# ***Resumen CADP***

**Informática**: es la *ciencia* que estudia el *análisis y resolución de problemas utilizando computadoras.* Se relaciona con la matemática y la ingeniería. Se pueden usar las herramientas informáticas en aplicaciones de diversas áreas (educación, biología, comercio, etc.).

**Computadora**: máquina digital y sincrónica, con cierta *capacidad de cálculo numérico y lógico* controlado por un programa almacenado y con probabilidad de comunicación con el mundo exterior. Ayuda al hombre a realizar *tareas repetitivas en menor tiempo y con mayor exactitud*. **No** *razona ni crea soluciones*, sino que ejecuta una serie de órdenes que le proporciona el ser humano.

El **objetivo de la informática** es resolver problemas del mundo real utilizando una computadora (mediante un *software*).

**Programa** = algoritmo + datos

**Algoritmo**: *especificación rigurosa* de la secuencia de pasos (instrucciones) a realizar sobre un *autómata* para alcanzar un resultado deseado en un *tiempo finito.*

>Especificación rigurosa: el algoritmo debe ser claro y unívoco.

>Autómata: computadora. Hay que escribir el algoritmo en un lenguaje entendible por ella.

>Tiempo finito: el algoritmo debe tener un principio y un fin.

**Datos**: son los valores de información de los que se necesita disponer y en ocasiones transformar para ejecutar la función del programa.

Son representaciones de un objeto del mundo real mediante el cual se pueden modelizar aspectos del problema que se quiere resolver. Puede ser constante o variable.

un tipo de dato es una clase de objetos ligados a un conjunto de operaciones para crearlos y manipularlos.

Los diferentes tipos de datos deben especificarse y a esta especificación dentro de un programa se la conoce como declaración.

**Estructura de datos:** una estructura de datos es un conjunto de variables (no necesariamente del mismo tipo) relacionadas entre sí de diversas formas. Permite al programador definir un tipo al que se asocian diferentes datos que tienen valores lógicamente relacionados y asociados bajo un nombre único.

Características 🡪 puede ser:

>*Homogénea*: Los elementos que la componen son del mismo tipo

>*Heterogénea*: Los elementos que la componen pueden ser de distinto tipo

>*Secuencial*: Para acceder a un elemento particular se debe respetar un orden predeterminado pasando por todos los elementos que le preceden.

>*Directo*: Se puede acceder a un elemento particular, directamente, sin necesidad de pasar por los anteriores a él, por ejemplo, referenciando una posición.

>*Dinámica*: El tamaño de la estructura puede variar durante la ejecución del programa

>*Estática*: El tamaño de la estructura no varía durante la ejecución del programa

>*Lineal*: Está formada por ninguno, uno o varios elementos que guardan una relación de adyacencia ordenada donde a cada elemento le sigue uno y le precede uno, solamente.

>*No* *lineal*: Para un elemento dado pueden existir 0, 1 o más elementos que le suceden y 0, 1 o más elementos que le preceden.

>**Simple**: aquellos que toman un único valor, en un momento determinado, de todos los permitidos para ese tipo.

\*Definido por el lenguaje: son provistos por el lenguaje y tanto la representación como sus operaciones y valores son reservadas al mismo. Son:

*-Dato* ***numérico*** *(integer, real)*

*-Dato* ***lógico*** *(boolean)*

*-Dato* ***carácter*** *(char)*

\*Definido por el programador: permiten definir nuevos tipos de datos a partir de los tipos simples. Son:

-***Subrango***: selección de integer o char para limitar los datos que se pueden usar. Por ejemplo, un subrango de enteros del 1..10 para calificar notas de alumnos.

-***Puntero***: Es un tipo de variable usada para almacenar una dirección en *memoria*. En esa dirección de memoria se encuentra el valor que puede ser de cualquiera de los tipos vistos (incluso otro puntero). Cada variable puntero solo puede apuntar a un único tipo de dato. Se crea con la palabra clave *“new()”,* se elimina con un *“dispose()”* y puede cortarse el enlace a memoria (sin liberar memoria como con dispose) haciéndolo apuntar a *“nil”.*

* *Ventajas de los tipos de datos definidos por el programador:*

- *Flexibilidad*: en el caso de ser necesario modificar la forma en que se representa el dato, sólo se debe modificar una declaración en lugar de un conjunto de declaraciones de variables.

- *Documentación*: se pueden usar como identificador de los tipos, nombres autoexplicativos, facilitando de esta manera el entendimiento y lectura del programa.

- *Seguridad*: se reducen los errores por uso de operaciones inadecuadas del dato a manejar, y se pueden obtener programas más confiables.

>**Compuesto**: pueden tomar varios valores a la vez que guardan alguna relación lógica entre ellos, bajo un único nombre.

\*Definido por el lenguaje:

-*Dato* ***cadena*** *(string)*

\*Definido por el programador:

-*Dato* ***registro*** *(record):* Es uno de los tipos de datos estructurados, que permiten agrupar diferentes clases de datos en una estructura única bajo un sólo nombre. Es de tipo heterogéneo y estático.

*-Dato* ***arreglo*** *(array):* es una estructura de datos compuesta que permite acceder a cada componente por una variable índice, que da la posición del componente dentro de la estructura de datos. Un vector (forma más simple de arreglo) es una colección de elementos que se guardan consecutivamente en la memoria y se pueden referenciar a través de un índice. Es de tipo homogéneo, estático y de acceso directo (indexado con un índice de tipo ordinal). Permite operaciones de buscar (distintas maneras, más y menos eficientes), agregar, insertar, ordenar, y eliminar elementos en él. Tiene:

* + - *Dimensión Física (dimF):* Se especifica en el momento de la declaración y determina su ocupación máxima de memoria. La cantidad de memoria total reservada no variará durante la ejecución del programa.
    - *Dimensión Lógica (dimL):* Se determina cuando se cargan contenidos a los elementos del arreglo. Indica la cantidad de posiciones de memoria ocupadas con contenido real. Nunca puede superar la dimensión física.

-*Dato* ***Lista****:* Colección de nodos, donde cada nodo de la lista se representa con un puntero, que apunta a un dato (elemento de la lista) y a una dirección (donde se ubica el siguiente elemento de la lista). Toda Lista se crea con nodo apuntando a nil; y ese último elemento de la Lista siempre debe seguir apuntando allí, o la estructura se perderá. Es de tipo homogénea, secuencial, dinámica y lineal. Al igual que en los arreglos, se puede agregar, insertar, ordenar, eliminar y buscar; pero al ser secuencial el acceso, es más complejo hacerlo.

**Tipos de lenguajes:**

>Algunos lenguajes exigen que se especifique a qué tipo pertenece cada una de las variables. Verifican que el tipo de los datos asignados a esa variable se correspondan con su definición. Esta clase de lenguajes se denomina **fuertemente tipados (strongly typed).**

>Otra clase de lenguajes, que verifica el tipo de las variables según su nombre, se denomina **auto tipados (self typed).**

>Existe una tercera clase de lenguajes que permiten que una variable tome valores de distinto tipo durante la ejecución de un programa. Esta se denomina **dinámicamente tipados (dinamically typed).**

**Estructuras de control**: conjunto de instrucciones que permiten especificar el control del algoritmo que se quiere implementar. Como mínimo en un lenguaje de programación se requieren: secuencia, decisión e iteración.

>Secuencia: sucesión de operaciones en la que el orden de ejecución coincide con el orden físico de aparición de las instrucciones. Ej:

Read(num); 🡨 se ejecuta 1ro

Write(‘Usted ingresó: ‘,num); 🡨 se ejecuta 2do

>Decisión: se eligen acciones en función de los datos del programa. Ej:

If (condición verdadera) then

…

Else

…

>Iteración: ejecutar un bloque de condiciones sin saber cuándo va a terminar exactamente, sino que termina en base a pre o post condiciones que se deben cumplir.

*-Pre condicional*: evalúan la condición y si es verdadera se ejecuta el bloque de acciones. Dicho bloque se pueda ejecutar 0, 1 o más veces. Se usa *while*.

* Con la estructura while es posible realizar un *corte de control* en un programa. Esto se realiza en situaciones donde se busca operar con todos los elementos que tengan una característica en común a la vez, informar o almacenar, y luego seguir con el resto, sin mezclarlos.

*-Post condicional*: Ejecutan las acciones, luego evalúan la condición y solo ejecutan las acciones mientras la condición es falsa. Dicho bloque se pueda ejecutar 1 o más veces. Se usa *repeat…until.*

>Repetición: Es una extensión natural de la secuencia. Consiste en repetir N veces un bloque de acciones. Este número de veces que se deben ejecutar las acciones es fijo y conocido de antemano. Se usa *for i:=* X *to* Y.

>Selección: Permite realizar distintas acciones dependiendo del valor de una variable de tipo ordinal. Ej:

*case* (variable) *of*

condicion1: acción1;

condición 2:

begin

acción2;

acción 3;

end;

…..

condición n: acción x;

end;

**Modularización:** división del programa en partes funcionalmente independientes, que encapsulen operaciones y datos.

Ventajas:

-*Mayor productividad*: al dividir un sistema de software en módulos funcionalmente independientes, un equipo de desarrollo puede trabajar simultáneamente en varios de estos, incrementando la productividad.

-*Reusabilidad*: posibilidad de utilizar repetidamente el producto de software desarrollado. Naturalmente la descomposición funcional que ofrece la modularización favorece el reuso.

-*Facilidad* *de crecimiento*: Los sistemas de software reales crecen en requerimientos del usuario. La modularización permite disminuir los riesgos y costos de incorporar nuevas prestaciones a un sistema en funcionamiento.

-*Legibilidad*: Un efecto de la modularización es una mayor claridad para leer y comprender el código fuente. El ser humano maneja y comprende con mayor facilidad un número limitado de instrucciones directamente relacionadas.

-*Facilidad de mantenimiento*: La experiencia demuestra que el costo de mantenimiento en un código que no ha sido apropiadamente modularizado es muy alto. En cambio, este puede reducirse si el código está bien modularizado dado que el tiempo requerido para entender el programa y luego modificarlo es notoriamente menor. Otro problema común cuando se realiza el mantenimiento de un programa es tener en cuenta los side effects (efectos colaterales), los que aparecen al cambiar el código para satisfacer cierta necesidad y producen efectos no deseados en otras partes del programa.

\*Módulo: Tarea específica bien definida que se comunica con otros módulos entre sí adecuadamente y cooperan para conseguir un objetivo común.

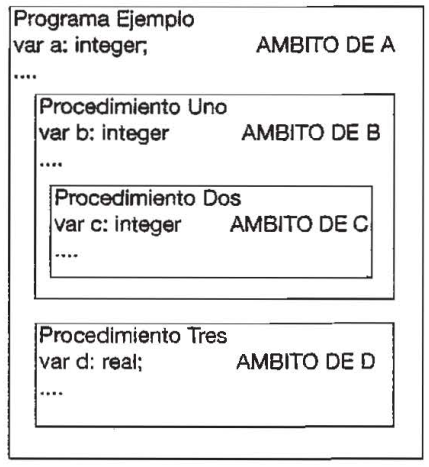
>Procedimiento (procedure): conjunto de instrucciones que realizan una tarea específica y retorna 0, 1 o más valores.

>Función (function): conjunto de instrucciones que realizan una tarea específica y retorna un único valor de tipo simple.

**Tipos de variables**:

*Variable local*: es aquella que está declarada y definida dentro de un programa o módulo, en el sentido que está dentro de ese módulo y es distinta a cualquier variable que tenga el mismo nombre y que estuviera declarada en otro lugar del programa.

*Variable global*: es aquella que está declarada en el programa y que puede ser utilizada por cualquier módulo de este (incluido el programa principal).



**Alcance de las variables** (orden de búsqueda de la computadora):

>Si es una variable utilizada en un módulo:

-Se busca si es variable local

-Se busca si es un parámetro

-Se busca si es variable global al programa

>Si es una variable utilizada en un programa:

-Se busca si es variable local al programa

-Se busca si es variable global al programa

**Desventajas de las variables globales:**

-Demasiados identificadores

-No se especifica la comunicación entre los módulos

-Conflictos de nombres de identificadores utilizados por diferentes programadores.

-Posibilidad de perder integridad de los datos, al modificar involuntariamente en un módulo datos de alguna variable que luego deberá utilizar otro módulo.

La **SOLUCIÓN** a estos problemas ocasionados por el uso de variables globales es una combinación de *ocultamiento de datos* (Data Hiding ) y uso de *parámetros*.

-El **ocultamiento de datos** significa que los datos exclusivos de un módulo *NO* deben ser "visibles" o utilizables por los demás módulos.

-Se denomina **parámetros** a la serie de datos con los que se comunican los módulos. Son variables que tienen como característica principal transferir información entre los módulos.

Se pueden usar *parámetros* *por* *valor* o *por* *referencia:*

*>Parámetro por valor* (IN): se recibe una *copia* de una variable enviada desde otro módulo (o desde programa principal). Las modificaciones son internas al módulo, NO se modifican en el resto del programa.

*>Parámetro por referencia* (OUT / INOUT): se recibe la *dirección* de la variable. Se opera con ella dentro del módulo y las modificaciones que se produzcan se reflejan en el resto del programa que utilice dicha variable.

Cuando se desarrollan los algoritmos hay 2 conceptos importantes que se deben tener en cuenta:

>**Corrección**: Un programa es correcto si se realiza de acuerdo a sus especificaciones. Es decir que deben cumplirse todas las pre y post condiciones.

*Pre condición*: es la información que se conoce como verdadera antes de iniciar el programa (o módulo).

Post condición: es la información que debería ser verdadera al concluir el programa (o módulo), si se cumplen adecuadamente los pasos especificados.

Los *errores* pueden provenir de *2 fuentes*:

-*El diseño del programa no es el adecuado*.

-*El programa no está escrito correctamente*.

Tenemos 3 tipos de errores:

-*Errores sintácticos*: se detectan en la compilación.

-*Errores lógicos*: generalmente se detectan en la ejecución.

-*Errores del sistema*: son muy raros los casos en los que ocurren.

Hay *4 técnicas* que se emplean para la corrección de programas (en cualquier caso pueden usarse 1, 2, 3 o las 4 técnicas). Son:

-*Testing*: el propósito es proveer evidencias convincentes de que el programa hace el trabajo esperado. Para ello se realiza un plan de pruebas, en el cual:

* Se deciden qué aspectos deben ser testeados.
* Se determina cuál es el resultado esperado en cada caso. Es recomendable hacerlo antes de escribir el programa, ya que es mejor diseñar los casos de prueba en base a lo que debe hacer el programa y no en base a cómo se escribió (si no podría llevar a tener errores no descubiertos).
* Poner atención en los casos límite.
* Se analizan todos los casos y si hay errores se corrigen.

-*Debugging*: es el proceso de descubrir la causa del error. Se agregan pruebas (sentencias del estilo write) adicionales para monitorear el comportamiento del programa más cercanamente y conocer la naturaleza del error.

-*Walkthrough*: es recorrer el programa frente a una audiencia. Como otra persona que es ajena al programa no comparte los preconceptos, es un buen medio para detectar errores. Cuando no se puede encontrar un error el programador trata de demostrar que no existe, pero mientras lo hace, puede detectar el error o que alguien más lo encuentre.

-*Verificación*: verificar un programa significa controlar que se cumplan las pre y post condiciones del mismo.

Una vez que se decide que un algoritmo es correcto hay que determinar su *eficiencia*.

>**Eficiencia**: Se define a la eficiencia como una métrica de calidad de los algoritmos, asociada con una utilización óptima de los recursos del sistema de cómputo donde se ejecutará el algoritmo. El análisis de la eficiencia de un algoritmo estudia el *tiempo* que tarda en ejecutarse y la *memoria* que requiere. Se relaciona con:

-Datos de entrada

-Calidad del código generado por el compilador

-Naturaleza y rapidez en la ejecución de las instrucciones de la máquina

-El tiempo del algoritmo base

Entonces, para saber lo eficiente que es un algoritmo hay que medir el *tiempo de ejecución* y la *memoria* que usa. ¿Cómo se mide esto?

>*Memoria*: se suma el peso de todas las variables (en bytes). Como la memoria se divide en memoria estática y dinámica, se hacen dos cálculos separados para cada una.

>*Tiempo de ejecución*: puede definirse como una función de entrada:

- Existen algoritmos donde el tiempo de ejecución no depende de las características de los datos de entrada, sino de la cantidad de datos de entrada o su tamaño.

- Existen otros algoritmos donde el tiempo de ejecución es una función de la entrada “específica”. En estos casos se habla del tiempo de ejecución del “peor” caso. En estas situaciones, se obtiene una cota superior del tiempo de ejecución para cualquier entrada.

El tiempo de ejecución puede calcularse con 2 tipos de análisis:

-*Análisis empírico*: Para realizar un análisis empírico, es necesario correr el programa y medir el tiempo consumido.

* *Pros*: fácil de realizar.
* *Contras*: es dependiente de la máquina donde se ejecuta; en otra computadora daría valores distintos. Además, requiere ejecutar el algoritmo repetidas veces.

-*Análisis teórico*: Implica encontrar una cuota máxima para expresar el tiempo de nuestro algoritmo, sin necesidad de ejecutarlo. Se considera:

- Sólo las instrucciones elementales del algoritmo: asignación, y operaciones aritmético/lógicas.

- Una instrucción elemental utiliza un tiempo constante para su ejecución, independientemente del tipo de dato con el que trabaje: 1UT (UT = unidad de tiempo).

**Fórmulas:**

-if/else: se suman UT por cada comparación. Si hay solo if se suman todas las UT dentro. Si hay un else también se calcula el tamaño del if y el del else y se toma el peor caso. Por ej: MAX(11,9) 🡪 tomo el 11

-for: 3(N) + 2 + N(cuerpo) 🡪 N= veces que se repite

-while: N + 1 + N(cuerpo)