresión que podrá tomar n valores distintos; según el valor que tome, se realizará una de las n acciones.
La debo utilizar en una situación en la que se deben realizar diferentes acciones que dependen de una variable. El objetivo es eliminar el anidamiento excesivo.
En pseudocódigo:

```
Caso <variable>
    (condición 1): (instrucciones)
    (condición 2): (instrucciones)
    ....
    (condición n): (instrucciones)

    En otro caso:
            ....
            ....(instrucciónes)
            ....
Fin Caso
```

EJ5) Realzar un programa en el que se ingresen cantidades de materia prima (frutas) y éste la acumule, teniendo en cuenta que existen 5 tipos distintos: A, B, C, y D, esta última se debe destinar a 2 sectores: el 30% para jugos y el resto para mermeladas.
El máximo de frutas que ingresa por carga es de 1 tonelada, y los contenedores en total pueden incluir sólo 8 toneladas, aunque nunca pasan las 4,5 ton.
Cada 2 días, se debe realizar un informe detallando la cantidad de frutas de cada tipo, y luego empezar a contar de cero nuevamente.

Análisis
Máximo p/ carga 1000 kg
Máximo contenedores 8000 kg // 5000 tope teórico //
tipos:
A
B
C
D: 30% jugo
70% mermelada

Datos de entrada:
N° de día (1, 2 ó 3), tipo de fruta y cantidad.

Datos de salida:
Informe cada 2 días.

Diseño
carga // lectura //
conten // acum. Va a chequear que el contenedor no pase las 8 ton.//
(A, B, C) // acum. //
jugo // acum. //
merme // acum. //
dia // lectura //
tipo //lectura//

Codificación

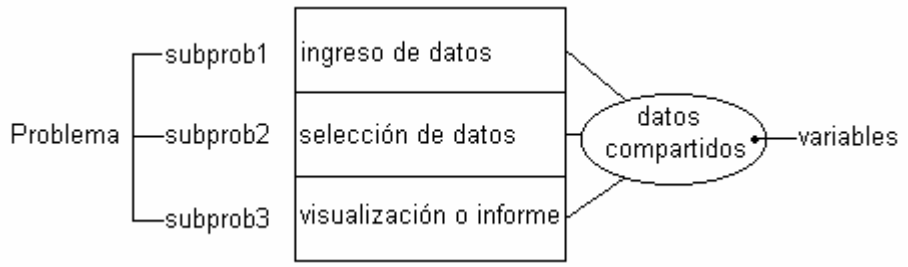
Programa: Frutas

```
  Variables
  carga entero 4
  conten entero 4
  (A, B, C) entero 4
  (jugo, merme) entero 4
  dia entero 1
  tipo caracter 1

  Hacer
    conten:= 0
    A:= 0
    B:= 0
    C:= 0
    jugo:= 0
    merme:= 0
    dia:= 1                          //obligo al programa a ejecutarse al menos 1 vez//

    Repetir Mientras (dia <3)
      Imprimir: "ingrese carga en kg."
      Leer: carga
      Imprimir: "ingrese calidad"
      Leer: tipo
      conten:= conten + carga
      Si: (conten=8000) Entonces
                                  Imprimir: "Los contenedores han llegado a su límite."
                          Sino
                              Si: (conten > 8000) Entonces
                                                      Imprimir: "Los contenedores han
          //anidamiento//                                       sobrepasado su capacidad!"
                              Fin Si
      Fin Si

      Caso tipo
        (tipo="A"): A:= A + carga
        (tipo="B"): B:= B + carga
        (tipo="C"): C:= C + carga
        (tipo="D"): jugo:= jugo + (30 * carga / 100)                    // 30 % //
                    merme:= merme + (70 * carga / 100)
      Fin Caso

      Imprimir: "ingrese el número de día"
      Leer: dia
    Fin Repetir mientras

          Imprimir: "cantidad de frutas tipo A=", A,"; tipo B=", B,"; tipo C=", C,"frutas para jugo=",
          jugo,"; para mermelada=", merme

  Fin Hacer

Fin Programa
```

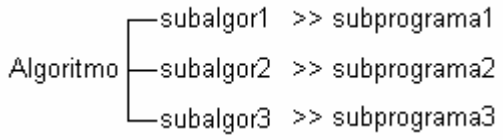
CAPITULO VI

SUBPROGRAMAS
PROCEDIMIENTOS Y FUNCIONES

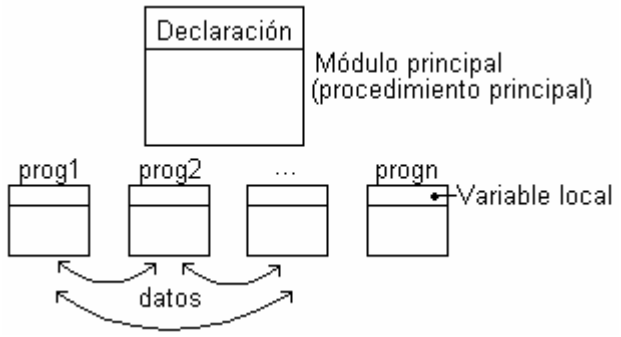
La resolución de problemas complejos se facilita considerablemente si se dividen en problemas mas pequeños; y la resolución de estos subproblemas se realiza mediante subalgoritmos.



Los subalgoritmos son unidades de programa o módulos que están diseñados para ejecutar laguna tarea específica. Éstos, constituidos por funciones o procedimientos, se escriben solamente una vez, pero pueden ser referenciados en diferentes puntos del programa, de modo que se puede evitar la duplicación innecesaria del código.



El módulo principal se ejecuta en una primera instancia, que da la orden de inicio de ejecución de los subprogramas. Puede ser ejecutado n veces.



Es importante saber que datos se van a compartir entre los programas.
El subprograma es un programa en sí mismo, ejecutado por la solicitud del programa principal o de otro subprograma, una n cantidad de veces.
Cuando realiza la solicitud, el programa se detiene hasta que el subprograma deja de realizar su tarea, luego continúa; esto se conoce como **control de ejecución**.

FUNCIONES

Una función es un subprograma que recibe, como argumentos o parámetros, datos de tipo numérico o no numérico, y devuelve un único resultado.
Las funciones incorporadas al sistema se denominan *funciones internas*, o intrínsecas; las funciones definidas por el usuario se llaman *funciones externas*.
El algoritmo o programa invoca la función con el nombre de esta última en una expresión seguida de una lista de argumentos que deben coincidir en cantidad, tipo y orden con los de la función que fue definida.

Declaración de funciones

En pseudocódigo:

SINTAXIS

Funcion nombrefun(lista de parámetros): Tipo
..
...(Declaraciones locales)
..

Hacer
..(cuerpo ejecutable de la función)
...
nombrefun:= <valor de la función> **// 1 //**
Fin Hacer

Fin Funcion

DESCRIPCIÓN

nombrefun : es el nombre indicatorio de la función.

lista de parámetros : es la lista de parámetros formales. Esta lista NO puede ser vacía.

Tipo : es el tipo de resultado que devuelve la función.

// 1 // En el cuerpo de la función debe existir una sentencia que asigne valor explícitamente al nombre de la función.
Para que las acciones descritas en un subprograma función sean ejecutadas, se necesita que éste sea invocado desde un programa principal o desde otros subprogramas a fin de proporcionarle los argumentos de entrada necesarios para realizar estas acciones.

Invocación a las funciones

Una función definida por el usuario se llama haciendo referencia a su nombre. En pseudocódigo:

..
... //líneas de programa//
...
nombrefun(lista de parámetros reales o actuales)
...
..

La sentencia **nombrefun** acompañada de los parámetros es la que inicia la ejecución de la función. El control de ejecución lo toma la función, ejecuta secuencialmente cada una de sus sentencias, y cuando termina de ejecutarse, le devuelve el control al programa llamador, ejecutándose la secuencia inmediatamente siguiente a la de la llamada. El resultado lo devuelve en el nombre de la función; el mismo se refiere a la zona de memoria que contiene el valor devuelto por la ejecución de la función.

Cada vez que se llama a una función desde el algoritmo principal, se establece automáticamente una correspondencia entre los parámetros formales y los reales. Debe haber exactamente el mismo número de parámetros reales que de formales en la declaración de la función, y se presupone una correspondencia uno a uno de izquierda a derecha entre los parámetros formales y reales.
Una llamada a una función implica los siguientes pasos:

1. A cada parámetro formal se le asigna el **valor real** de su correspondiente parámetro actual (cabe destacar que digo “real” refiriéndome al valor verdadero con el cual va a trabajar el subprograma, y no al tipo de dato).
2. Se ejecuta el cuerpo de acciones de la función.
3. Se devuelve el valor de la función al nombre de la función y se retorna al punto de llamada.

PROCEDIMIENTOS

Un procedimiento es un subprograma que ejecuta una tarea determinada. Está compuesto por un conjunto de sentencias, a las que s le asigna un nombre, o identificador. Constituyen unidades del programa, y su tarea se ejecuta siempre y cuando encuentre el nombre que se le asignó a dicho procedimiento.
Los procedimientos deben ser declarados obligatoriamente antes de que puedan ser llamados en el cuerpo del programa principal. Para ser activados o ejecutados, deben ser llamados desde el programa en que fueron declarados.
Pueden recibir cero o mas valores del programa principal que lo llama y lo activa, y devolver cero o mas valores a dicho programa llamador.
Todo procedimiento, al igual que un programa principal, consta de una cabecera, que proporciona su nombre y sus parámetros de comunicación; de una sección de declaraciones locales y el cuerpo de sentencias ejecutables. Las ventajas mas destacables de usar procedimientos son:

1. Facilitan el diseño top-down.
2. Se pueden ejecutar mas de una vez en un programa, con solo llamarlos las veces que ase desee. Con esto se ahorra tiempo de programación.
3. El mismo procedimiento se puede usar en distintos programas.
4. Su uso facilita la división de tareas entre los programadores de un equipo.
5. Se pueden probar individualmente e incorporarlos en *librerías* o *bibliotecas*.

Declaración de procedimientos

Al igual que cualquier otra variable, los procedimientos se deben declarar dentro del cuerpo del programa que los usa. La declaración del mismo NO indica que debe ejecutarse ese procedimiento, sino que le indica a la computadora cuales son las instrucciones del mismo y donde están localizadas. En pseudocódigo:

SINTAXIS

Procedimiento nombre(lista de parámetros formales)
Declaraciones locales
...

Hacer
...
cuerpo del procedimiento
...
Fin Hacer

Fin Procedimiento

DESCRIPCIÓN

nombre : se refiere al nombre del procedimiento y debe ser un identificador válido.

lista de parámetros formales : es el conjunto de valores que sirven para pasar y/o devolver información a/desde el procedimiento, desde/a el programa llamador. En la lista se especifica cada parámetro indicando nombre y tipo de cada uno de ellos. Esta lista puede ser vacía, es decir, existen procedimientos sin parámetros.

La posición adecuada de la declaración de los procedimientos está entre la declaración de las variables y el cuerpo del programa principal:

Programa nombreprog

Constantes
...(declaración de las constantes)

Variables
...(declaración de las variables)

Procedimientos y funciones
...(declaración de los procedimientos y/o las funciones)
..
..
Hacer
..
...(cuerpo del programa principal)
..
Fin Hacer
Fin Programa

Llamada a los procedimientos

Se los llama por su nombre, dentro del programa que los declara, respetando este formato:

.... //líneas del programa//
....
nombredelprocedimiento(lista de parámetros reales)
....

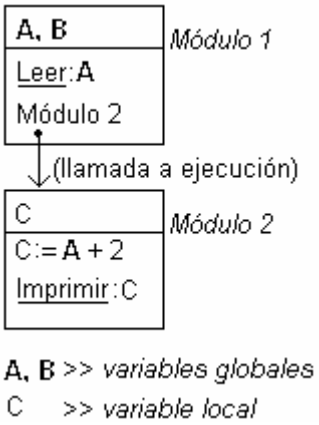
....

La sentencia *nombredelprocedimiento* acompañada de la liste de parámetros (si es que ésta existe), es la que inicia la ejecución del procedimiento. El control lo toma el procedimiento, ejecuta secuencialmente cada una de sus sentencias, y cuando termina de ejecutársele devuelve el control al programa llamador, ejecutándose la sentencia inmediatamente siguiente a la de la llamada.

PROCEDIMIENTO	FUNCIÓN
• Puede devolver uno, muchos o ningún resultado. • Su nombre no está asociado a ninguno de los resultados que obtiene. • Existen procedimientos sin parámetros.	• Devuelve un único resultado. • El resultado lo devuelve en el nombre de la función, que es el identificador. • No existen funciones sin parámetros.

ÁMBITO: VARIABLES LOCALES Y GLOBALES

Las variables utilizadas en los programas principales y subprogramas se clasifican en dos tipos: locales y globales.

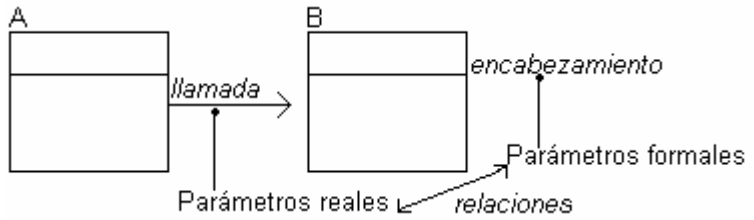


Una variable **local** es una variable que está declarada dentro de un subprograma, y se dice que es local al subprograma. Una variable local sólo está disponible durante el funcionamiento del subprograma que la declara, y su valor se pierde una vez que finaliza la ejecución del subprograma.
Las variables declaradas en el programa principal se llaman **globales**, pues pueden ser utilizadas en el programa principal y en todos los subprogramas en él declarados. Si existen dos variables con el mismo nombre, pero una es global y la otra es local, dentro del subprograma tiene prioridad la variable local de igual nombre. La variable global deja de existir cuando finaliza la ejecución del programa.

//ERROR DE EFECTO LATERAL: Se produce al realizarse un cambio de nombre, de naturaleza, o eliminación de una variable global, que afecta al subprograma. Esto se logra evitar utilizando parámetros.//

COMUNICACIÓN CON SUBPROGRAMAS: PASO DE PARÁMETROS

Cuando un programa llama a un subprograma, la información entre ellos se comunica a través de la lista de parámetros, y se establece una correspondencia automática entre los parámetros formales y los reales.
Un parámetro es un método de pasaje de valores de variables desde el programa principal al procedimiento y viceversa; es una variable cuyo valor debe ser proporcionado por el programa llamador al procedimiento, o bien ser devuelto desde el procedimiento, con un valor, hasta el programa que lo llama.
Cada subprograma tiene un encabezamiento, en el cual se indican los parámetros formales En el momento en que un subprograma realiza la llamada a otro módulo, se indican los parámetros reales Los parámetros reales son utilizados por el subprograma en lugar de los parámetros formales.



Paso de parámetros

Existen diferentes métodos para el paso de parámetros a subprogramas. Es preciso conocer el método adoptado por cada lenguaje, ya que la elección puede afectar a la semántica del código. Los parámetros pueden ser clasificados como:

- Entradas (E): las entradas proporcionan valores desde el programa que llama, y se utilizan dentro del procedimiento.
- Salidas (S): las salidas proporcionan los resultados del subprograma.
- Entradas/Salidas (E/S): un solo parámetro se utiliza para mandar argumentos a un programa y para devolver resultados.

Los métodos mas empleados para realizar el paso de parámetros son:

- Paso por valor (parámetro valor)
- Paso por referencia o dirección (parámetro variable)
- Paso por nombre
- Paso por resultado

Los parámetros formales (locales al subprograma) reciben como valores iniciales los valores de los parámetros reales, y con ellos se ejecutan las acciones descritas en el subprograma.

Paso por valor

Son los parámetros unidireccionales, que se usan para dar información al subprograma, pero no pueden devolver valores desde el mismo. Aunque el procedimiento les cambie su valor, este nuevo valor no se refleja en el programa llamador. Esto se debe a que en la llamada al subprograma, se le asigna el valor del parámetro real a la variable que representa al parámetro formal correspondiente, dicho de otra forma, se crea una copia del parámetro real. Todo parámetro que NO esté calificado con la palabra reservada **ref.** es tomado como parámetro valor.

Paso por referencia

Son los parámetros que están precedidos por la palabra reservada **ref.**, que indica que sólo reciben valor en el subprograma, o bien proporcionan valor al subprograma y reciben un valor nuevo en el mismo. Así, todo cambio realizado sobre los parámetros formales

precedidos por "**ref.**", se refleja en los parámetros reales correspondientes. Se los considera como parámetros bidireccionales o variables, ya que son de Entrada y/o Salida.

EFECTOS LATERALES

Las modificaciones que se produzcan mediante una función o procedimiento en los elementos situados fuera del subprograma se denominan efectos laterales.

En procedimientos

Si un procedimiento modifica una variable global (distinta de un parámetro real), éste es un efecto lateral. Por ello, excepto en contadas ocasiones, no debe aparecer en la declaración del procedimiento. Si se necesita una variable temporal en un procedimiento, se debe utilizar una variable local, no una global. Si se desea que el procedimiento modifique el valor de una variable global, utilícela como el parámetro real en una llamada al procedimiento.

En Funciones

Una función puede tener parámetros variables además de parámetros valor en la lista de parámetros formales. Una función puede cambiar el contenido de una variable global y ejecutar instrucciones de entrada/salida. Estas operaciones se conocen como parámetros laterales, y se deben evitar.

RECURSIÓN (recursividad)

Un subprograma que se puede llamar a sí mismo se llama recursivo. La recursión puede ser utilizada como una alternativa a la repetición o estructura repetitiva. La escritura de un procedimiento o función recursiva es similar a sus homónimos no recursivos; sin embargo, para evitar que la recursión continúe indefinidamente, es preciso incluir una condición de terminación.

CAPITULO VII

ESTRUCURAS DE DATOS.
ARREGLOS

Un arreglo es una secuencia de posiciones de memoria a las que se puede acceder directamente que almacenan valores del mismo tipo, los cuales se identifican por su posición, que en pseudocódigo comienza en 1.

INTRODUCCIÓN A LAS ESTRUCTURAS DE DATOS

Hasta aquí, hemos trabajado con el concepto de dato, y como almacenarlo en una variable. Esta *estructura de datos simple* podía ser un número entero, un real, un carácter o un valor lógico. Ahora vamos a ampliar ese concepto de dato a un *conjunto de datos*.

En la resolución de problemas, el primer paso es la detección de los datos que conforman el universo de resolver el problema. Por lo tanto, la selección de datos y su posterior organización es fundamental para definir y resolver el problema. Todas las formas en que se relacionan lógicamente los datos primitivos extraídos del enunciado del problema y que se deducen de él, definen distintas estructuras de datos.
Las distintas estructuras se diferencian por la forma en que sus componentes se relacionan, y por el tipo de las mismas. Todos los datos estructurados deben ser construidos a partir de datos primitivos. Por ejemplo, una estructura de datos conocida es el número complejo: toma la forma de par ordenado de números reales (siendo el número real un tipo de datos primitivos). Pasemos entonces a una definición formal de estructura de datos compuesta:

"*Una estructura de datos compuesta es un conjunto de datos reunidos bajo un nombre único y colectivo.*"

Ahora bien, estas estructuras de datos compuestas se dividen en homogéneas y heterogéneas. En las homogéneas sólo se pueden almacenar datos de un mismo tipo (los elementos del conjunto son: o todos reales, o todos enteros, o todos caracter, etc.). En las heterogéneas, el conjunto de datos puede ser una combinación de tipos.

Los tipos de datos más frecuentes utilizados en los diferentes lenguajes de programación son:

DATOS SIMPLES:
- Estándar:
  - Entero (integer)
  - Real (real)
  - Caracter (char)
  - Lógico (boolean)
- Definido por el usuario:
  - Subrango (subrange)
  - Enumerativo (enumerated)

DATOS ESTRUCTURADOS:

Estáticos:

Arreglo Vector
Arreglo Matriz
Registro
Archivo (fichero)
Conjunto
Cadena

Dinámicos:

Lista (pila / cola)
Lista enlazada
Árbol
Grafo

(Estructuras homogéneas)
(Estructuras heterogéneas)

- Los tipos de datos simples o primitivos no están compuestos de otras estructuras de datos.
- Las estructuras de datos estáticas son aquellas en las que el tamaño ocupado en memoria se define antes de que el programa se ejecute y no puede modificarse dicho tamaño durante la ejecución del programa.
- Las estructuras de datos dinámicas no tienen las restricciones o limitaciones en el tamaño de memoria ocupada que son propias de las estructuras estáticas.
- Los tipos de datos simples tienen como característica común que cada variable representa a un elemento.
- Los tipos de datos estructurados tienen como característica común que un identificador (nombre) puede representar múltiples datos individuales, pudiendo cada uno de éstos ser referenciado independientemente.

ARREGLOS

Un arreglo (matriz o vector) es un conjunto finito y ordenado de elementos homogéneos. La propiedad "ordenado" significa que el elemento primero, segundo, tercero,..., enésimo de un arreglo puede ser identificado. Los elementos de un arreglo son homogéneos, es decir, del mismo tipo de dato.
Cada componente de un arreglo se denota, explícitamente, y es accedida, directamente, mencionando el nombre del arreglo seguido de una expresión encerrada entre paréntesis, a la que llamamos *índice* del arreglo.
La cantidad de componentes del arreglo se india explícitamente cuando se declara el mismo, y queda desde entonces INVARIABLE. A ese número lo llamamos *dimensión* del arreglo.
El índice del arreglo toma valores entre uno y la dimensión del mismo. Si se direcciona una posición que no se encuentra entre dichos límites, se producirá un error de ejecución, pues se estará intentando acceder a una zona de memoria indefinida (pues el vector está definido entre las posiciones 1 y dimensión).
A un arreglo lineal se lo denomina *unidimensional*, y se utiliza un solo índice para denotar sus elementos. A un arreglo tipo matricial se lo denomina arreglo *bidimensional* o de dos dimensiones, y usa dos índices para determinar sus elementos.

Operaciones sobre arreglos

Las operaciones que se pueden realizar con arreglos durante el proceso de resolución de un problema son:

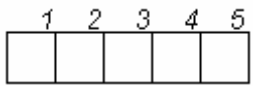
- Asignación;
- Lectura / Escritura;
- Recorrido (acceso secuencial);

- Actualización (añadir, borrar, insertar);
- Ordenamiento;
- Búsqueda.

En general, las operaciones con arreglos implican el tratamiento o procesamiento de los elementos individuales del arreglo.

ARREGLOS UNIDIMENSIONALES: VECTORES

Los elementos de un arreglo lineal se almacenan en posiciones sucesivas de memoria, es decir, un elemento al lado del otro. Si cada elemento de un arreglo de dimensión k, ocupa x posiciones, entonces el arreglo ocupa en total x * k posiciones. El almacenamiento es igual al de las variables, con la particularidad que ahora el conjunto de variables ocupa posiciones sucesivas o contiguas de la memoria.



siendo una variable: 

Declaración de vectores

Los vectores se definen en la zona de declaración del programa. Dicha definición se encabeza con la palabra reservada “Tipos estructurados”. Para declararlos se explicita el nombre del mismo, acompañado de su dimensión, y a continuación se especifica el tipo de sus elementos. Todos los elementos de un vector serán del mismo tipo. En pseudocódigo:

Programa.....

Tipos estructurados

<nom_vector>= Arreglo [Dimensión] : <tipo> //declaro un vector...//

Variables

<nom_variable>: <nom_vector> **//y luego una variable de tipo// //vector, donde alojarlo//**

Hacer

...

Fin Hacer

Fin Programa.....

EJ6) Programa con arreglos de tipo vector.

Programa: vector

Tipos Estructurados

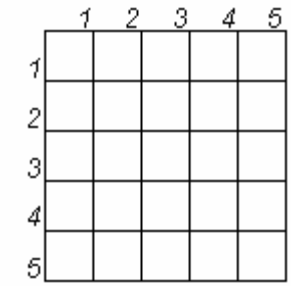
A= Arreglo [1..10] : entero 3

B= Arreglo [1..70] : real 7,2

```
Variables
  V1: A                                  //declaro una variable de tipo vector A//
  (V7, V2): B                            //declaro otras de tipo vector A//
  I: entero 2                            //índice//

  Hacer
    V1 [1]:= 0                           //se inicializan las variables//
    V1 [2]:= 7                           //de a una...//
    V1 [3]:= 10
    ...
    ...                                  //lo cual no sería lo ideal...//
    V1 [10]:= 14

    Repetir Para: I:= 1,10,1             //otra forma de hacerlo...//
      Imprimir: "ingrese un valor entero"
      Leer: V1 [I]                       //el valor ingresado por el usuario será asignado al vector//
    Fin Repetir Para                     //en la posición que indique el índice //

    //vista del arreglo vector//

    Repetir Para: I:= 1,10,1
      Imprimir: "V1 [",  I  ,"]=",  V1 [I]
    Fin Repetir Para

    //carga y promedio//

    Repetir Para: I:= 1,70,1
      Imprimir: "ingrese nota escuela 1 posición", I
      Leer: V2 [I]
      Imprimir: "ingrese nota escuela 2 posición", I
      Leer: V7 [I]

      V7 [I]:= (V2 [I] + V7 [I] ) / 2    //calculo el promedio x posición//
    Fin Repetir Para

    //visualizo V7 de la posición mayor a la menor...//

    I:= 70

    Repetir Mientras (I >= 1)
      Imprimir: "promedio posición",  I  ,"es:",  V7 [I]
      I:= I - 1
    Fin Repetir Mientras

  Fin Hacer

Fin Programa
```

ARREGLOS BIDIMENSIONALES: MATRICES

Las matrices pertenecen al conjunto de estructuras de datos compuestas homogéneas. Podríamos hacer una primera aproximación a estas estructuras definiéndola así:
Una matriz es un vector de vectores; un arreglo lineal donde cada elemento as a su vez un vector. Por supuesto que dichos vectores son todos de igual tipo, precisión y dimensión.
El problema de este esquema es que se hace complicado referenciar un elemento: primero hay que determinar cual vector se quiere trabajar, y luego cual es el elemento dentro del mismo.
Para simplificar esto, se visualiza gráficamente a la matriz como una cuadrícula:



líneas horizontales = filas
líneas verticales = columnas

Con este esquema se simplifica el concepto de matriz: es un conjunto de vectores. Cada fila representa un vector, asimismo, cada columna representa un vector. De esta forma es mas sencillo el acceso: se determina primero cual de todas las columnas o filas interesa, y luego se usa el elemento deseado como se hace con cualquier vector.
Podemos ver entonces, que todo dato que se guarde en esta cuadrícula debe ser referenciado a través de un índice de fila y uno de columna: cada elemento tiene dos entradas, una horizontal o de fila, y una vertical o de columna; en la intersección de fila y columna se hallará el dato.

En forma similar a un arreglo lineal, la matriz tiene un nombre único y genérico, que respeta las normas de declaración de variables, y para designar a sus elementos es necesario usar 2 índices. Así, el nombre de la matriz, seguido de los 2 índices separados con coma (el primero para la fila; el segundo para la columna), y encerrados entre corchetes, denota *un elemento* de la matriz en particular. EJ: **Mat[2,3]** es el elemento de la matriz Mat que se halla en la intersección se la 2da. Fila y la 3er. Columna.

Por ser un arreglo homogéneo, sus elementos son todos del mismo tipo y precisión. La dimensión de una matriz (cantidad total de elementos) se calcula haciendo el producto entre la cantidad total de filas por la cantidad total de columnas. En forma genérica, cada matriz de M filas y N columnas tendrá (M*N) elementos.

Almacenamiento de matrices en memoria

El espacio que ocupa en memoria una matriz se calcula teniendo en cuenta la cantidad total de elementos, multiplicada por el espacio que ocupa un elemento del tipo y precisión especificado para dicha matriz. EJ: dada una matriz de M filas por M columnas, de enteros de 2 dígitos, el espacio que ocupa en memoria es (M*N*espacio que ocupa un entero de 2 dígitos).

Los elementos de una matriz se almacenan en posiciones sucesivas de memoria, al igual que los vectores. En otras palabras podemos decir que se almacenan como vectores contiguos: una fila de la matriz al lado de la otra, y cada elemento de la fila al lado del siguiente.

Declaración de matrices

Las matrices se definen en la zona de declaraciones del programa, dentro del bloque de Tipos estructurados. Para ello se especifica el nombre de la misma seguido de la cantidad total de filas y de columnas, separadas por coma y encerradas entre corchetes, y luego se escribe el tipo y precisión de los elementos. En pseudocódigo:

Programa.....

Tipos estructurados

<nom_matriz>= Arreglo [filas,columnas] : <tipo> //declaro una matriz...//

Variables

<nom_variable>: <nom_matriz> **//y luego una variable de tipo// //matriz, donde alojarla//**

Hacer

...

Fin Hacer

Fin Programa.....

La declaración reserva lugar en la memoria, lo asocia con el nombre de la matriz, pero su contenido queda indeterminado, como ocurre con las variables. Es necesario hacer una operación de asignación de valores, para que la matriz tenga elementos.

EJ7) Declaración y asignacion de valores a una matriz.

Programa: 2D

Tipos Estructurados

//Declaro de la forma **<nom_matriz>= Arreglo [Dimensión] : <tipo>** //

MAT= Arreglo [1..5, 1..5]: entero 7

Variables

M: MAT //**declaro una variable de tipo matriz *MAT*//**

(I, J): entero 1 //índices//

Hacer

Repetir Para: I:= 1, 5, 1 // **filas** //

Repetir Para: J := 1, 5, 1 //**columnas** //

M [I, J]:= 0 //inicializo en 0 toda la matriz//

//sino también podría ser: ***Leer: M [I, J]*** //

Fin Repetir Para

Fin Repetir Para

Fin Hacer

Fin Programa

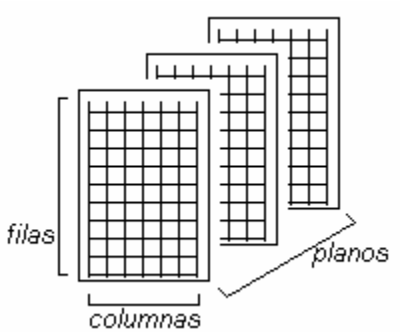
Arreglo de 3 dimensiones: TENSOR

La declaración sería la siguiente:

Dentro de Tipos Estructurados...

EST3= ARREGLO [1..X, 1..Y, 1..Z]: <tipo de dato>

...donde X determina la cantidad de filas, Y la cantidad de columnas, Z la cantidad de planos:



CAPITULO VIII

CADENAS DE CARACTERES

Una cadena de caracteres es una secuencia de cero o mas símbolos, que incluyen letras del alfabeto, dígitos, y caracteres especiales.
Los lenguajes de programación utilizan *juegos de caracteres* para comunicarse con la computadora. Hoy día, la mayoría de las computadoras trabaja con diferentes tipos de juegos de caracteres, de los que se destacan el código ASCII (American Standard Code for Information Interchange) y el EBCDIC (Extended Binary Code Decimal Interchange Code).
En general, el caracter ocupa un byte de almacenamiento de memoria.

CÓDIGO ASCII

El código ASCII básico utiliza 7 bits (dígitos binarios, 0, 1)) para cada carácter a representar, lo que supone un total de 2^7 (128) caracteres distintos. El ASCII ampliado utiliza 8 bits y, en este caso, consta de 256 caracteres:

- Alfabéticos
- Numéricos
- Especiales
- De control: son caracteres no imprimibles y que realizan una serie de funciones relacionadas con la escritura, transmisión de datos, separador de archivos, etc.

CÓDIGO EBCDIC

Es muy similar al ASCII, incluyendo también, además de los caracteres alfanuméricos y especiales, caracteres de control.

Cadena de caracteres

La longitud de una cadena es la cantidad de caracteres que contiene. La cadena que no contiene ningún caracter se denomina cadena nula, y su longitud es cero; no se debe confundir con una cadena compuesta sólo de espacios en blanco, ya que esta última tendrá como longitud el número de blancos de la misma.
La representación de las cadenas suele ser con comillas simples o dobles.
Una subcadena es una cadena de caracteres que ha sido extraída de otra cadena de mayor longitud.

Datos tipo carácter

CONSTANTES

Una constante tipo caracter es un carácter encerrado entre comillas;
Una constante tipo cadena es un conjunto de caracteres válidos entre comillas, para evitar confundirla con nombres de variables, operadores, enteros, etc.

VARIABLES

Una variable de tipo caracter o cadena es una variable cuyo valor es una cadena de caracteres. Éstas se deben declarar en el algoritmo, y atendiendo a la declaración de la longitud, las variables se dividen en estáticas, semiestáticas y dinámicas.

Variables estáticas son aquellas en las que su longitud se define antes de ejecutar el programa, la cual no puede modificarse a lo largo de éste.

Variables semiestáticas son aquellas cuya longitud puede variar durante la ejecución del programa, pero sin sobrepasar cierto límite.

Variables dinámicas son aquellas cuya longitud puede variar sin limitaciones dentro del programa.

Cadenas de longitud fija: Se consideran como vectores de longitud declarada, con blancos a izquierda o derecha, si la cadena no tiene longitud declarada.

Cadenas de longitud variable con un máximo: Se considera como un puntero con dos campos que contienen la longitud máxima y la longitud actual.

Cadenas de longitud indefinida: Se representan mediante listas enlazadas, que son listas que se unen mediante un puntero. Estas listas contienen elementos como caracteres empaquetados y enlazados cada uno con el siguiente por un puntero.

Operaciones con cadenas

Las operaciones mas usuales con cadenas son:

- Cálculo de la longitud;
- Comparaciones;
- Concatenación;
- Extracción de subcadenas;
- Búsqueda de información.

CAPITULO IX

ORDENAMIENTO Y BÚSQUEDA EN ARREGLOS

PROCESAMIENTO DE DATOS

Acciones básicas de manejo de datos
(sobre arreglos que ya están cargados en memoria).

Ordenamiento

Logra optimizar el almacenamiento / acceso a los datos:
-Simple (arreglo; vector)
-complejo (lista)

Búsqueda

Consiste en poder acceder a un dato dentro de un conjunto de datos, dependiendo si el último está ordenado o no:
-Sobre un conjunto ordenado >> búsqueda binaria
-Sobre uno desordenado >> búsqueda secuencial

Intercalación

Se refiere a mezclar datos de diferentes conjuntos, es decir unir varios conjuntos para crear uno nuevo:
-Previamente ordenados >> intercalo
-Sin ordenar >> intercalo y luego ordeno

ORDENAMIENTO

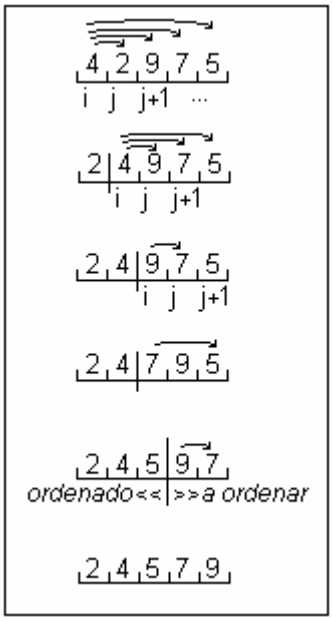
El ordenamiento provee un método para organizar la información, facilitando de esta manera la recuperación de datos específicos. Imaginemos lo difícil que sería usar un diccionario que no estuviese ordenado alfabéticamente.
La elección de un algoritmo debe tener en cuenta el uso eficiente de la memoria que se tiene disponible, así como también el tiempo de ejecución del mismo.

Clasificación

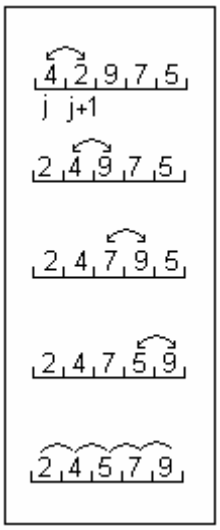
Simple (arreglo; vector)
Complejo (lista)

Principios Básicos
(de ordenamiento de datos entre sí)

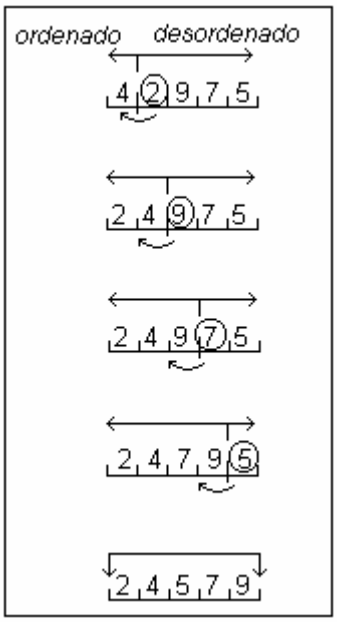
Selección
Intercambio
Inserción



Selección
Compara contenidos; mantiene fija una posición "i", y la compara con otra "j", luego con "j + 1" y así sucesivamente hasta recorrer completamente el vector.



Intercambio
Compara una posición "j" contra una "j + 1".



Inserción
Construye un grupo ordenado final cuando los datos que van llegando tienen un manejo especial; del conjunto original genera un subconjunto "ordenado" en el que va insertando datos y los coloca entre los valores mayor y menor correspondientes, desplazando otros datos.

Diferencias entre el simple y el complejo

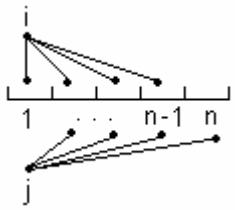
Simple
-Cada método tiene incorporado 1 principio básico;
-Para una cantidad pequeña de elementos a ordenar, los simples no son tan malos en rendimiento;
-Es fácil de construir.

Complejo
-Diferentes criterios de manejo de datos;
-Para una cantidad grande de elementos corresponde un método complejo.

Factores básicos que inciden en el rendimiento

-Cantidad de elementos (n)
-Grado de desorden (mayor desorden = mas ciclos de comparación

MÉTODO DE SELECCIÓN




```
Repetir Para: I := 1, n - 1                 //controla al cantidad de pasadas//

  Repetir Para: J := I + 1, n               //manejo el valor de posición / índice//

    Si: (V[I] < V [J]) Entonces             //comparación de elementos en forma ascendente//
                        AUX:= V[I]
                        V[I]:= V[J]
                        V[J]:= AUX
    Fin Si

  Fin Repetir Para

Fin Repetir Para
```

- Cantidad de operaciones en el orden de n al cuadrado

Análisis de las operaciones de comparación de datos
//En este caso, no depende del desorden sino de la cantidad de elementos a ordenar//

Primera pasada: n - 1 //cantidad de comparaciones//
Segunda pasada: n - 2
...
n - 1 pasada: 1

$$\text{Total: } (n-1)+(n-2)+(n-3)\ldots+1 = \frac{n^2-n}{2}$$

Análisis con respecto a los movimientos
//Depende de n pero también del grado de desorden//

Peor caso: 3 x (n^2) / 2 = (n^2 / 2) - (n / 2) para n >> ∞ (n^2 / 2)

Mejor caso: 0

>> promedio: 3/4 n^2 - 3/4 n para n >> ∞ 3/4 n^2

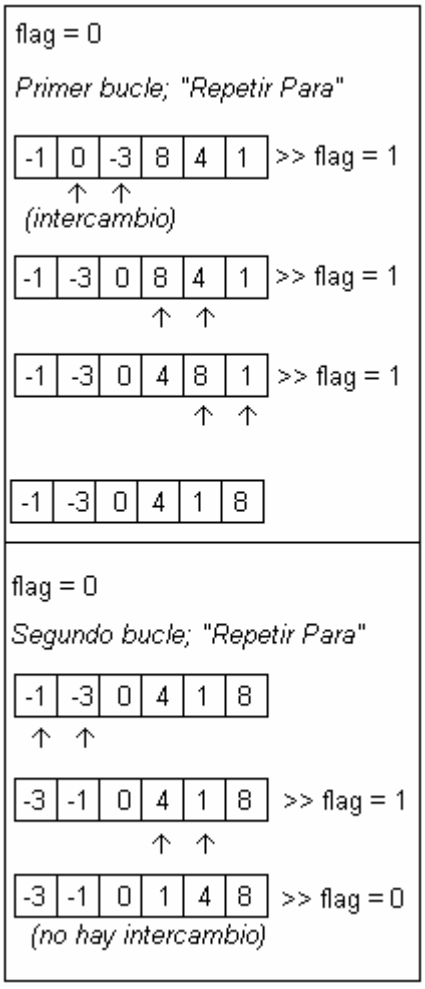
MÉTODO DE INTERCAMBIO (BURBUJA)

flag / testigo / centinela = 1 >> hubo intercambio
... = 0 >> no hubo intercambio

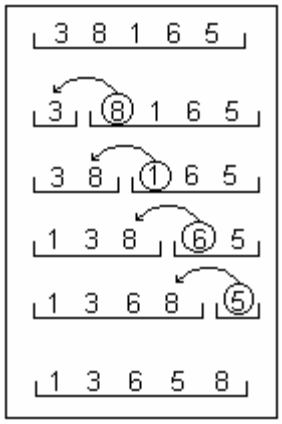
//La inclusión de un flag al método me permite darme cuenta cuando deja de haber intercambio de valores, es decir, cuando el vector ya está ordenado. Así se reduce la cantidad de pasadas a sólo las necesarias.//

```
flag:= 1              //hubo intercambio, obligo al programa a ejecutarse al menos una vez//

Repetir Mientras (flag= 1)

  flag:= 0            //no hubo intercambio//

  Repetir Para: I:= 1, n - 1, 1

    Si: (VEC [I] > VEC [I + 1]) Entonces        //intercambio//

                                    AUX:= VEC [I]
                                    VEC [I]:= VEC [I + 1]
                                    VEC [I + 1]:= AUX

                                    flag:= 1
    Fin Si

  Fin Repetir Para

Fin Repetir Mientras
```

Prueba de escritorio

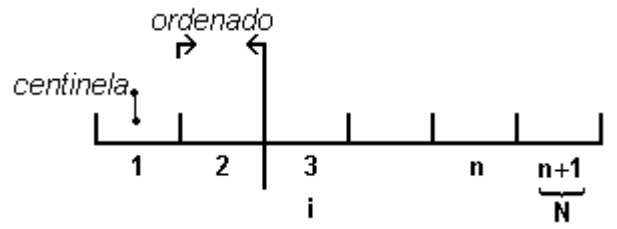


MÉTODO DE INSERCIÓN



//Tarea adicional: el índice debe estar en la posición correcta//

Para dejar de tener en cuenta si el índice se sale de rango se busca colocar en una posición mas a la izquierda el mismo contenido a insertar, así nunca llego a cero.



Resulta conveniente utilizar esta modificación al método cuando **se deben ordenar elementos a medida que se van leyendo:**

Leer: VEC [2]	//ingresa un valor a la posición 2 del vector, para conformar el grupo ordenado.//
Repetir Para: **I:= 3, N**	//todos los elementos a ordenar//
Leer: VEC [1]	//asigno el centinela, elemento a ordenar.//
J:= I - 1	//el índice J marca el último elemento del grupo ordenado//
	//Cabe destacar que **I siempre va a marcar el primer elemento del grupo desordenado**, en este caso vacío, porque los elementos ingresan de a uno**//**
Repetir Mientras (VEC[1] < VEC[J])	//es decir, repetir mientras el valor a insertar y ordenar sea menor al valor que indique el índice J//
VEC [J + 1]:= VEC [J]	//se copia el valor que se encuentra en J, y se lo deposita en J + 1, es decir en I //
J:= J - 1	//mueve el índice a la posición anterior//
	//Se desea desplazar la cant. de elementos necesaria del grupo ordenado un lugar hacia la derecha, haciendo un lugar para insertar el valor a ordenar entre los valores correspondientes mayor y menor...//
Fin Repetir mientras	
VEC [J + 1]:= VEC [1]	//...Ahora ingresa el valor, que ha sido guardado anteriormente en el "centinela", a la izquierda del valor mayor que le sigue//
Fin Repetir Para	

Si el vector a ordenar **ya está cargado**, el método se construye de la siguiente manera:

```
Repetir Para: I:= 3, N

  VEC [1]:= VEC [I]        // En este caso, el centinela va a ser el elemento que se tome del
                           grupo desordenado, que se desea incluir entre los valores corresp.
                           del grupo ordenado.//

  J:= I - 1

  Repetir Mientras ( VEC[1] < VEC[J] )

    VEC [J + 1]:= VEC [J]
    J:= J - 1

  Fin Repetir Mientras

  VEC [J + 1]:= VEC [1]

Fin Repetir Para
```

BÚSQUEDA

El problema de la búsqueda se refiere a localizar un dato dentro de un conjunto; el dato a localizar puede o no estar dentro de los elementos del conjunto. La búsqueda de un elemento determinado es un área muy frecuente; por ejemplo buscar por el nombre y apellido de un alumno de Algoritmos el resto de sus datos.
Debido a la importancia del tema, se han desarrollado una serie de algoritmos, donde la eficiencia de los mismos se mide por la velocidad con que se encuentra el dato buscado.
Formalmente, al problema de búsqueda lo podemos enunciar así:

Dado un conjunto de n elementos distintos, y un dato k llamado argumento, determinar:
-si k pertenece al conjunto, y en ese caso, indicar cual es su posición en el mismo;
-si k no pertenece al conjunto.

BÚSQUEDA SECUENCIAL

La manera mas sencilla y natural para encontrar un argumento k, dentro de un conjunto es comparar k con cada elemento hasta encontrarlo o bien, hasta agotar el conjunto. En pseudocódigo:

```
Programa Busco

  Tipos estructurados
    V= ARREGLO[1..30]: entero 2

  Variables
    K : entero 2
    I : entero 2
    Esta : bolean
    VEC : V
```

```
Hacer                          //carga del vector//
  Repetir Para: I:= 1,30
        Leer:VEC[I]

  Fin Repetir Para

  Leer: K                      //ingresa el valor a buscar//

  Esta:= false                 //obligo a la estructura siguiente a funcionar al menos 1 vez//

  Repetir Mientras (I<=30 and Esta=false)   //mientras no termine de recorrer el vector, y
                                            mientras no lo encuentre//
    Si : (K = V[I]) Entonces
        Imprimir: "está en la posición", I
        Esta:= true                 //lo encontré, y sale del mientras//
        Sino
        I:= 1 + I                   //no lo encontré, me fijo en la posición siguiente//
    Fin Si

  Fin Repetir Mientras

  Si: (Esta= false) Entonces
        Imprimir:"no está"          //recorrí todo el vector y no estaba//
  Fin Si
 Fin Hacer
Fin Programa
```

Análisis de la cantidad de comparaciones

Este algoritmo requiere una sola comparación si el argumento está en el primer lugar del vector, y N comparaciones en el peor de los casos, es decir si es el último elemento del vector, o bien si no pertenece al conjunto. Por consiguiente, la cantidad total de comparaciones, en promedio es : (N + 1)/ 2

BÚSQUEDA BINARIA O DICOTÓMICA

Si los elementos del conjunto se encuentran ordenados, aplicaremos un método de búsqueda mas eficiente.
Supongamos que los elementos han sido ordenados por cualquier método en forma ascendente. El método dicotómico consiste en localizar, aproximadamente, la posición media del arreglo y examinar el valor allí encontrado. Si este valor es mayor que el argumento buscado, entonces se sigue buscando el dato en la primera mitad del arreglo; repitiéndose el proceso en la mitad correspondiente del mismo, hasta hallar el valor deseado. Si por el contrario, el elemento de la mitad es menor que el buscado, entonces se continúa la búsqueda en la mitad derecha del arreglo. Este proceso de división por mitades se repite hasta encontrar el valor buscado en alguna mitad, o bien hasta que el intervalo de búsqueda haya quedado vacío.

Dado un vector V y un dato K, después de comparar el argumento K con el **elemento mitad V[I],** existen 3 posibilidades:

1. Que K < V[I], entonces se eliminan V[I + 1], V[I + 2], ..., V[N] de la búsqueda.
2. Que K = V[I], entonces termina la búsqueda.
3. Que K > V[I], entonces se elimina V[1], V[2], ..., V[I] de la búsqueda.

EJ) Sea el siguiente conjunto:

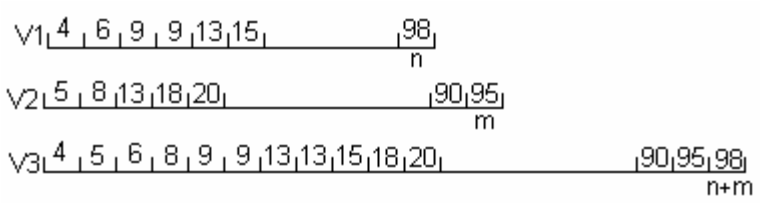
2 8 14 17 21 26 35 42

y el valor a buscar es k = 35

El elemento central del arreglo es 17. Como 35 > 17, se desechan los valores 2, 8, 14, y 17. El proceso sigue, usando la mitad derecha del arreglo: desde V[5] a V[8]. El elemento mitad de este nuevo sub-vector es 26. Como 35 > 26, descartamos los elementos 21 y 26. sigue la búsqueda con V[7] y V[8]. La mitad de este intervalo es 35. Como es el valor buscado, el proceso termina con éxito. En pseudocódigo:

```
Programa Busca

  Tipos estructurados
    Vector = ARREGLO[1..30]: entero 3

  Variables
    K: entero 3
    (pos, Vi, Vf, Mitad, I): entero 2
    Vec: Vector

  Hacer
    Repetir Para: I:= 1,30
          Leer:Vec[I]
    Fin Repetir Para

    Leer: K
    Vi:= 1
    Vf:= 30
    Pos:= 0

    Repetir Mientras ((Vi < Vf) and (pos = 0))
          Mitad:= Ent ((Vi + Vf) / 2)

          Si: (K= Vec(Mitad)) Entonces

                                Pos:=Mitad
                          Sino
                                Si: (K < Vec[Mitad]) Entonces

                                                      Vf:= Mitad – 1
                                                Sino
                                                      Vi:= Mitad + 1
                                Fin Si
          Fin Si
    Fin Repetir Mientras

    Si: (pos <> 0) Entonces
                      Imprimir: “el valor está en”, pos
                Sino
                      Imprimir: “el valor no está.”
    Fin Si
  Fin Hacer
Fin Programa
```

INTERCALACIÓN (MEZCLA)



El vector V3 fue creado a partir de V1 y V2, mezclando los valores contenidos en los dos últimos. Para realizar este proceso, se deben tener en cuenta los siguientes factores:
-Los vectores a mezclar no suelen tener la misma longitud;
-La longitud del tercer vector siempre resulta de la suma de las long. de los dos primeros.
-Cuando se realiza el proceso, siempre hay un vector que termina de recorrerse antes que otro, sin importar sus longitudes;
-Las comparaciones que se realizan entre ambos no son parejas; es decir, no se compara el primer valor del vector 1 con el primero del vector 2, y luego el segundo valor del vector 1 con el segundo valor del vector 2, etc.;
-Si los vectores a mezclar ya están ordenados, solo hay que mezclarlos;
-Si están desordenados, primero hay que intercalarlos y luego ordenarlos.

```
I:= 1                //índice del V1//
J:= 1                //índice del V2//
K:= 0                //índice del V3//

Repetir Mientras ((I <= n) and (J <= m))     //Deja de rep. si algún vector se recorre
                                              completamente//

  K:= K + 1                                  //El V3 siempre va adquirir un valor, ya sea de V1 ó de
                                               V2//
  Si: (V1 [I] <= V2 [J]) Entonces            //V1 menor ó igual que V2//
                    V3 [K]:= V1 [I]
                    I := I + 1               //entonces avanzo un lugar para comparar con el valor
                                               que corresponde de V2//

                Sinó                         //V2 menor que V1//
                    V3 [K]:= V2 [J]
                    J := J + 1
  Fin Si
Fin Repetir Mientras

Si: (I <= n) Entonces                        //El vector que resta por recorrer es V1//

        Repetir Para: X := I, n
        K := K + 1
        V3 [K] := V1 [X]                     //entonces se copian los valores de V1 a V3//
        Fin Repetir Para

    Sinó
        Repetir Para: X := J, m              //El que resta por recorrer es V2//
        K := K + 1
        V3 [K] := V2 [X]                     //Se copia a V3//
        Fin Repetir Para

Fin Si
```

CAPITULO X

ARCHIVOS

INTRODUCCIÓN

En un arreglo podemos almacenar información del mismo tipo, es decir homogénea. Pero la realidad nos conduce a veces a situaciones en donde la información que manejamos no es toda del mismo tipo. Ello nos demuestra que en ciertas ocasiones es necesario trabajar datos que se hallan relacionados lógicamente, pero que no son homogéneos.
Consideremos el siguiente ejemplo: Los siguientes datos le pertenecen a un alumno, NUMERO DE ALUMNO, NOMBRE Y APELLIDO, CARRERA, DOCUMENTO, DIRECCIÓN. Podemos ver que por cada dato genérico alumno, hay una mezcla de datos numéricos y alfabéticos que lo definen unívocamente como tal entidad. Para almacenar esta información heterogénea vamos a usar estructuras de datos compuestas, que reciben el nombre de **registros**. O sea que, bajo un mismo nombre genérico, vamos a manejar un conjunto de datos como un todo, y que serán almacenados sobre variables y/o arreglos de distinto tipo.

Así, el registro nos permitirá almacenar en un conjunto de variables y/o arreglos, información relacionada lógicamente.

Cada una de estas variables y arreglos constituye un *campo* del registro. Un campo es la unidad mínima de información de un registro. Cada campo recibe un nombre que respeta las normas de declaración de variables y arreglos. Además, para cada campo se especifica el *tipo* y la *longitud* de los datos que en ellos se almacenan.
A la variable compuesta del ejemplo anterior la escribiremos de la siguiente manera:

ALUMNO: Registro
NUM: entero 4
NOMYAP: caracter 40
CARR: caracter 40
DOCUM: entero 10
DIREC: caracter 40
Fin Registro

Donde ALUMNO es el nombre del registro, y NUM, NOMYAP, CARR, DOCUM, DIREC el nombre de cada uno de sus campos. Como se puede observar, el registro es una estructura jerárquica.
A diferencia del arreglo, no se exige que cada componente sea del mismo tipo, así como tampoco se accede a cada campo por un subíndice, sino por el nombre del campo.
El registro se maneja por su nombre genérico, ya sea para leerlo o grabarlo. Los campos se usan por su nombre, y calificados por el nombre del registro.
Los nombres de los campos deben ser únicos dentro del registro; pero sí pueden existir dos registros que tengan los mismos nombres de campo. En ese caso es fundamental calificar el nombre de cada campo antes de usarlo, por ejemplo:

ALUMNO: Registro
NUM: entero 4
Fin Registro

RINDE: Registro
NUM: entero
NOTA: entero 2
Fin Registro

Para referirnos al campo NOM del registro ALUMNO, debemos hacerlo así: ALUM.NOM; si se está usando el del registro RINDE, se indica RINDE.NOM.

Los campos de un registro pueden ser de tipo entero, real, caracter, boléanos, arreglos o registros. El uso del registro es muy similar al de las variables; puede asignársele datos, usarlo como argumento de un imprimir, etc. El uso mas destacado del mismo es que sirve para estructurar la información de los archivos: **se lo utiliza como recepción de los valores de un archivo, desde cada registro físico.**
Se define de manera similar a los arreglos; se pueden definir arreglos de datos tipo registro como un registro con **arreglos como componentes**.
Se ordena un arreglo de registros de manera similar al ordenamiento de una tabla por filas.
El registro como estructura de datos es también temporaria.

Los registros se declaran en la zona de Tipos estructurados, especificando su nombre genérico y el nombre, tipo y precisión de cada uno de sus campos.
Nuevamente, el efecto de la declaración es guardar lugar en memoria, en posiciones consecutivas, para almacenar los datos en el registro. Su contenido queda indefinido, y así se mantiene hasta aplicarle una acción de asignación de valor sobre el mismo. El espacio que ocupa un registro en memoria es igual a la suma de los espacios que ocupan cada uno de sus campos.

ARCHIVOS

Hasta el momento hemos visto todas estructuras de datos que nos permitían guardar o mantener datos en la memoria principal de una computadora. Pero existen varias razones por las cuales no siempre es posible mantener toda la información necesaria en dicha memoria, a la vez. Entre ellas podemos mencionar:

- La cantidad de datos que manipula un programa es muy grande y no hay suficiente espacio en la memoria principal;
- Es frecuente que el resultado o salida de un programa sirva de entrada para otro, por lo cual es necesario guardarlo en forma permanente;
- Es engorroso y lleva mucho tiempo tener que cargar datos a cada rato cuando el conjunto de los mismos es muy grande.

Para lograr un mejor y mas provechoso manejo de tales datos, necesitaremos una memoria externa o secundaria. Este dispositivo de memoria externa permite almacenar los datos y recuperarlos mediante operaciones de escritura y lectura respectivamente. Entre esos dispositivos, podemos mencionar al disco duro, los diskettes, cintas, etc..
Para poder hacer uso de esos dispositivos, surge la necesidad de organizar o estructurar los datos que allí se almacenan; con ese fin usaremos los registros; los registros se agruparán, formado archivos.

Un conjunto de registros con ciertos aspectos en común, y organizados para algún propósito particular, constituye un archivo. Al mismo se lo identifica con un nombre genérico.

Físicamente, un archivo se almacena como una sucesión de datos estructurados por el diseño del registro. La información se guarda con el formato especificado por el registro, y se recupera con ese mismo formato.
En forma transparente al usuario, además, la máquina agrega al final de cada registro físico, una marca de fin de registro, y al final de todos los registros, una marca especial llamada MARCA DE FIN DE ARCHIVO, o EOF (end of file).
Si se tiene el siguiente ejemplo:

LIBRO: Registro
TITULO: caracter 30
NRO: entero 5
AUTOR: caracter 39
Fin Registro

ARCH1: Archivo secuencial de registro: LIBRO

Físicamente podemos imaginarlo como:

La información está estructurada por el diseño del registro LIBRO. Sólo se puede recuperar usando ese diseño de registro.

Declaración de archivos

Para declarar los archivos se coloca en la zona de Tipos estructurados el nombre del mismo, acompañado de la estructura de registro que le da su diseño. En pseudocódigo:

Programa Arch

Tipos estructurados

ALUMNO: Registro
NUM: entero 4
NOMYAP: caracter 40
CARR: caracter 40
DOCUM: entero 10
DIREC: caracter 40
Fin Registro

ARCH: archivo secuencial de registro: ALUMNO

Variables

A : ARCH //variable tipo archivo//

Hacer

...
Fin Hacer
Fin Programa

Como vemos, obviamente también debe figurar la declaración del registro de diseño, antes que la del archivo.

Algunos conceptos y definiciones

BASES DE DATOS
Una colección de archivos a los que puede accederse por un conjunto de programas y que contienen, todos ellos, datos relacionados, constituye una base de datos.

ESTRUCTURA JERÁRQUICA
Las estructuras de datos se organizan de modo jerárquico, de modo tal que el nivel mas alto lo constituye la base de datos, y el nivel mas bajo el caracter.

CLAVE
Una clave o indicativo es un campo de datos que identifica al registro, y lo diferencia de otros registros. Esta clave debe ser diferente para cada registro. Claves típicas son nombres o números de identificación.

REGISTRO FÍSICO o BLOQUE
Un registro físico es la cantidad mas pequeña de datos que puede transferirse en una operación de entrada/salida entre la memoria central y los dispositivos periféricos o viceversa.
Un bloque puede contener uno o mas registros lógicos. Un registro lógico puede ocupar menos de un registro físico, un registro físico, o mas de un registro físico.

FACTOR DE BLOQUEO
El número de registros lógicos que puede contener un registro físico se denomina factor de bloqueo o *blocaje*. Se pueden dar las siguientes situaciones:

- *Registro lógico* > *Registro físico*. En un bloque se contienen varios registros físicos por bloque: se denominan registros expandidos.
- *Registro lógico* = *Registro físico.* El factor bloqueo es 1 y se dice que los registros están bloqueados.
- *Registro lógico* < *Registro físico*. El factor de bloqueo es mayor que 1 y los registros están bloqueados.

No se deben incluir todos los registros en un solo bloque ya que las operaciones de entrada/salida que se realizan por bloques, se hacen a través de la memoria central denominada *memoria intermedia (buffer),* y entonces el aumento del bloque implicará el aumento de la memoria intermedia y, por consiguiente, se reducirá el tamaño de la memoria central.

Soportes secuenciales y direccionables

El soporte es el medio físico donde se almacenan los datos. Los tipos de soporte utilizados en la gestión de archivos son:
-Soportes secuenciales.
-Soportes direccionables.

Los soportes secuenciales son aquellos en los que los registros están escritos unos a continuación de otros, y para acceder aun determinado registro n se necesita pasar por los n-1 registros anteriores. La secuencia puede corresponder al orden físico de los registros en el archivo o bien al orden de claves (ascendente o descendente) de los registros.

Los soportes direccionables se estructuran de modo tal que la información registrada se puede localizar directamente por su dirección y no se requiere pasar por los registros precedentes. En estos soportes, los registros deben poseer un campo clave que los diferencie del resto de los registros del archivo. Una dirección en un soporte direccionable puede ser un número de pista y un número de sector en un disco.
Los soportes direccionables son los discos magnéticos, aunque pueden actuar como soporte secuencial.

Organización de archivos

Según las características del soporte empleado, y el modo en que se han organizado los registros, se consideran dos tipos de acceso a los registros de un archivo:

- Acceso secuencial: implica el acceso a un archivo según el orden de almacenamiento de sus registros, uno tras otro.
- Acceso directo: Implica el acceso a un registro determinado, sin que ello implique la consulta de los registros precedentes.

La organización de un archivo define la forma en que los registros se disponen sobre el soporte de almacenamiento, o también se define la organización como la forma en que se

estructuran los datos de un archivo. En general, se consideran tres organizaciones fundamentales:

ORGANIZACIÓN SECUENCIAL
Un archivo con organización secuencial es una sucesión de registros almacenados consecutivamente sobre el soporte externo, de tal modo que para acceder a un registro N dado, es obligatorio pasar por todos los N-1 artículos que le preceden.
Los ficheros organizados secuencialmente contienen un registro particular (el último) que contiene una marca de fin de archivo.

ORGANIZACIÓN DIRECTA
Un archivo está organizado en modo directo cuando el orden físico no se corresponde con el orden lógico. Los datos se sitúan en el archivo y se accede a ellos aleatoriamente mediante su posición, es decir, el lugar relativo que ocupan.
La organización directa tiene el inconveniente de que se necesita programar la relación existente entre el contenido de un registro y la posición que ocupa. El acceso a los registros en modo directo implica la posible existencia de huecos libres dentro del soporte, y por consiguiente, pueden existir huecos libres entre registros.
Las condiciones para que un archivo sea de organización directa son:
-debe estar almacenado en un soporte direccionable;
-los registros deben contener un campo específico denominado clave que identifique a cada registro de modo único;
-debe haber una correspondencia entre los posibles valores de la clave y las direcciones disponibles sobre el soporte.

ORGANIZACIÓN SECUENCIAL INDEXADA
Un diccionario es un archivo secuencial, cuyos registros son las entradas y cuyas claves son las palabras definidas por las entradas. Para buscar una palabra (clave) no se busca secuencialmente desde la “a” hasta la ”z”, sino que se abre el diccionario por la letra inicial de la palabra. El diccionario es un ejemplo típico de archivo secuencial indexado con dos niveles de índices, el nivel superior para las letras iniciales y el nivel menor para las cabeceras de página. En una organización de computadora, las letras y las cabeceras de páginas se guardarán en un archivo de índice independiente de las entradas del diccionario (archivo de datos). Por consiguiente, cada archivo secuencial indexado consta de un archivo índice y un archivo de datos, y consta de las siguientes partes:

- Área de datos primaria: Contiene los registros en forma secuencial y está organizada en secuencia de claves sin dejar huecos intercalados;
- Área de índices: Es una tabla que contiene los niveles de índice; la existencia de varios índices enlazados se denomina nivel de indexación.
- Área de excedentes: Utilizada, si fuese necesario, para las actualizaciones.

Operaciones sobre archivos

Las distintas operaciones que se pueden realizar son:

CREACIÓN
Para utilizar un archivo, éste tiene que existir, es decir, la información de este archivo tiene que haber sido almacenada sobre un soporte; la creación exige organización, estructura, localización o reserva de espacio en el soporte de almacenamiento, transferencia del soporte antiguo al nuevo.

CONSULTA
Es la operación que permite al usuario acceder al archivo de datos para conocer el contenido de uno, varios o todos los registros

ACTUALIZACIÓN
Es la operación que permite tener actualizado el archivo.

CLASIFICACIÓN
Se realizará de acuerdo con el valor de un campo específico, pudiendo ser ascendente o descendente: alfabética o numéricamente.

REORGANIZACIÓN
Suele consistir en la copia de un nuevo archivo a partir del archivo modificado, a fin de obtener una nueva estructura lo mas óptima posible.

DESTRUCCIÓN
Es la operación inversa a la creación de un archivo. Cuando se destruye un archivo, éste ya no se puede utilizar y, por consiguiente, no se podrá acceder a ninguno de sus registros.

REUNIÓN; FUSIÓN
Reunión: Permite obtener un archivo a partir de otros varios.
Fusión: Se realiza cuando se reúnen varios archivos en unos solo, intercalándose unos en otros, siguiendo ciertos criterios.

ROTURA / ESTALLIDO
Es la operación de obtener varios archivos a partir de un mismo archivo inicial.

Gestión de archivos

Las operaciones mas usuales en los registros son:
-Consulta: lectura del contenido de un registro;
-Modificación: alterar la información contenida en un registro;
-Inserción: añadir un nuevo registro al archivo;
-Borrado: suprimir un registro del archivo.

CREAR UN ARCHIVO
Para crear un nuevo archivo dentro de un sistema de computadora se necesitan los siguientes datos:
-*nombre del dispositivo/usuario*: indica el lugar donde se situará el archivo cuando se cree
-*nombre del archivo*: identifica al archivo entre los restantes archivos de una computadora
-*tamaño del archivo*: indica el espacio necesario para la creación del archivo
-*organización del archivo:* tipo de organización del archivo
tamaño del bloque o registro físico : cantidad de datos que se leen o escriben en cada operación de entrada/salida.

ABRIR UN ARCHIVO
Permite al usuario localizar y acceder a los archivos que fueron creados anteriormente. La instrucción de abrir un archivo consiste en la creación de un canal que comunica a un usuario a través de un programa con el archivo correspondiente situado en un soporte. En pseudocódigo:

Abrir: (ARCH, REG)

Es decir, abrir el archivo dentro de un registro que coincide con el diseño físico del archivo.

CERRAR UN ARCHIVO
El propósito de la operación de cerrar un archivo, es permitir al usuario cortar el acceso o detener el uso del archivo, permitiendo a otros usuarios acceder al archivo. En pseudocódigo:

Cerrar (ARCH)

LECTURA

Leer: REG

Las operaciones de lectura y escritura sobre el archivo corren el puntero de direccionamiento un lugar, es decir lo dejan apuntando al registro inmediatamente siguiente al corriente, en forma automática, y luego de haberse ejecutado la operación correspondiente (leer/grabar).

Características de un archivo

TAMAÑO
Se refiere a la cantidad de registros que tiene el archivo, y al espacio que ocupa el mismo en memoria. Espacio = tamaño de un registro * cantidad de registros.

ACTIVIDAD
Se refiere a determinar con que frecuencia se usa el archivo y cuántos registros del mismo se utilizan.

CRECIMIENTO
Es el tamaño que se prevé que puede llegar a tener el archivo al cabo de su vida útil, en cuanto a la cantidad de registros.

LUGAR DE RESIDENCIA
Se refiere al lugar donde se almacena el archivo, y depende específicamente del tamaño del archivo, de su actividad, crecimiento y del método de acceso (EJ: en cinta se guardan archivos grandes, de baja actividad y gran crecimiento).

ORGANIZACIÓN
Se refiere a cómo se organiza el lugar de residencia o almacenamiento.

TÉCNICAS DE ACCESO
Se refiere a qué método de acceso se usa para recuperar os datos del archivo. Secuencial: se procesa un registro detrás de otro sin saltear ninguno; Directo: se lee o graba por calve o índice el registro deseado, independientemente de los otros.

Todo archivo de organización secuencial se accede por métodos secuenciales, y los de organización directa admiten tanto una técnica como la otra.

EJ8) Programa con registros y archivos.

Programa: Horario

Tipos Estructurados

REG1: Registro
Nombre: caracter 20
Prioridad: entero 2
Modulo: entero 1
Dia: entero 1
Fin Registro

ARCH: archivo secuencial de registro: REG1

DAT: Registro
Solicitud: entero 2
Dia: entero 1
Menor: caracter 30
Mayor: caracter 30
Fin Registro

CONTROL= ARREGLO [1..5]: DAT //arreglo de registros DAT (vector)//

Variables

A: ARCH //variable tipo archivo//
R: REG1 //variable tipo registro//
INFO: CONTROL //variable tipo arreglo vector//

Hacer

Abrir: (A, R)

Leer: (R)

Repetir Mientras ***(no EOF(A)*** *)* //mientras hayan registros físicos//

Leer: (R)

.....

....

Fin Repetir Mientras

Cerrar (A)

Fin Hacer

Fin Programa

CAPITULO XI

CONTROL Y ERRORES USUALES

CORTE DE CONTROL

Consiste en **emitir resultados** luego de finalizado un proceso **de acuerdo a una variable**, y reiniciar otras variables para reutilizar, hasta que finaliza el archivo.
Se trabaja con un archivo de organización secuencial ordenado por uno o dos de sus campos. Se utilizan archivos de sólo lectura.

EJ9) Se tiene un AOS (archivo organización secuencial) en el cual se encuentran los datos de alumnos de distintas escuelas. Se sabe que existen 102 escuelas en La Plata.
Se desea ordenar el archivo por código de establecimiento, sabiendo que el diseño físico del mismo es el siguiente:

DNI: entero 10
Nombre: caracter 20
Apellido: caracter 20
Cod Establecimiento: caracter 20 //2 letras y 2 números respectivamente//
Localidad: caracter 20

Programa: Escuelas

Tipos Estructurados

R= Registro
DNI: entero 10
Nombre: caracter 20
Apellido: caracter 20
Cod_Est: caracter 20
Localidad: caracter 20
Fin Registro

VEC= ARREGLO [1..8160]: R

ARCH: Archivo de organización secuencial: R

Procedimientos y Funciones

Procedimiento: CARGA (Ref: S: vec, Ref: H : entero 4, AUXA: R)

Hacer
S[H].DNI:= AUXA. DNI
S[H].Nombre:=AUXA.Nombre
S[H].Apellido:=AUXA. Apellido
S[H].Cod_Est:=AUXA.Cod_Est
S[H].Localidad:=AUXA.Localidad

H:=H + 1
Fin Hacer

```
   Fin Procedimiento

 Variables
   AUX: R
   A: VEC
   AR: ARCH
   I: entero 4
 Hacer
   Abrir: (AR,AUX)

   Leer: (AUX)

   I:= 1
     Repetir Mientras(not EOF(AR) and I <= 8160)

          Si: (AUX.localidad = "La Plata") Entonces
                                          CARGA (A, I, AUX)
          Fin Si
          Leer: (AUX)
     Fin Repetir Mientras
      ORDENAR (AR)          //proced. no declarado...//

     Cerrar (AR)
  Fin Hacer
Fin Programa
```

CLASIFICACIÓN DE TIPOS DE ERRORES

TIPOS DE ERRORES

Declaración (inicialización)	- No declarar una variable importante **-** Declarar con incorrecto tipo de dato - Valores iniciales incorrectos
Referencias de variables	-identificador no conocido -Índices fuera de rango ó no enteros
Computación	- Asignar un valor de diferente tipo - División por cero - Un valor mayor al esperado
Comparación	-Expresiones lógicas incompletas -Error entre comparaciones y valores lógicos
Control de procesos	- Bucles infinitos = programas sin finalización - Bucle que nunca se ejecuta - Anidamientos mal cerrados
Interfases	-Diferente cantidad de parámetros reales y formales -Diferente tipo de dato entre el parámetro formal y real asociado -Uso erróneo de var. globales (efecto lateral)
Entrada / Salida (memoria)	- Diferente estructura de datos con respecto al registro físico - Uso erróneo de instrucciones sobre archivos