

Fundamentos de la Programación

Contenido

INTRODUCCION.....	2
CAPITULO I. CONCEPTOS BÁSICOS Y METODOLOGÍA PARA LA SOLUCIÓN DE PROBLEMAS POR MEDIO DE COMPUTADORAS.	3
1.1 Introducción	4
1.2 Definición de Lenguaje.....	6
1.3 Definición de Algoritmo	7
1.4 Tipos de Algoritmos	7
1.5 Lenguajes Algorítmicos	7
CAPITULO II.....	10
ENTIDADES PRIMITIVAS PARA EL DESARROLLO DE	10
ALGORITMOS	10
2.1 Tipos De Datos	11
2.2 Expresiones	11
2.3 Operadores y Operandos.....	12
2.4 Identificadores	16
CAPITULO III. TÉCNICAS DE DISEÑO.....	18
CAPITULO IV. TÉCNICAS PARA LA FORMULACIÓN DE ALGORITMOS	20
4.1 Diagrama de Flujo	21
4.2 Pseudocódigo	22
CAPITULO V. ESTRUCTURAS ALGORITMICAS	24
5.2 Condicionales	24
5.3 Repetición fila condicional.....	24
5.1. Estructuras Secuenciales.....	25
5.2 Estructuras de Condicionales.....	28
5.3. Estructuras Cíclicas.....	36
CAPITULO VI. ARREGLOS	40
6.1. Vectores	41
6.2 Matriz.....	42
CAPITULO VII. PROGRAMACIÓN ESTRUCTURADA	44
6.1 Introducción	45
6.2 Tipos de módulos.....	46

INTRODUCCION

El desarrollo de algoritmos es un tema fundamental en el diseño de programas por lo cual el alumno debe tener buenas bases que le sirvan para poder desarrollar de manera fácil y rápida sus programas.

Estos apuntes servirán de apoyo al catedrático del Instituto Tecnológico de Tuxtepec, en su labor cotidiana de enseñanza y al estudiante le facilitará desarrollar su capacidad analítica y creadora, para de esta manera mejorar su destreza en la elaboración de algoritmos que sirven como base para la codificación de los diferentes programas que tendrá que desarrollar a lo largo de su carrera.

CAPITULO I. CONCEPTOS BÁSICOS Y METODOLOGÍA PARA LA SOLUCIÓN DE PROBLEMAS POR MEDIO DE COMPUTADORAS.

1.1 Introducción

-De los problemas a los programas

-Breves prácticas de programación

1.2 Definición de lenguaje

1.3 Definición de algoritmo

1.4 Algoritmos cotidianos

1.5 Definición de lenguajes algorítmicos

1.6 Metodología para la solución de problemas por medio de computadora

-Definición del problema

-Análisis del problema

-Diseño del algoritmo

-Codificación

-Prueba y depuración

-Documentación

-Mantenimiento

OBJETIVO:

El alumno:

- Conocerá la terminología relacionada con los algoritmos; así como la importancia de aplicar técnicas adecuadas de programación.
- Conocerá la metodología en cada una de sus etapas.

1.1 Introducción

La computadora no solamente es una máquina que puede realizar procesos para darnos resultados, sin que tengamos la noción exacta de las operaciones que realiza para llegar a esos resultados. Con la computadora además de lo anterior también podemos diseñar soluciones a la medida, de problemas específicos que se nos presenten. Más aún, si estos involucran operaciones matemáticas complejas y/o repetitivas, o requieren del manejo de un volumen muy grande de datos.

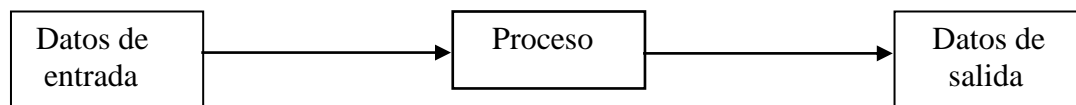
El diseño de soluciones a la medida de nuestros problemas, requiere como en otras disciplinas una metodología que nos enseñe de manera gradual, la forma de llegar a estas soluciones.

A las soluciones creadas por computadora se les conoce como **programas** y no son más que una serie de operaciones que realiza la computadora para llegar a un resultado, con un grupo de datos específicos. Lo anterior nos lleva al razonamiento de que un **programa** nos sirve para solucionar un problema específico.

Para poder realizar **programas**, además de conocer la metodología mencionada, también debemos de conocer, de manera específica las funciones que puede realizar la computadora y las formas en que se pueden manejar los elementos que hay en la misma.

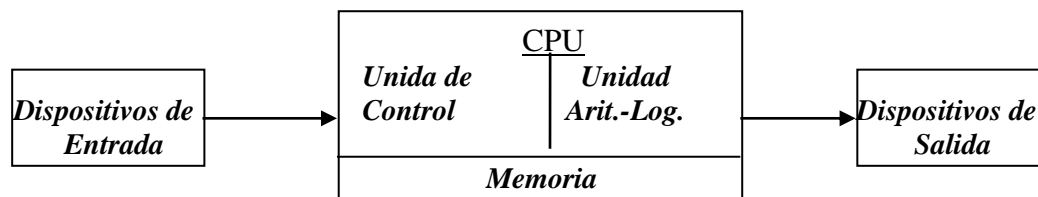
Computadora: Es un dispositivo electrónico utilizado para procesar información y obtener resultados. Los datos y la información se pueden introducir en la computadora como entrada (input) y a continuación se procesan para producir una salida (output).

Proceso de información en la computadora



Programa: Es el conjunto de instrucciones escritas de algún lenguaje de programación y que ejecutadas secuencialmente resuelven un problema específico.

Organización física de una computadora



Dispositivos de Entrada: Como su nombre lo indica, sirven para introducir datos (información) en la computadora para su proceso. Los datos se leen de los dispositivos de entrada y se almacenan en la memoria central o interna. Ejemplos: teclado , scanners (digitalizadores de rastreo), mouse (ratón), trackball (bola de ratón estacionario), joystick (palancas de juego), lápiz óptico.

Dispositivos de Salida: Regresan los datos procesados que sirven de información al usuario. Ejemplo: monitor, impresora.

La Unidad Central de Procesamiento (C.P.U) se divide en dos:

- Unidad de control
- Unidad Aritmético - Lógica

Unidad de Control: Coordina las actividades de la computadora y determina que operaciones se deben realizar y en que orden; así mismo controla todo el proceso de la computadora.

Unidad Aritmético - Lógica: Realiza operaciones aritméticas y lógicas, tales como suma, resta, multiplicación, división y comparaciones.

La Memoria de la computadora se divide en dos:

- Memoria Central o Interna
- Memoria Auxiliar o Externa

Memoria Central (interna): La CPU utiliza la memoria de la computadora para guardar información mientras trabaja con ella; mientras esta información permanezca en memoria, la computadora puede tener acceso a ella en forma directa. Esta memoria construida internamente se llama memoria de acceso aleatorio (RAM).

La **memoria interna** consta de dos áreas de memoria:

La memoria **RAM (Random Access Memory)**: Recibe el nombre de memoria principal o memoria del usuario, en ella se almacena información solo mientras la computadora esta encendida. Cuando se apaga o arranca nuevamente la computadora, la información se pierde, por lo que se dice que la memoria RAM es una memoria volátil.

La memoria **ROM (Read Only Memory)**: Es una memoria estática que no puede cambiar, la computadora puede leer los datos almacenados en la memoria ROM, pero no se pueden introducir datos en ella, o cambiar los datos que ahí se encuentran; por lo que se dice que esta memoria es de solo lectura. Los datos de la memoria ROM están grabados en forma permanente y son introducidos por el fabricante de la computadora.

Memoria Auxiliar (Externa): Es donde se almacenan todos los programas o datos que el usuario desee. Los dispositivos de almacenamiento o memorias auxiliares (externas o secundarias) mas comúnmente utilizados son: cintas magnéticas y discos magnéticos.

1.2 Definición de Lenguaje

Lenguaje: Es una serie de símbolos que sirven para transmitir uno o mas mensajes (ideas) entre dos entidades diferentes. A la transmisión de mensajes se le conoce comúnmente como **comunicación**.

La **comunicación** es un proceso complejo que requiere una serie de reglas simples, pero indispensables para poderse llevar a cabo. Las dos principales son las siguientes:

- * Los mensajes deben correr en un sentido a la vez.
- * Debe forzosamente existir 4 elementos: Emisor, Receptor, Medio de Comunicación y Mensaje.

Lenguajes de Programación

Es un conjunto de símbolos, caracteres y reglas (programas) que le permiten a las personas comunicarse con la computadora.

Los lenguajes de programación tienen un conjunto de instrucciones que nos permiten realizar operaciones de entrada/salida, calculo, manipulación de textos, lógica/comparación y almacenamiento/recuperación.

Los lenguajes de programación se clasifican en:

➤ **Lenguaje Maquina:** Son aquellos cuyas instrucciones son directamente entendibles por la computadora y no necesitan traducción posterior para que la CPU pueda comprender y ejecutar el programa. Las instrucciones en lenguaje maquina se expresan en términos de la unidad de memoria mas pequeña el bit (dígito binario 0 o 1).

➤ **Lenguaje de Bajo Nivel (Ensamblador):** En este lenguaje las instrucciones se escriben en códigos alfabéticos conocidos como mnemotécnicos para las operaciones y direcciones simbólicas.

➤ **Lenguaje de Alto Nivel:** Los lenguajes de programación de alto nivel (BASIC, pascal, cobol, fortran, etc.) son aquellos en los que las instrucciones o sentencias a la computadora son escritas con palabras similares a los lenguajes humanos (en general en ingles), lo que facilita la escritura y comprensión del programa.

1.3 Definición de Algoritmo

La palabra algoritmo se deriva de la traducción al latín de la palabra árabe alkhwarizmi, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX.

Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.

1.4 Tipos de Algoritmos

- ***Cualitativos:*** Son aquellos en los que se describen los pasos utilizando palabras.
- ***Cuantitativos:*** Son aquellos en los que se utilizan cálculos numéricos para definir los pasos del proceso.

1.5 Lenguajes Algorítmicos

Es una serie de símbolos y reglas que se utilizan para describir de manera explícita un proceso.

Tipos de Lenguajes Algorítmicos

- ***Gráficos:*** Es la representación gráfica de las operaciones que realiza un algoritmo (diagrama de flujo).
- ***No Gráficos:*** Representa en forma descriptiva las operaciones que debe realizar un algoritmo (pseudocódigo).

1.6 Metodología para la solución de problemas por medio de computadora

Definición del Problema

Esta fase está dada por el enunciado del problema, el cual requiere una definición clara y precisa. Es importante que se conozca lo que se desea que realice la computadora; mientras esto no se conozca del todo no tiene mucho caso continuar con la siguiente etapa.

Análisis del Problema

Una vez que se ha comprendido lo que se desea de la computadora, es necesario definir:

Los datos de entrada.

Cual es la información que se desea producir (salida)

Los métodos y fórmulas que se necesitan para procesar los datos.

Una recomendación muy practica es el que nos pongamos en el lugar de la computadora y analicemos que es lo que necesitamos que nos ordenen y en que secuencia para producir los resultados esperados.

Diseño del Algoritmo

Las características de un buen algoritmo son:

Debe tener un punto particular de inicio.

Debe ser definido, no debe permitir dobles interpretaciones.

Debe ser general, es decir, soportar la mayoría de las variantes que se puedan presentar en la definición del problema.

Debe ser finito en tamaño y tiempo de ejecución.

Codificación

La codificación es la operación de escribir la solución del problema (de acuerdo a la lógica del diagrama de flujo o pseudocódigo), en una serie de instrucciones detalladas, en un código reconocible por la computadora, la serie de instrucciones detalladas se le conoce como código fuente, el cual se escribe en un lenguaje de programación o lenguaje de alto nivel.

Prueba y Depuración

Los errores humanos dentro de la programación de computadoras son muchos y aumentan considerablemente con la complejidad del problema. El proceso de identificar y eliminar errores, para dar paso a una solución sin errores se le llama ***depuración***.

La ***depuración o prueba*** resulta una tarea tan creativa como el mismo desarrollo de la solución, por ello se debe considerar con el mismo interés y entusiasmo.

Resulta conveniente observar los siguientes principios al realizar una depuración, ya que de este trabajo depende el éxito de nuestra solución.

Documentación

Es la guía o comunicación escrita es sus variadas formas, ya sea en enunciados, procedimientos, dibujos o diagramas.

A menudo un programa escrito por una persona, es usado por otra. Por ello la documentación sirve para ayudar a comprender o usar un programa o para facilitar futuras modificaciones (mantenimiento).

La **documentación** se divide en tres partes:

Documentación Interna
Documentación Externa
Manual del Usuario

- Documentación Interna: Son los comentarios o mensaje que se añaden al código fuente para hacer mas claro el entendimiento de un proceso.
- Documentación Externa: Se define en un documento escrito los siguientes puntos:
 - Descripción del Problema
 - Nombre del Autor
 - Algoritmo (diagrama de flujo o pseudocodigo)
 - Diccionario de Datos
 - Código Fuente (programa)
- Manual del Usuario: Describe paso a paso la manera como funciona el programa, con el fin de que el usuario obtenga el resultado deseado.

Mantenimiento

Se lleva acabo después de terminado el programa, cuando se detecta que es necesario hacer algún cambio, ajuste o complementación al programa para que siga trabajando de manera correcta. Para poder realizar este trabajo se requiere que el programa este correctamente documentado.

CAPITULO II.

***ENTIDADES PRIMITIVAS PARA EL DESARROLLO DE
ALGORITMOS***

- 2.1 Tipos de datos
- 2.2 Expresiones
- 2.3 Operadores y operandos
- 2.4 Identificadores como localidades de memoria

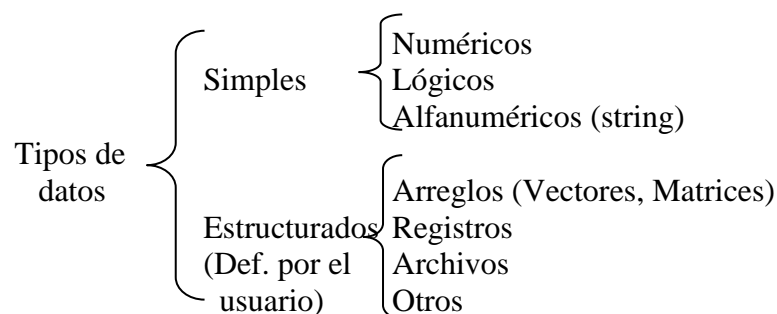
OBJETIVO:

El alumno:

- Conocerá las reglas para cambiar fórmulas matemáticas a expresiones válidas para la computadora, además de diferenciar constantes e identificadores y tipos de datos simples.

2.1 Tipos De Datos

Todos los datos tienen un tipo asociado con ellos. Un dato puede ser un simple carácter, tal como 'b', un valor entero tal como 35. El tipo de dato determina la naturaleza del conjunto de valores que puede tomar una variable.



Tipos de Datos Simples

- **Datos Numéricos:** Permiten representar valores escalares de forma numérica, esto incluye a los números enteros y los reales. Este tipo de datos permiten realizar operaciones aritméticas comunes.
- **Datos Lógicos:** Son aquellos que solo pueden tener dos valores (cierto o falso) ya que representan el resultado de una comparación entre otros datos (numéricos o alfanuméricos).
- **Datos Alfanuméricos (String):** Es una secuencia de caracteres alfanuméricos que permiten representar valores identificables de forma descriptiva, esto incluye nombres de personas, direcciones, etc. Es posible representar números como alfanuméricos, pero estos pierden su propiedad matemática, es decir no es posible hacer operaciones con ellos. Este tipo de datos se representan encerrados entre comillas.

Ejemplo:

“Instituto Tecnológico de Tuxtepec”
“1997”

2.2 Expresiones

Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales. Por ejemplo:

$$a+(b+3)/c$$

Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas.

Una expresión consta de operadores y operandos. Según sea el tipo de datos que manipulan, se clasifican las expresiones en:

- Aritméticas
- Relacionales
- Lógicas

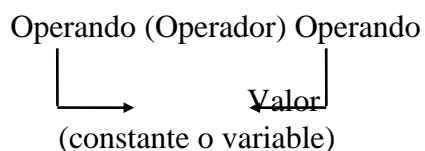
2.3 Operadores y Operandos

➤ **Operadores:** Son elementos que relacionan de forma diferente, los valores de una o mas variables y/o constantes. Es decir, los operadores nos permiten manipular valores.



➤ **Operadores Aritméticos:** Los operadores aritméticos permiten la realización de operaciones matemáticas con los valores (variables y constantes).

Los operadores aritméticos pueden ser utilizados con tipos de datos enteros o reales. Si ambos son enteros, el resultado es entero; si alguno de ellos es real, el resultado es real.



Operadores Aritméticos

+	Suma
-	Resta
*	Multiplicación
/	División
Mod	Modulo (residuo de la división entera)

Ejemplos:

Expresión	Resultado
7 / 2	3.5

$$\begin{array}{ll} 12 \bmod 7 & 5 \\ 4 + 2 * 5 & 14 \end{array}$$

Prioridad de los Operadores Aritméticos

□ Todas las expresiones entre paréntesis se evalúan primero. Las expresiones con paréntesis anidados se evalúan de dentro a fuera, el paréntesis mas interno se evalúa primero.

□ Dentro de una misma expresión los operadores se evalúan en el siguiente orden.

- 1.- ^ Exponenciación
- 2.- *, /, mod Multiplicación, división, modulo.
- 3.- +, - Suma y resta.

□ Los operadores en una misma expresión con igual nivel de prioridad se evalúan de izquierda a derecha.

Ejemplos:

$$\begin{array}{ll} 4 + 2 * 5 = 14 & 46 / 5 = 9.2 \\ 23 * 2 / 5 = 9.2 & 3 + 5 * (10 - 6) = 3 + 5 * 4 = 3 + 20 = 23 \\ 3 + 5 * (10 - (2 + 4)) = 23 & 3.5 + 5.09 - 14.0 / 40 = 5.09 \\ 3.5 + 5.09 - 14.0 / 40 = 5.09 & 2.1 * (1.5 + 3.0 * 4.1) = 28.98 \\ 2.1 * (1.5 + 3.0 * 4.1) = 28.98 & 2.1 * (1.5 + 12.3) = 2.1 * 13.8 = 28.98 \end{array}$$

➤ Operadores Relacionales:

- Se utilizan para establecer una relación entre dos valores.
- Compara estos valores entre si y esta comparación produce un resultado de certeza o falsedad (verdadero o falso).
- Los operadores relacionales comparan valores del mismo tipo (numéricos o cadenas)
- Tienen el mismo nivel de prioridad en su evaluación.
- Los operadores relacionales tiene menor prioridad que los aritméticos.

Operadores Relacionales

>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
<>	Diferente
=	Igual

Ejemplos:

$$\text{Si } a = 10 \quad b = 20 \quad c = 30$$

$a + b > c$	Falso
$a - b < c$	Verdadero
$a - b = c$	Falso
$a * b < > c$	Verdadero

Ejemplos no lógicos:

$a < b < c$

$10 < 20 < 30$

$T < 30$ (no es lógico porque tiene diferentes operandos)

➤ **Operadores Lógicos:**

- Estos operadores se utilizan para establecer relaciones entre valores lógicos.
- Estos valores pueden ser resultado de una expresión relacional.

Operadores Lógicos

And	Y
Or	O
Not	Negación

Operador And

<i>Operando1</i>	<i>Operador</i>	<i>Operando2</i>	<i>Resultado</i>
T	AND	T	T
T		F	F
F		T	F
F		F	F

Operador Or

<i>Operando1</i>	<i>Operador</i>	<i>Operando2</i>	<i>Resultado</i>
T	OR	T	T
T		F	T
F		T	T
F		F	F

Operador Not

<i>Operando</i>	<i>Resultado</i>
T	F
F	T

Ejemplos:

Fundamentos de la Programación

$(a < b) \text{ and } (b < c)$
 $(10 < 20) \text{ and } (20 < 30)$
T and T
└─→ T ←─┘

Prioridad de los Operadores Lógicos

Not
And
Or

Prioridad de los Operadores en General

- 1.- ()
- 2.- ^
- 3.- *, /, Mod, Not
- 4.- +, -, And
- 5.- >, <, >=, <=, <>, =, Or

Ejemplos:

a = 10 b = 12 c = 13 d = 10

1) $((a > b) \text{ or } (a < c)) \text{ and } ((a = c) \text{ or } (a > = b))$
F T F F
└─ T ─┘ └─ F ─┘
└──────── F ─────────┘

2) $((a > = b) \text{ or } (a < d)) \text{ and } ((a > = d) \text{ and } (c > d))$
F F T T
└─ F ─┘ └─ T ─┘
└──────── F ─────────┘

3) $\text{not } (a = c) \text{ and } (c > b)$
T F T
└─ T ─┘
└──────── T ─────────┘

2.4 Identificadores

Los *identificadores* representan los datos de un programa (constantes, variables, tipos de datos). Un identificador es una secuencia de caracteres que sirve para identificar una posición en la memoria de la computadora, que nos permite acceder a su contenido.

Ejemplo: Nombre
 Num_hrs
 Calif2

Reglas para formar un Identificador

- ❑ Debe comenzar con una letra (A a Z, mayúsculas o minúsculas) y no deben contener espacios en blanco.
- ❑ Letras, dígitos y caracteres como la subraya (_) están permitidos después del primer carácter.
- ❑ La longitud de identificadores puede ser de hasta 8 caracteres.

Constantes y Variables

- **Constante:** Una constante es un dato numérico o alfanumérico que no cambia durante la ejecución del programa.

Ejemplo:

pi = 3.1416

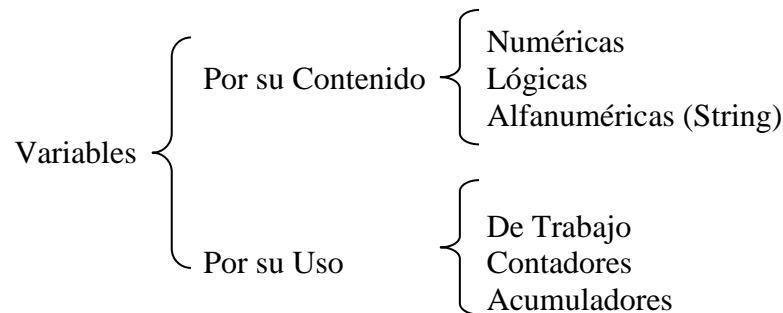
- **Variable:** Es un espacio en la memoria de la computadora que permite almacenar temporalmente un dato durante la ejecución de un proceso, su contenido puede cambiar durante la ejecución del programa. Para poder reconocer una variable en la memoria de la computadora, es necesario darle un nombre con el cual podamos identificarla dentro de un algoritmo.

Ejemplo:

área = pi * radio ^ 2

Las variables son : el radio, el área y la constante es pi

Clasificación de las Variables



Por su Contenido

➤ **Variable Numéricas:** Son aquellas en las cuales se almacenan valores numéricos, positivos o negativos, es decir almacenan números del 0 al 9, signos (+ y -) y el punto decimal. Ejemplo:

iva=0.15 pi=3.1416 costo=2500

➤ **Variables Lógicas:** Son aquellas que solo pueden tener dos valores (cierto o falso) estos representan el resultado de una comparación entre otros datos.

➤ **Variables Alfanuméricas:** Esta formada por caracteres alfanuméricos (letras, números y caracteres especiales). Ejemplo:

letra='a' apellido='lopez' direccion='Av. Libertad #190'

Por su Uso

➤ **Variables de Trabajo:** Variables que reciben el resultado de una operación matemática completa y que se usan normalmente dentro de un programa. Ejemplo:

suma=a+b/c

➤ **Contadores:** Se utilizan para llevar el control del numero de ocasiones en que se realiza una operación o se cumple una condición. Con los incrementos generalmente de uno en uno.

➤ **Acumuladores:** Forma que toma una variable y que sirve para llevar la suma acumulativa de una serie de valores que se van leyendo o calculando progresivamente.

CAPITULO III. TÉCNICAS DE DISEÑO

3.1 Top down

3.2 Bottom up

OBJETIVO:

El alumno:

- Conocerá las características de las técnicas de diseño más empleadas, así como su aplicación a cada tipo de problemas

3.1 *Top Down*

También conocida como de arriba-abajo y consiste en establecer una serie de niveles de mayor a menor complejidad (arriba-abajo) que den solución al problema. Consiste en efectuar una relación entre las etapas de la estructuración de forma que una etapa jerárquica y su inmediato inferior se relacionen mediante entradas y salidas de información.

Este diseño consiste en una serie de descomposiciones sucesivas del problema inicial, que recibe el refinamiento progresivo del repertorio de instrucciones que van a formar parte del programa.

La utilización de la técnica de diseño ***Top-Down*** tiene los siguientes objetivos básicos:

- Simplificación del problema y de los subprogramas de cada descomposición.
- Las diferentes partes del problema pueden ser programadas de modo independiente e incluso por diferentes personas.
- El programa final queda estructurado en forma de bloque o módulos lo que hace más sencilla su lectura y mantenimiento.

3.2 *Bottom Up*

El diseño ascendente se refiere a la identificación de aquellos procesos que necesitan computarizarse con forme vayan apareciendo, su análisis como sistema y su codificación, o bien, la adquisición de paquetes de software para satisfacer el problema inmediato.

Cuando la programación se realiza internamente y haciendo un enfoque ascendente, es difícil llegar a integrar los subsistemas al grado tal de que el desempeño global, sea fluido. Los problemas de integración entre los subsistemas son sumamente costosos y muchos de ellos no se solucionan hasta que la programación alcanza la fecha límite para la integración total del sistema. En esta fecha, ya se cuenta con muy poco tiempo, presupuesto o paciencia de los usuarios, como para corregir aquellas delicadas interfaces, que en un principio, se ignoran.

Aunque cada subsistema parece ofrecer lo que se requiere, cuando se contempla al sistema como una entidad global, adolece de ciertas limitaciones por haber tomado un enfoque ascendente. Uno de ellos es la duplicación de esfuerzos para acceder el software y más aun al introducir los datos. Otro es, que se introducen al sistema muchos datos carentes de valor. Un tercero y tal vez el mas serio inconveniente del enfoque ascendente, es que los objetivos globales de la organización no fueron considerados y en consecuencia no se satisfacen.

CAPITULO IV. TÉCNICAS PARA LA FORMULACIÓN DE ALGORITMOS

- 4.1 Diagrama de flujo
- 4.2 Pseudocódigo
- 4.3 Diagrama estructurado (nassi-schneiderman)

OBJETIVO:

El alumno:

- Será capaz de diferenciar los métodos de representación y formulación de algoritmos, así como de conocer las características más importantes de cada técnica.

Las dos herramientas utilizadas comúnmente para diseñar algoritmos son:

Diagrama de Flujo


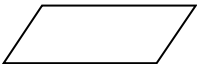

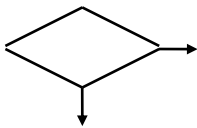
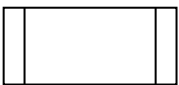

Pseudocódigo

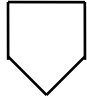
4.1 Diagrama de Flujo

Un diagrama de flujo es la representación gráfica de un algoritmo. También se puede decir que es la representación detallada en forma gráfica de como deben realizarse los pasos en la computadora para producir resultados.

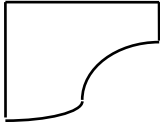
Esta representación gráfica se da cuando varios símbolos (que indican diferentes procesos en la computadora), se relacionan entre si mediante líneas que indican el orden en que se deben ejecutar los procesos.

Los símbolos utilizados han sido normalizados por el instituto norteamericano de normalización (ANSI).

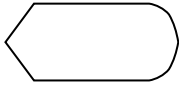
<u>SÍMBOLO</u>	<u>DESCRIPCIÓN</u>
	Indica el inicio y el final de nuestro diagrama de flujo.
	Indica la entrada y salida de datos.
	Símbolo de proceso y nos indica la asignación de un valor en la memoria y/o la ejecución de una operación aritmética.
	Símbolo de decisión indica la realización de una comparación de valores.
	Se utiliza para representar los subprogramas.
	Conector dentro de pagina. Representa la continuidad del diagrama dentro de la misma pagina.



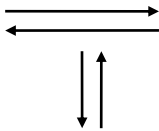
Conector fuera de pagina. Representa la continuidad del diagrama en otra pagina.



Indica la salida de información por impresora.



Indica la salida de información en la pantalla o monitor.



Líneas de flujo o dirección. Indican la secuencia en que se realizan las operaciones.

Recomendaciones para el diseño de Diagramas de Flujo

- ☐ Se deben usar solamente líneas de flujo horizontales y/o verticales.
- ☐ Se debe evitar el cruce de líneas utilizando los conectores.
- ☐ Se deben usar conectores solo cuando sea necesario.
- ☐ No deben quedar líneas de flujo sin conectar.
- ☐ Se deben trazar los símbolos de manera que se puedan leer de arriba hacia abajo y de izquierda a derecha.
- ☐ Todo texto escrito dentro de un símbolo deberá ser escrito claramente, evitando el uso de muchas palabras.

4.2 Pseudocódigo

Mezcla de lenguaje de programación y español (o inglés o cualquier otro idioma) que se emplea, dentro de la programación estructurada, para realizar el diseño de un programa. En esencial, el pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos.

Es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un problema determinado. El pseudocódigo utiliza palabras que indican el proceso a realizar.

Ventajas de utilizar un Pseudocódigo a un Diagrama de Flujo

- ☐ Ocupa menos espacio en una hoja de papel
- ☐ Permite representar en forma fácil operaciones repetitivas complejas
- ☐ Es muy fácil pasar de pseudocódigo a un programa en algún lenguaje de programación.
- ☐ Si se siguen las reglas se puede observar claramente los niveles que tiene cada operación.

CAPITULO V. ESTRUCTURAS ALGORITMICAS

5.1 Secuenciales

- Asignación
- Entrada
- Salida

5.2 Condicionales

- Simples
- Múltiples

5.3 Repetición fila condicional

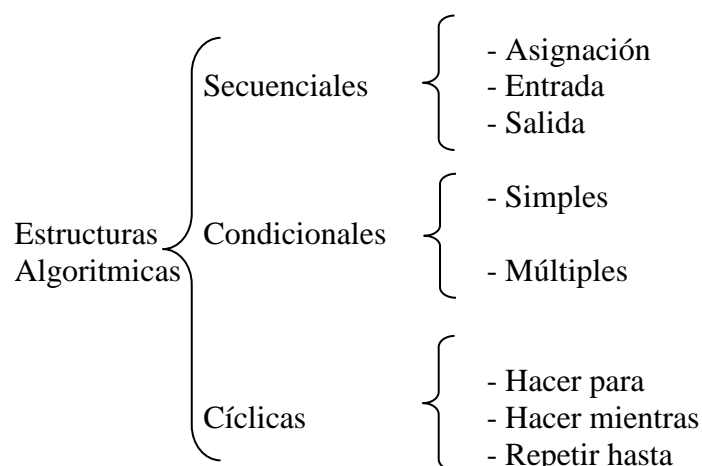
OBJETIVO:

El alumno:

- Conocerá las diferentes estructuras algorítmicas como componentes básicos de los programas y aplicara la combinación de ellas para el desarrollo de algoritmos más complejos.

ESTRUCTURAS ALGORITMICAS

Las estructuras de operación de programas son un grupo de formas de trabajo, que permiten, mediante la manipulación de variables, realizar ciertos procesos específicos que nos lleven a la solución de problemas. Estas estructuras se clasifican de acuerdo con su complejidad en:



5.1. Estructuras Secuenciales

La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso. Una estructura secuencial se representa de la siguiente forma:

```
Inicio
Accion1
Accion2
.
.
AccionN
Fin
```

- **Asignación**: La asignación consiste, en el paso de valores o resultados a una zona de la memoria. Dicha zona será reconocida con el nombre de la variable que recibe el valor. La asignación se puede clasificar de la siguiente forma:

- **Simple**: Consiste en pasar un valor constante a una variable ($a=15$)
- **Contador**: Consiste en usarla como un verificador del número de veces que se realiza un proceso ($a=a+1$)
- **Acumulador**: Consiste en usarla como un sumador en un proceso ($a=a+b$)
- **De trabajo**: Donde puede recibir el resultado de una operación matemática que involucre muchas variables ($a=c+b*2/4$).

- **Lectura:** La lectura consiste en recibir desde un dispositivo de entrada (p.ej. el teclado) un valor. Esta operación se representa en un pseudocódigo como sigue:

Leer a, b

Donde “a” y “b” son las variables que recibirán los valores

Escritura: Consiste en mandar por un dispositivo de salida (p.ej. monitor o impresora) un resultado o mensaje. Este proceso se representa en un pseudocódigo como sigue:

Escribe “El resultado es:”, R

Donde “El resultado es:” es un mensaje que se desea aparezca y R es una variable que contiene un valor.

Problemas Secuenciales

1) Suponga que un individuo desea invertir su capital en un banco y desea saber cuanto dinero ganara después de un mes si el banco paga a razón de 2% mensual.

Inicio

Leer cap_inv

gan = cap_inv * 0.02

Imprimir gan

Fin

2) Un vendedor recibe un sueldo base más un 10% extra por comisión de sus ventas, el vendedor desea saber cuanto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá en el mes tomando en cuenta su sueldo base y comisiones.

Inicio

Leer sb, v1, v2, v3

tot_vta = v1 + v2 + v3

com = tot_vta * 0.10

tpag = sb + com

Imprimir tpag, com

Fin

3) Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuanto deberá pagar finalmente por su compra.

Inicio

Leer tc

d = tc * 0.15

tp = tc - d

Imprimir tp

Fin

4) Un alumno desea saber cual será su calificación final en la materia de Algoritmos. Dicha calificación se compone de los siguientes porcentajes:

55% del promedio de sus tres calificaciones parciales.

30% de la calificación del examen final.

15% de la calificación de un trabajo final.

Inicio

Leer c1, c2, c3, ef, tf

$\text{prom} = (c1 + c2 + c3)/3$

$\text{ppar} = \text{prom} * 0.55$

$\text{pef} = \text{ef} * 0.30$

$\text{ptf} = \text{tf} * 0.15$

$\text{cf} = \text{ppar} + \text{pef} + \text{ptf}$

Imprimir cf

Fin

5) Un maestro desea saber que porcentaje de hombres y que porcentaje de mujeres hay en un grupo de estudiantes.

Inicio

Leer nh, nm

$\text{ta} = \text{nh} + \text{nm}$

$\text{ph} = \text{nh} * 100 / \text{ta}$

$\text{pm} = \text{nm} * 100 / \text{ta}$

Imprimir ph, pm

Fin

6) Realizar un algoritmo que calcule la edad de una persona.

Inicio

Leer fnac, fact

$\text{edad} = \text{fact} - \text{fnac}$

Imprimir edad

Fin.

5.2 Estructuras de Condicionales

Las estructuras condicionales comparan una variable contra otro(s) valor(es), para que en base al resultado de esta comparación, se siga un curso de acción dentro del programa. Cabe mencionar que la comparación se puede hacer contra otra variable o contra una constante, según se necesite. Existen dos tipos básicos, las simples y las múltiples.

- **Simples:** Las estructuras condicionales simples se les conoce como “Tomas de decisión”. Estas tomas de decisión tienen la siguiente forma:

```
Si <condición> entonces
    Acción(es)
Fin-si
```

- **Dobles:** Las estructuras condicionales dobles permiten elegir entre dos opciones o alternativas posibles en función del cumplimiento o no de una determinada condición. Se representa de la siguiente forma:

```
Si <condición> entonces
    Acción(es)
si no
    Acción(es)
Fin-si
```

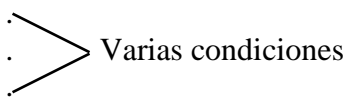
Donde:

Si	Indica el comando de comparación
Condición.....	Indica la condición a evaluar
entonces.....	Precede a las acciones a realizar cuando se cumple la condición
acción(es).....	Son las acciones a realizar cuando se cumple o no la condición
si no.....	Precede a las acciones a realizar cuando no se cumple la condición

Dependiendo de si la comparación es cierta o falsa, se pueden realizar una o mas acciones.

- **Múltiples:** Las estructuras de comparación múltiples, son tomas de decisión especializadas que permiten comparar una variable contra distintos posibles resultados, ejecutando para cada caso una serie de instrucciones específicas. La forma común es la siguiente:

```
Si <condición> entonces
    Acción(es)
si no
    Si <condición> entonces
        Acción(es)
    si no
        .
        .
        .
    Finsi
finsi
```



- **Forma General** [nota: lo veremos en los lenguajes de programación]

```
Casos Variable
    Op1: Acción(es)
    Op2: Acción(es)
    .
    .
    OpN: acción
Fin-casos
```

Problemas Selectivos Simples

1) Un hombre desea saber cuanto dinero se genera por concepto de intereses sobre la cantidad que tiene en inversión en el banco. El decidirá reinvertir los intereses siempre y cuando estos excedan a \$7000, y en ese caso desea saber cuanto dinero tendrá finalmente en su cuenta.

```
Inicio
    Leer p_int, cap
    int = cap * p_int
    si int > 7000 entonces
        capf = cap + int
    fin-si
    Imprimir capf
fin
```

2) Determinar si un alumno aprueba a reprueba un curso, sabiendo que aprobara si su promedio de tres calificaciones es mayor o igual a 70; reprueba en caso contrario.

```
Inicio
  Leer calif1, calif2, calif3
  prom = (calif1 + calif2 + calif3)/3
  Si prom >= 70 entonces
    Imprimir "alumno aprobado"
  si no
    Imprimir "alumno reprobado"
  Fin-si
Fin
```

3) En un almacén se hace un 20% de descuento a los clientes cuya compra supere los \$1000
¿ Cual será la cantidad que pagara una persona por su compra?

```
Inicio
  Leer compra
  Si compra > 1000 entonces
    desc = compra * 0.20
  si no
    desc = 0
  fin-si
  tot_pag = compra - desc
  imprimir tot_pag
fin.
```

4) Un obrero necesita calcular su salario semanal, el cual se obtiene de la sig. manera:

Si trabaja 40 horas o menos se le paga \$16 por hora

Si trabaja mas de 40 horas se le paga \$16 por cada una de las primeras 40 horas y \$20 por cada hora extra.

```
Inicio
  Leer ht
  Si ht > 40 entonces
    he = ht - 40
    ss = he * 20 + 40 * 16
  si no
    ss = ht * 16
  Fin-si
  Imprimir ss
Fin
```

5) Un hombre desea saber cuanto dinero se genera por concepto de intereses sobre la cantidad que tiene en inversión en el banco. El decidirá reinvertir los intereses siempre y cuando estos excedan a \$7000, y en ese caso desea saber cuanto dinero tendrá finalmente en su cuenta.

```
Inicio
  Leer p_int, cap
  int = cap * p_int
  si int > 7000 entonces
    capf = cap + int
  fin-si
  Imprimir capf
fin
```

6) Que lea dos números y los imprima en forma ascendente

```
Inicio
  Leer num1, num2
  Si num1 < num2 entonces
    Imprimir num1, num2
  si no
    Imprimir num2, num1
  fin-si
fin
```

7) Una persona enferma, que pesa 70 kg, se encuentra en reposo y desea saber cuantas calorías consume su cuerpo durante todo el tiempo que realice una misma actividad. Las actividades que tiene permitido realizar son únicamente dormir o estar sentado en reposo. Los datos que tiene son que estando dormido consume 1.08 calorías por minuto y estando sentado en reposo consume 1.66 calorías por minuto.

```
Inicio
  Leer act$, tiemp
  Si act$ = "dormido" entonces
    cg = 1.08 * tiemp
  si no
    cg = 1.66 * tiemp
  fin-si
  Imprimir cg
Fin
```


8) Hacer un algoritmo que imprima el nombre de un artículo, clave, precio original y su precio con descuento. El descuento lo hace en base a la clave, si la clave es 01 el descuento es del 10% y si la clave es 02 el descuento es del 20% (solo existen dos claves).

```
Inicio
  Leer nomb, cve, prec_orig
  Si cve = 01 entonces
    prec_desc = prec_orig - prec_orig * 0.10
  si no
    prec_desc = prec_orig - prec_orig * 0.20
  fin-si
  Imprimir nomb, cve, prec_orig, prec_desc
fin
```

9) Hacer un algoritmo que calcule el total a pagar por la compra de camisas. Si se compran tres camisas o más se aplica un descuento del 20% sobre el total de la compra y si son menos de tres camisas un descuento del 10%

```
Inicio
  Leer num_camisas, prec
  tot_comp = num_camisas * prec
  Si num_camisas >= 3 entonces
    tot_pag = tot_comp - tot_comp * 0.20
  si no
    tot_pag = tot_comp - tot_comp * 0.10
  fin-si
  Imprimir tot_pag
fin
```

10) Una empresa quiere hacer una compra de varias piezas de la misma clase a una fábrica de refacciones. La empresa, dependiendo del monto total de la compra, decidirá que hacer para pagar al fabricante.

Si el monto total de la compra excede de \$500 000 la empresa tendrá la capacidad de invertir de su propio dinero un 55% del monto de la compra, pedir prestado al banco un 30% y el resto lo pagará solicitando un crédito al fabricante.

Si el monto total de la compra no excede de \$500 000 la empresa tendrá capacidad de invertir de su propio dinero un 70% y el restante 30% lo pagará solicitando crédito al fabricante.

El fabricante cobra por concepto de intereses un 20% sobre la cantidad que se le pague a crédito.

```
Inicio
  Leer costopza, numpza
  totcomp = costopza * numpza
  Si totcomp > 500 000 entonces
    cantinv = totcomp * 0.55
    préstamo = totcomp * 0.30
    crédito = totcomp * 0.15
  si no
```

```
        cantinv = totcomp * 0.70
        crédito = totcomp * 0.30
        préstamo = 0
    fin-si
    int = crédito * 0.20
    Imprimir cantinv, préstamo, crédito, int
Fin
```

Problemas Selectivos Compuestos

1) Leer 2 números; si son iguales que los multiplique, si el primero es mayor que el segundo que los reste y si no que los sume.

```
Inicio
    Leer num1, num2
    si num1 = num2 entonces
        resul = num1 * num2
    si no
        si num1 > num2 entonces
            resul = num1 - num2
        si no
            resul = num1 + num2
    fin-si
fin-si
fin
```

2) Leer tres números diferentes e imprimir el numero mayor de los tres.

```
Inicio
    Leer num1, num2, num3
    Si (num1 > num2) and (num1 > num3) entonces
        mayor = num1
    si no
        Si (num2 > num1) and (num2 > num3) entonces
            mayor = num2
        si no
            mayor = num3
    fin-si
fin-si
Imprimir mayor
fin
```

3) Determinar la cantidad de dinero que recibirá un trabajador por concepto de las horas extras trabajadas en una empresa, sabiendo que cuando las horas de trabajo exceden de 40, el resto se consideran horas extras y que estas se pagan al doble de una hora normal cuando

no exceden de 8; si las horas extras exceden de 8 se pagan las primeras 8 al doble de lo que se pagan las horas normales y el resto al triple.

```

Inicio
  Leer ht, pph
  Si ht <= 40 entonces
    tp = ht * pph
  si no
    he = ht - 40
    Si he <= 8 entonces
      pe = he * pph * 2
    si no
      pd = 8 * pph * 2
      pt = (he - 8) * pph * 3
      pe = pd + pt
    fin-si
  tp = 40 * pph + pe
fin-si
Imprimir tp
fin
  
```

4) Calcular la utilidad que un trabajador recibe en el reparto anual de utilidades si este se le asigna como un porcentaje de su salario mensual que depende de su antigüedad en la empresa de acuerdo con la sig. tabla:

Tiempo	Utilidad
Menos de 1 año	5 % del salario
1 año o mas y menos de 2 años	7% del salario
2 años o mas y menos de 5 años	10% del salario
5 años o mas y menos de 10 años	15% del salario
10 años o mas	20% del salario

```

Inicio
  Leer sm, antig
  Si antig < 1 entonces
    util = sm * 0.05
  si no
    Si (antig >= 1) and (antig < 2) entonces
      util = sm * 0.07
    si no
      Si (antig >= 2) and (antig < 5) entonces
        util = sm * 0.10
      si no
        Si (antig >= 5) and (antig < 10) entonces
          util = sm * 0.15
        si no
          util = sm * 0.20
        fin-si
      fin-si
    fin-si
  fin-si
  
```

```
        fin-si
    fin-si
    Imprimir util
fin
```

5) En una tienda de descuento se efectúa una promoción en la cual se hace un descuento sobre el valor de la compra total según el color de la bolita que el cliente saque al pagar en caja. Si la bolita es de color blanco no se le hará descuento alguno, si es verde se le hará un 10% de descuento, si es amarilla un 25%, si es azul un 50% y si es roja un 100%. Determinar la cantidad final que el cliente deberá pagar por su compra. se sabe que solo hay bolitas de los colores mencionados.

```
Inicio
    leer tc, b$
    si b$ = 'blanca' entonces
        d=0
    si no
        si b$ = 'verde' entonces
            d=tc*0.10
        si no
            si b$ = 'amarilla' entonces
                d=tc*0.25
            si no
                si b$ = 'azul' entonces
                    d=tc*0.50
                si no
                    d=tc
            fin-si
        fin-si
    fin-si
    Imprimir d
fin
```

6) El IMSS requiere clasificar a las personas que se jubilaran en el año de 1997. Existen tres tipos de jubilaciones: por edad, por antigüedad joven y por antigüedad adulta. Las personas adscritas a la jubilación por edad deben tener 60 años o mas y una antigüedad en su empleo de menos de 25 años. Las personas adscritas a la jubilación por antigüedad joven deben tener menos de 60 años y una antigüedad en su empleo de 25 años o mas.

Las personas adscritas a la jubilación por antigüedad adulta deben tener 60 años o mas y una antigüedad en su empleo de 25 años o mas.

Determinar en que tipo de jubilación, quedara adscrita una persona.

```
Inicio
    leer edad,ant
    si edad >= 60 and ant < 25 entonces
        imprimir "la jubilación es por edad"
    si no
        si edad >= 60 and ant > 25 entonces
```

```
        imprimir "la jubilación es por edad adulta"
    si no
        si edad < 60 and ant > 25 entonces
            imprimir "la jubilación es por antigüedad joven"
        si no
            imprimir "no tiene por que jubilarse"
        fin-si
    fin-si
fin-si
fin
```

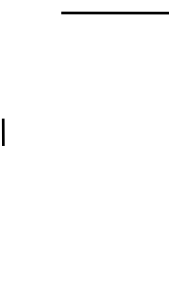
5.3. Estructuras Cíclicas

Se llaman problemas repetitivos o cíclicos a aquellos en cuya solución es necesario utilizar un mismo conjunto de acciones que se puedan ejecutar una cantidad específica de veces. Esta cantidad puede ser fija (previamente determinada por el programador) o puede ser variable (estar en función de algún dato dentro del programa). Los ciclos se clasifican en:

- **Ciclos con un Numero Determinado de Iteraciones (Hacer-Para)**

Son aquellos en que el número de iteraciones se conoce antes de ejecutarse el ciclo. La forma de esta estructura es la siguiente:

```
DESDE V.C = L.I HASTA L.S
    Accion1
    Accion2
    .
    .
    .
    AccionN
Fin-para
```



Donde:

V.C Variable de control del ciclo
L.I Limite inferior
L.S Límite superior

En este ciclo la variable de control toma el valor inicial del ciclo y el ciclo se repite hasta que la variable de control llegue al límite superior.

Problemas

1) Calcular el promedio de un alumno que tiene 7 calificaciones en la materia de Diseño

Estructurado de Algoritmos

```
Inicio
  Sum=0
  Leer Nom
  DESDE c = 1 a 7
    Leer calif
    Sum = sum + calif
  Fin-desde
  prom = sum /7
  Imprimir prom
Fin.
```

2) Leer 10 números y obtener su cubo y su cuarta.

```
Inicio
  DESDE n = 1 a 10
    Leer num
    cubo = num * num * num
    cuarta = cubo * num
    Imprimir cubo, cuarta
  Fin-desde
Fin.
```

3) Leer 10 números e imprimir solamente los números positivos

```
Inicio
  DESDE n = 1 a 10
    Leer num
    Si num > 0 entonces
      Imprimir num
    fin-si
  Fin-DESDE
Fin.
```

4) Leer 20 números e imprimir cuantos son positivos, cuantos negativos y cuantos neutros.

```
Inicio
  cn = 0
  cp = 0
  cneg = 0
  DESDE x = 1 a 20
```

```
Leer num
Sin num = 0 entonces
    cn = cn + 1
si no
    Si num > 0 entonces
        cp = cp + 1
    si no
        cneg = cneg + 1
Fin-si
Fin-si
Fin-DESDE
Imprimir cn, cp, cneg
Fin.
```

5) Leer 15 números negativos y convertirlos a positivos e imprimir dichos números.

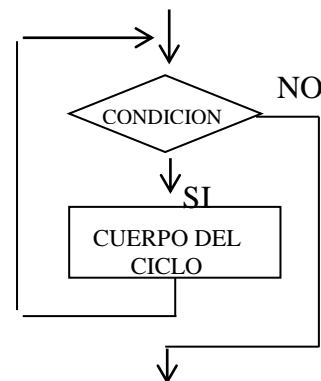
```
Inicio
Hacer para x = 1 a 15
    Leer num
    pos = num * -1
    Imprimir num, pos
Fin-para
Fin.
```

- **Ciclos con un Numero Indeterminado de Iteraciones (Hacer-Mientras, Repetir-Hasta)**

Son aquellos en que el numero de iteraciones no se conoce con exactitud, ya que esta dado en función de un dato dentro del programa.

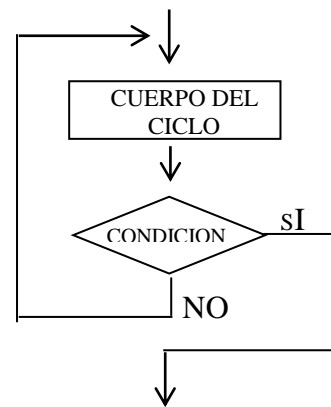
- Hacer-Mientras: Esta es una estructura que repetira un proceso durante “N” veces, donde “N” puede ser fijo o variable. Para esto, la instrucción se vale de una condición que es la que debe cumplirse para que se siga ejecutando. Cuando la condición ya no se cumple, entonces ya no se ejecuta el proceso. La forma de esta estructura es la siguiente:

```
Hacer mientras <condición>
    Accion1
    Accion2
    .
    .
    AccionN
Fin-mientras
```



- ***Repetir-Hasta:*** Esta es una estructura similar en algunas características, a la anterior. Repite un proceso una cantidad de veces, pero a diferencia del Hacer-Mientras, el Repetir-Hasta lo hace hasta que la condición se cumple y no mientras, como en el Hacer-Mientras. Por otra parte, esta estructura permite realizar el proceso cuando menos una vez, ya que la condición se evalúa al final del proceso, mientras que en el Hacer-Mientras puede ser que nunca llegue a entrar si la condición no se cumple desde un principio. La forma de esta estructura es la siguiente:

```
Repetir
    Accion1
    Accion2
    .
    .
    AccionN
Hasta <condición>
```



CAPITULO VI. ARREGLOS

- 6.1 Vectores
- 6.2 Matrices
- 6.3 Manejo de cadenas de caracteres

OBJETIVO:

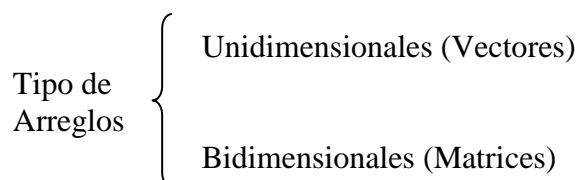
El alumno:

- Será capaz de utilizar los datos de tipo arreglo para plantear la solución de problemas que requieran de esta estructura.

Arreglo: Un *Arreglo* es una estructura de datos que almacena bajo el mismo nombre (variable) a una colección de datos del mismo tipo.

Los arreglos se caracterizan por:

- Almacenan los elementos en posiciones contiguas de memoria
- Tienen un mismo nombre de variable que representa a todos los elementos. Para hacer referencia a esos elementos es necesario utilizar un índice que especifica el lugar que ocupa cada elemento dentro del archivo.



6.1. Vectores

Es un arreglo de “N” elementos organizados en una dimensión donde “N” recibe el nombre de longitud o tamaño del vector. Para hacer referencia a un elemento del vector se usa el nombre del mismo, seguido del índice (entre corchetes), el cual indica una posición en particular del vector. Por ejemplo:

Vec[x]

Donde:

Vec..... Nombre del arreglo

x..... Numero de datos que constituyen el arreglo

Representación gráfica de un vector

Vec[1]	7
Vec[2]	8
Vec[3]	9
Vec[4]	1
	0

Llenado de un Vector

- Hacer para $I = 1$ a 10
 Leer $\text{vec}[I]$
Fin-para
- Hacer mientras $I \leq 10$
 Leer $\text{vec}[I]$
Fin-mientras
- $I=1$
Repetir
 Leer $\text{vec}[I]$
 $I = I + 1$
Hasta-que $I > 10$

6.2 Matriz

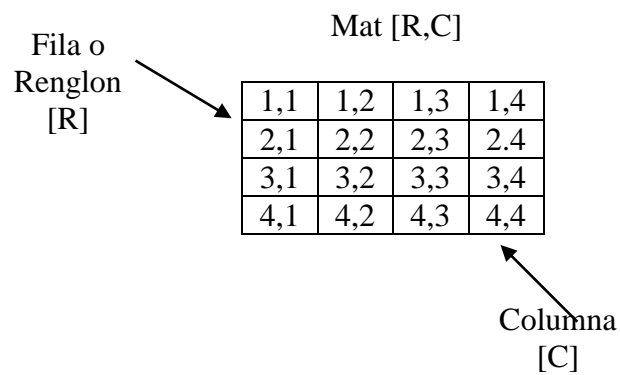
Es un arreglo de $M * N$ elementos organizados en dos dimensiones donde “M” es el numero de filas o renglones y “N” el numero de columnas.

Para representar una matriz se necesita un nombre de matriz se necesita un nombre de matriz acompañado de dos índices.

$\text{Mat} [R,C]$

Donde R indica el renglón y C indica la columna, donde se encuentra almacenado el dato.

Representación gráfica de una matriz



Llenado de una matriz

- ***Por renglones***
Hacer para R = 1 a 5
 Hacer para C = 1 a 5
 Leer Mat [R,C]
 Fin-para
Fin-para
- ***Por columnas***
Hacer para C = 1 a 5
 Hacer para R = 1 a 5
 Leer Mat [R,C]
 Fin-para
Fin-para

Nota: Para hacer el llenado de una matriz se deben de usar dos variables para los índices y se utilizan 2 ciclos uno para los renglones y otro para las columnas; a estos ciclos se les llama ciclos anidados (un ciclo dentro de otro ciclo).

CAPITULO VII. PROGRAMACIÓN ESTRUCTURADA

7.1 Introducción

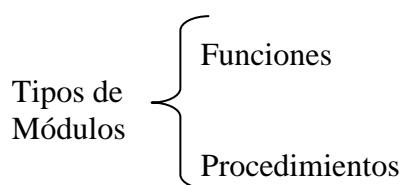
7.2 Tipos módulos

6.1 Introducción

Un problema complejo se puede dividir en pequeños subproblemas más sencillos. Estos subproblemas se conocen como “*Módulos*” y su complementación en un lenguaje se llama subprograma (procedimientos y funciones).

Un subprograma realiza las mismas acciones que un programa, sin embargo, un subprograma lo utiliza solamente un programa para un propósito específico.

Un subprograma recibe datos de un programa y le devuelve resultados (el programa “llama” o “invoca” al subprograma, este ejecuta una tarea específica y devuelve el “control” al programa que lo llamo).



PARTES DE UN PROGRAMA

INICIO
PARTE DECLARATIVA
CUERPO PROGRAMA
FIN

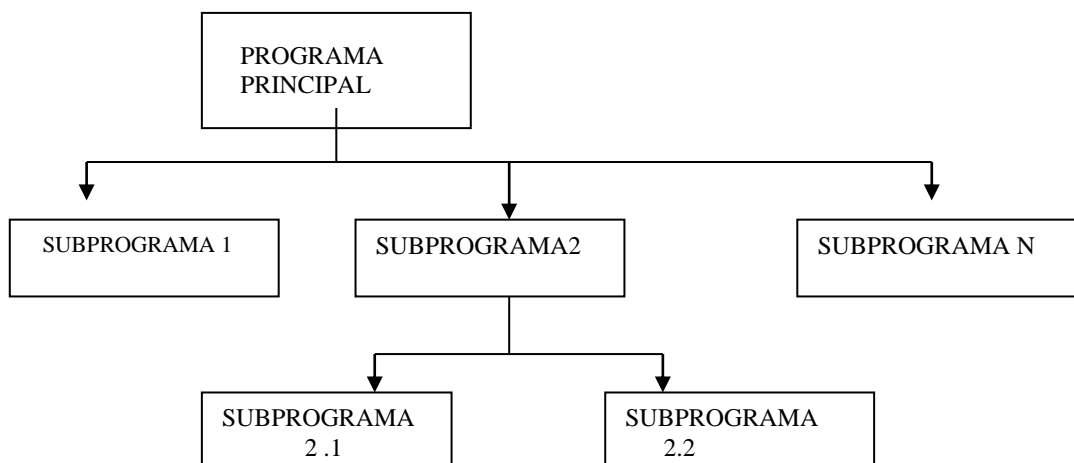
En la parte declarativa, iría la declaración de las constantes y variables.

En la parte del cuerpo del programa estarán los subprogramas (si los tiene) o la acción a realizar (si es demasiado sencillo).

SUBPROGRAMAS

El método para solucionar un problema muy complejo es dividirlo en subproblemas (problemas más sencillos) y a su vez estos en otros más simples.

Esta técnica se la conoce como “divide y vencerás”. El método de diseñar la solución de un problema principal obteniendo sus soluciones de sus subproblemas se conoce como diseño descendente (top-down), es decir, de lo general a lo específico.



6.2 Tipos de módulos

Los subprogramas pueden ser procedimientos o funciones

PROCEDIMIENTOS	FUNCIONES
PROCEDIMIENTO Nombre (Lista Parametros*) INICIO --Parte declarativa (variables locales) --Cuerpo Subprograma _____ _____ FIN	FUNCION Nombre (Lista Parametros*): TipoDevuelto INICIO --Parte declarativa (variables locales) --Cuerpo Subprograma _____ _____ FIN
Devuelve de 0 a N valores en sus prarmetros referenciados	Devuelve un valor y solo uno
Parametros por valor y/o referencia	Solo parametros por valor
Usarse en una línea única, una sentencia	Puede usarse en en cualquier tipo de expresión porque su valor se asocia al nombre de la función
Al menos se le pasa un parámetro en su lista de parámetros	No tiene por qué llevar parámetros

Fundamentos de la Programación

*Lista de Parámetros

-donde hemos puesto lista de parámetros en el cuadro nos referimos a:
(argumento1: tipo1, argumento2: tipo2, argumentoN: tipoN)

-Dos formas de hacer el paso de parámetros:

-*por valor*: no devuelve información al programa que llama, es decir los valores con los que entra van a ser los mismos cuando salga

-*por referencia*: las modificaciones que se realicen el subprograma a los valores de los argumentos si van a llegar al programa o subprograma que le llama (es decir, puede que salgan con un valor distinto a como habían entrado). Se pone VAR

BIBLIOGRAFÍA

JEAN Paul Tremblay, B. Bunt Richard; "Introducción a la ciencias de las computadoras (enfoque algoritmico)" Mc Graw Hill

JOYANES Aguilar Luis; "Metodología de la programación" Mc Graw Hill

JOYANES Aguilar Luis; "Problemas de metodología de la programación" Mc Graw Hill

CORREA Uribe Guillermo; "Desarrollo de algoritmos y sus aplicaciones en Basic, Pascal y C (3ª. Edición)" Mc Graw Hill

Levine Guillermo; "Introducción a la computación y a la programación estructurada" Mc Graw Hill

JOYANES Aguilar Luis; "Fundamentos de programación, algoritmos y estructura de datos" Mc Graw Hill

JOYANES Aguilar Luis, Luis Rodríguez Baena y Matilde Fernández Azuela; "Fundamentos de programación, libro de problemas" Mc Graw Hill

Bores Rosario, Rosales Roman; "Computación. Metodología, lógica computacional y programación" Mc Graw Hill

LOZANO Letvin; "Diagramación y programación estructurada y libre" Mc Graw Hill

LOPEZ Roman Leobardo; "Programación estructurada (enfoque algoritmico)" Computec