Informática: Es la ciencia que estudia el análisis y resolución de problemas utilizando computadoras.

Computadora: Es una máquina digital y sincrónica, con cierta capacidad de cálculo numérico y lógico, controlada por un programa almacenado y con posibilidad de comunicación con el mundo exterior.

Programa: Conjunto de instrucciones , ejecutables sobre una computadora, que permite cumplir una función específica. Normalmente, alcanzan su función en tiempo finito.

Dato: Es una representación de un objeto del mundo real mediante la cual se pueden modelizar aspectos de un problema que se desea resolver con un programa sobre una computadora.

Modelización de problemas del mundo real

Abstracción: Es el proceso de análisis del mundo real para interpretar los aspectos esenciales de un problema y expresarlo en términos precisos.

Modelización: Es Abstraer un problema del mundo real y simplificar su expresión, tratando de encontrar los aspectos principales que se pueden resolver (requerimientos), los datos que se han de procesar y el contexto del problema.

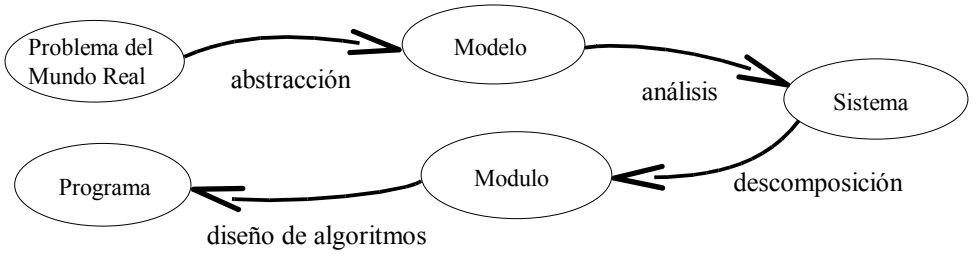
Pre condición: Es una información que se conoce como verdadera antes de iniciar el programa.

Pos condición: Es una información que debiera ser verdadera al concluir el programa, si se cumple adecuadamente el requerimiento pedido.

Especificación de problemas del mundo real

Especificación: Es el proceso de analizar los problemas del mundo real, determinar en forma clara y concreta el objetivo que se desea. Todo esto condiciona el resultado final del programa.

Lenguaje de programación: Es el conjunto de instrucciones permitidas y definidas por sus reglas sintácticas y su valor semántico, para la expresión de soluciones a problemas.



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

Las partes de un programa

Instrucciones: Representación de las operaciones que ejecutará la computadora al interpretar el programa.

Datos: Son valores de información de los que se necesita disponer, y en ocasiones transformar, para ejecutar la función del programa. Tanto los datos constantes como los variables deben guardarse en la memoria de datos de una computadora. En ambos casos estarán representados simbólicamente por un nombre que se asocia con una dirección única de memoria.

Comentarios: Son textos aclaratorios para el programador o usuario, que no es entendido ni ejecutado por la computadora.

Corrección de un Programa

Un programa es correcto si cumple con su especificación. Para poder evaluar el comportamiento de un programa es necesario tener una especificación clara de lo que debe hacer. Hay varias **formas de evidenciar que un programa es correcto**:

Testeo de un programa (Testing)

El objetivo del proceso de prueba es **evidenciar que el programa trabaja**. Esta batería de **pruebas (diseñadas adecuadamente)** debe ser planteada y no implementada de forma aleatoria. Lo que cuenta es la **calidad de las pruebas, no la cantidad.** Si la prueba **sale mal**, el programa debe ser **corregido y probado con la misma batería de test** que las que tuvo cuando falló.

Debugging

El proceso de **descubrir un error en el código y repararlo** es lo que se conoce como debugging. Los errores pueden aparecer en el programa de **distintas formas:**

- **Problemas en el diseño del programa**, será necesario corregirlo o rehacerlo.
- **La utilización de un proceso erróneo.**
- **Codificación incorrecta.**
- **Errores sintácticos, semánticos o lógicos**, es decir errores de diseño.
- **Errores de hardware o errores del sistema (poco probables).**
- **El programa puede ser aplicado a situaciones no previstas produciendo resultados incorrectos.**

A veces hace falta poner checkpoints o breakpoints, o incluso hacer un seguimiento paso a paso (trace), inspeccionando los valores de las variables hasta hallar un error.

Walkthroughs

La idea es s**eguir o leer el código de programa ya escrito intentando comprender, o haciendo comprender a otro como funciona.** Un programa bien estructurado es fácil de leer.

Verificación

Cuando probamos nuestro programa, **observamos y analizamos su comportamiento.** Sobre la base de este análisis podemos hacer ciertas afirmaciones del tipo: si nuestro programa es utilizado con ciertos datos, produce ciertos resultados. En otras palabras **se verifica si se cumplen las pre y poscondiciones del programa íntegramente.**

Etapas en la resolución de problemas con la computadora

Análisis del problema

En esta primera etapa, se analiza el problema en su contexto del mundo real. Deben obtenerse los requerimientos del usuario. El resultado de este análisis es un modelo preciso del ambiente del problema del objetivo a resolver . Un componente importante de este modelo don los datos a utilizar y las transformaciones de los mismos que llevan al objetivo.

Diseño de una solución

Suponiendo que el problema es computable, a partir del modelo se debe definir una estructura de sistema de hardware y software que lo resuelva. El primer paso en el diseño de la solución es la modularización del problema, es decir , la descomposición del mismo en partes que tendrán una función bien definida y datos propios (locales). A su vez, debe establecerse la comunicación entre los módulos del sistema de software propuesto.

Especificación de algoritmos

Cada uno de los módulos del sistema de software tiene una función que se puede traducir en un algoritmo. La elección del algoritmo adecuado para la función del módulo es muy importante para la eficiencia posterior del sistema de software.

Escritura de programas

Un algoritmo es una especificación simbólica que debe convertirse en un programa real sobre un lenguaje de programación concreto. Este proceso de programación tiende a automatizarse en la medida que los lenguajes algorítmicos se acercan a los lenguajes reales de programación. A su vez el programa escrito debe compilarse, lo que permite detectar y corregir errores sintácticos que se cometan en la escritura del programa.

Verificación

Una vez que se tiene un programa escrito en un lenguaje real y depurado de errores sintácticos, se debe verificar que su ejecución conduzca al resultado deseado, con datos representativos del problema real. Sería deseable poder afirmar que el programa es correcto más allá de los datos particulares de la ejecución. Sin embargo, en los casos reales es muy difícil realizar una verificación exhaustiva de todas las posibles condiciones de ejecución de un sistema de software.

La facilidad de verificación y la depuración de errores de funcionamiento del programa conducen a una mejor calidad del sistema y este es el objetivo central de la Ingeniería de Software. Nótese que un error de funcionamiento de alguno de los programas del sistema puede llevar a la necesidad de rehacer cualquiera de las etapas anteriores, incluso volver a discutir los requerimientos.

Eficiencia de un algoritmo

Eficiencia: Es la métrica de calidad de los algoritmos, asociada con una utilización óptima de los recursos del sistema de cómputo donde se ejecutará el algoritmo.

Tipos de datos

Tipo de dato: Es una clase de objeto ligado a un conjunto de operaciones para crearlo y manipularlo. Se caracteriza por tener:

- Un rango de valores posibles.
- Un conjunto de operaciones realizables sobre ese tipo.
- Su representación interna.

Tipo de datos simples.

Los tipos de datos simples son:

- Numérico (Entero, Real, etc.).
- Lógico (Boolean).
- Carácter(Char).

Tipo de dato definido por el usuario: Es aquel tipo de dato que no existe en la definición del lenguaje, donde el usuario es el encargado de determinar su denominación, y el conjunto de valores operaciones que dispondrá el mismo. Sus ventajas son:

- Flexibilidad.
- Facilitan la documentación.
- Seguridad.

Los tipos de datos definidos por el usuario son:

- **Enumerativo:** Es una colección de datos cuyos valores son constantes simbólicas (identificadores que detallan los valores posibles del tipo) que solo permite operaciones de asignación o comparación, dado que se inicializa con valores.
- **Subrango**: Es un tipo ordinal, que consiste en una secuencia de algún tipo ordinal, llamado tipo pase del subrango.
- **Conjunto:** Es una colección de datos simples, todos del mismo tipo.
- **String:** Es una sucesión de caracteres que se almacenan en un área contigua de la memoria y puede ser leído o escrito.

Estructura de datos

Estructura de datos: Es un conjunto de variables (no necesariamente del mismo tipo) relacionadas entre si de diversas formas.

Clasificación

- **Por tipo de dato: Simples** (Entero, carácter, etc.) o **Compuestas** (registro, pila, etc.).
- **Por contenido: Homogéneas** (pilas, colas, etc.) o **Heterogéneas** (registros, etc.).
- **Por alocación en memoria: Estáticas** (Arreglos, etc.) o **Dinámicas** (listas, arboles, etc.).
- **Por tipo de acceso: Indexado** (Arreglos, etc.) o **Secuencial** (listas, etc.).

Tipo de datos compuesto

Registro

Un registro es un **conjunto de valores** , con tres características básicas :

- Es una estructura **Heterogénea.**
- Los valores almacenados en un registro son llamados **campos**, y cada uno de ellos tiene un identificador; los campos son nombrados individualmente, como variables ordinarias.
- Es una estructura **Estática.**

Pilas

Una pila es una **colección ordenada de elementos**, con tres características:

- Es una estructura **Homogénea**.
- Los elementos pueden recuperarse en orden inverso al que fueron almacenados (**LIFO**).
- Es una estructura **Dinámica**.

Colas

Una cola es una **colección ordenada de elementos**, con tres características:

- Es una estructura **Homogénea.**
- Los elementos pueden recuperarse en el orden en que fueron almacenados (**FIFO**).
- Es una estructura **Dinámica**.

Tipo de datos compuestos Indexados

Arreglo

Un arreglo es una **colección ordenada e indexada** de elementos, con las siguientes características:

- Es una estructura **Homogénea**.
- Los elementos pueden **recuperarse en cualquier orden**, simplemente indicando la **posición** que ocupan dentro de la estructura; por este motivo es una **estructura indexada**.
- Es una estructura **Estática**.

Vectores

Es un **arreglo lineal**, tiene solo un índice, o sea **una sola dimensión**.

Matriz

Es un tipo de dato con **dos dimensiones** o índices. También puede pensarse en ella como un **vector de vectores**. Es un grupo de elementos **homogéneo**, con un **orden interno** y en el que se necesitan **dos índices para referenciar un único elemento** de la estructura.

Recursividad

La **Recursividad** se da cuando se obtienen soluciones a problemas en los que una **función o procedimientos se llama a si mismo** para resolver el problema, entonces se obtiene subprogramas recursivos.

Análisis de algoritmos: El concepto de eficiencia

Un algoritmo es eficiente si realiza una **administración correcta** de los **recursos del sistema** en el cual se ejecuta.

Se analizan básicamente dos aspectos de la eficiencia de un algoritmo:

- **Tiempo de ejecución.**

  Desde este punto de vista se consideran mas eficientes aquellos algoritmos que **cumplan con la especificación** del problema en el **menor tiempo posible**. En este caso, el recurso a **optimizar** es **el tiempo de procesamiento**. A esta clase pertenecen aquellas aplicaciones con **tiempo de respuesta finito**.

- **Administración o uso de la memoria.**

  Serán eficientes aquellos algoritmos que utilicen las **estructuras de datos adecuadas** de manera de **minimizar la memoria ocupada**. Si bien el **tiempo de procesamiento no se deja de lado**, en este punto se consideran los problemas donde el **énfasis** esta puesto en el v**olumen de la información a manejar en la memoria**.

Hay dos formas de **estimar el tiempo de ejecución de un algoritmo**:

1) Realizar un **análisis teórico** : Se busca obtener una medida del trabajo realizado por el algoritmo a fin de obtener una estimación teórica de su tiempo de ejecución. Básicamente, se calculan el número de comparaciones y de asignaciones que requiere el

algoritmo. Los análisis similares realizados sobre diferentes soluciones de un mismo problema permiten estimar cuál es la solución más eficiente.

2) Realizar un **análisis empírico**: se basa en la aplicación de juegos de datos diferentes a una implementación del algoritmo, de manera de medir sus tiempos de respuesta. La aplicación de los mismos datos a distintas soluciones del mismo problema presupone la obtención de una herramienta de comparación entre ellos. El análisis empírico tiene la ventaja de ser muy fácil de implementar, pero **no tiene en cuenta algunos factores** como:
   - **La velocidad de la máquina**, esto es, la ejecución del mismo algoritmo en computadoras distintas produce diferentes resultados. De esta forma no es posible tener una medida de referencia.
   - **Los datos con los que se ejecuta el algoritmo**, ya que los datos empleados pueden ser favorables a una de los algoritmos, pero pueden no representar el caso general, con lo cual las conclusiones de la evaluación pueden ser erróneas.

Reglas generales

Regla 1: Para lazos incondicionales.

El tiempo de ejecución de un lazo incondicional es, a lo sumo, el tiempo de ejecución de las sentencias que están dentro del lazo, incluyendo testeos, multiplicada por la cantidad de iteraciones que se realizan.

Regla 2: Para lazos incondicionales anidados.

Se debe **realizar el análisis desde adentro hacia afuera**. El tiempo total de ejecución de un bloque dentro de lazos anidados es el tiempo de ejecución del bloque multiplicado por el producto de los tamaños de todos los lazos incondicionales.

Regla 3: If - Then - Else.

Dado un fragmento de código de la forma:

If condición
 then S1
 else S2

El tiempo de ejecución no puede ser superior al tiempo del testeo más el máx(t1,t2) donde t1 es el tiempo de ejecución de S1, y t2 es el tiempo de ejecución de S2.

Regla 4: Para sentencias consecutivas.

Eficiencia en algoritmos recursivos

Desventajas desde el punto de vista de la eficiencia

- El **overhead** (sobrecarga) de tiempo y memoria asociado con la llamada a subprogramas.
- La **ineficiencia inherente** de algunos **algoritmos** recursivos.

Algoritmos de busqueda

Definición: El proceso de ubicar información particular en una colección de datos es conocido como algoritmos de búsqueda.

Búsqueda Lineal

Búsqueda Binaria

La búsqueda binaria sobre un vector ordenado se basa en la estrategia de “divide y vencerás” y aplica el mismo criterio que el que se utiliza para encontrar el cero de una función continua.

Algoritmos de ordenación

Definición: el proceso por el cual, un grupo de elementos se puede ordenar se conoce como algoritmo de ordenación.

Método de selección

Para ordenar un vector de n elementos, a[1], ... , a[n], busca el elemento menor según el criterio adoptado y lo ubica al comienzo. Para mantener la dimensión del vector , se intercambia con a[1]. A continuación se busca el segundo elemento menor del vector y se intercambia con a [2]. Continúa así sucesivamente buscando el próximo menor y lo coloca en su lugar, hasta que todos los elementos quedan ordenados.

Para ordenar un vector completo es necesario realizar n-1 pasadas por el vector.

Método de intercambio o Burbujeo

Procede de una manera similar al de selección, en el sentido de que realiza n-1 pasadas sobre los datos para ordenar un vector de n ítems. Pero a diferencia del anterior, no mueve los elementos grandes distancias, sino que a lo sumo realiza correcciones locales de distancia 1, es decir, compara ítems adyacentes y los intercambia si están desordenados. Al final de la primera pasada, se habrán comparado n-1 pares y el elemento más grande habrá sido arrastrado, hacia el final del vector. Al final de la segunda pasada, el segundo máximo habrá sido ubicado en la posición n-1. En general, en la i-ésima pasada, se posicionará el máximo entre los i primero elementos del vector, comparando i-1 pares e intercambiando en cada comparación de ser necesario. Es importante notar que a pesar de no haber cambios, los elementos del vector se siguen comparando hasta terminar, realizando de esta forma un trabajo innecesario.

Esto puede corregirse fácilmente inspeccionando cuándo ha habido cambios y en caso de no producirse ninguno, se termina el proceso.

Método de inserción

Este método ordena le vector de entrada insertando cada elemento a[i] entre los i-1 anteriores que ya están ordenados. Para realizar esto comienza a partir del segundo elemento, suponiendo que el primero está ordenado. Si los dos primero elementos están desordenados, los intercambia. Luego, toma el tercer elemento y busca su posición correcta con respecto a los dos primeros. En general, para el elemento i, busca su posición con respecto al los i-1 elementos anteriores y de ser necesario lo inserta adecuadamente.

Datos compuestos enlazados

Lista como estructura de datos

Definición: una lista dinámica es un **conjunto de elementos de tipo homogéneo**, donde los mismos **no ocupan posiciones secuenciales** o contiguas de memoria, es decir, los elementos o componentes de una lista pueden aparecer físicamente dispersos en la memoria, si bien mantienen un **orden lógico interno**. Es una estructura **lineal**.

Punteros

Un puntero es un tipo de variable, en el cual se **almacena la dirección de un dato y permite manejar direcciones** "apuntando" a un elemento determinado.

Cada elemento de una lista puede representarse de la siguiente manera:

| Dato | Indicador |

El campo dato es la información que se maneja, y el campo indicador es una variable de tipo puntero, cuyo contenido es la dirección del próximo elemento de la estructura.

Operaciones con lista

- Recorrer una lista.
- Acceder al k-ésimo elemento de la lista.
- Intercalar un nuevo elemento de la lista.
- Agregar un elemento al final de la lista.
- Combinar dos listas, para formar una sola.
- Localizar un elemento de la lista para conocer su posición dentro de ella.
- Borrar un elemento de la lista.

Listas circulares

Lista enlazada en la cual **el último elemento apunta al primero** o principio de la lista.

Ventajas :

Cada nodo de la lista circular es **accesible desde cualquier otro nodo** de ella.

Las operaciones de concatenación y división de las listas son más eficaces con listas circulares.

Desventajas :

Se pueden producir **bucles o lazos infinitos**. Una forma de evitar estos bucles es disponer de un nodo especial denominado cabecera que se puede diferenciar de otros nodos por tener un valor especial

Lista doblemente enlazadas

Lista enlazada la cual **se puede recorrer en ambos sentidos.** En este tipo de listas, además del campo información se cuenta con dos variables del tipo puntero que apuntan al nodo anterior y siguiente.

Arboles

Un árbol es una estructura de datos que **satisface tres propiedades:**

(1) Cada elemento del árbol **(nodo) se puede relacionar con cero o más elementos**, los cuales llama **"hijos".**

(2) Si el árbol no está vacío, **hay un único elemento al cual se llama raíz y que no tiene padre (predecesor)**, es decir, no es hijo de ningún otro.

(3) **Todo otro elemento** del árbol **posee un único padre y es un descendiente** (hijo del hijo del hijo, etc.) **de la raíz.**

Arbol binario

Un árbol donde **cada nodo no tiene más de dos hijos** se denomina árbol binario y se dice que está ordenado cuando cada nodo del árbol es mayor (según algún criterio preestablecido) que los elemento de su subárbol izquierdo (el que tiene como raíz a su hijo izquierdo) y menor o igual que los de su subárbol derecho (el que tiene como raíz a su hijo derecho).

Introducción a tipos abstractos de datos (TAD)

Modulo: Es cada una de las partes funcionalmente independientes en que se divide un programa.

Interfaz: Es la especificación de las funcionalidades del módulo y que puede contener las declaraciones de tipos y variables que daban ser conocidas externamente. **(Pública).**

Implementación: Abarca el código de los procedimientos que concretan las funcionalidades internas. **(Privado).**

Encapsulamiento

Un paso tendiente a la creación de especificaciones abstractas verdaderas o abstraccíon de datos es lograr **encapsulamiento o empaquetamiento** de los datos: se define un nuevo tipo y se integran en un módulo todas las operaciones que se pueden hacer con él.

Si además el lenguaje permite **separar la parte visible (interfaz) de la implementación**, **se tendrá ocultamiento de datos (data hidding).**

Y si se logra que la solución del cuerpo del módulo pueda **modificar la representación del objeto-dato, sin cambiar la parte visible del módulo**, se tendrá **independencia de la representación**.

De hacer todo lo anterior, se tiende a **mejorar la calidad** de los programas, su **legibilidad**, su **reutilización** y su **mantenimiento**.

Diferencia entre tipo de dato y tipo abstracto de dato

El concepto de tipo abstracto de dato es un tipo de dato definido por el programador que incluye:

- Especificación de la representación de los elementos del tipo.
- Especificación de las operaciones permitidas con el tipo.
- Encapsulamiento de todo lo anterior, de manera que el usuario no pueda manipular los datos del objeto, excepto por el uso de las operaciones definidas en el punto anterior.
- La forma de programación utilizada en los capítulos anteriores corresponde a la clásica ecuación de Wirth (Wirth, 1984):

Programa = Datos + Algoritmos

Es decir, dado un problema a resolver, se busca representar los objetos que participan en el mismo e incorporar los algoritmos necesarios para llegar a la solución.

La concepción de este algoritmo se realiza pensando en la solución del problema en su conjunto y en tal sentido se aplican las técnicas de modularización a din de reducir la complejidad del problema original.

Se debe notar que el comportamiento de los datos no se tiene en cuenta en esta etapa. De esta forma, el programa implementado se aplica únicamente a la solución del problema original, ya que funciona como un todo. De otra manera, la única relación que existe entre su módulos tiene como fin la solución de un problema especifico.

Sin embargo, esto puede mejorarse si se refina la ecuación anterior. En la ecuación de Wirth la parte Algoritmos se puede expresar como:

Algoritmo = Algoritmo de datos + Algoritmo de control

donde se entiende como **Algoritmo de datos** a la parte del algoritmo encargada de manipular las estructuras de datos del problema, y **Algoritmo de control** a la parte que representa el método de solución del problema, independiente de las estructuras de datos seleccionadas.

Dado que los TAD reúnen en su definición la representación y el comportamiento de los objetos del mundo real se puede escribir la ecuación inicial como:

Programa = Datos + Algoritmos de Datos + Algoritmos de Control

Reemplazando "**Datos + Algoritmos de Datos**" como TAD se establece la siguiente ecuación:

Programa = TAD + Algoritmos de Control

que describe el enfoque de desarrollo utilizando Tipos Abstractos de Datos.

Se puede notar que en este caso, el programa principal es diferente al que resulta al no utilizar TAD, ya que la parte Algoritmos de Control se refiere a la relación de los distintos objetos para resolver el problema original pero el **comportamiento** de cada objeto se encuentra **definido en el TAD**.

Ventajas de uso de TAD

- En el momento de diseñar y desarrollar el TAD no interesa conocer la aplicación que lo utilizará. Esto permite concentrarse únicamente en la implementación del nuevo tipo.
- En el momento de utilizar el TAD no interesa saber cómo funcionaría internamente, solo bastará con conocer las operaciones que permiten manejarlo.

Formas de Abstracción: tipos de datos abstractos y tipos de datos lógicos

Al definir pilas y colas se ha realizado abstracción en dos direcciones: **abstracción de datos** (se definieron las **características del objeto**) y **abstracción de operaciones** (se definieron las **operaciones permitidas sobre el objeto indicando la interfaz**).

Este tipo de dato se denomina TDL (Tipo de dato lógico) pues existe conexión lógica, conocida por el usuario entre los tipos, por ejemplo, la declaración del tipo pila, y los procedimientos o funciones asociadas con dicho tipo.

Con los TDL no se llega a un verdadero **tipo abstracto de datos** porque no existe una vinculación estructural entre dicho tipo y las operaciones asociadas al mismo; es más: los datos declarados dentro del tipo podrían ser accedidos por oras operaciones y, asimismo, las operaciones asociadas con el tipo podrían ser utilizadas para otros datos.

De hecho **no existe** lo que se conoce como **encapsulamiento del dato abstracto**, salvo en la mente del programador o usuario.

Requerimientos y deseño de un TAD

Disponer de un TAD brinda la posibilidad de tener código reusable, donde se represente no solo la **estructura de los datos**, sino también su **comportamiento**. Esto requiere:

- Poder **encapsular** dentro de un módulo del lenguaje tanto la **especificación visible como la implementación de las operaciones.**

- Poder **declarar los tipos protegidos** de modo que la **representación interna esté oculta** de la parte visible del módulo.
- Poder **heredar el TAD**, es decir, **crear instancias a partir de ese molde**, donde se repitan o modifiquen las funcionalidades del tipo, respetando sus características básicas.

El **diseño** de un tipo de dato abstracto lleva consigo la selección de:

- **Una representación interna**, lo que implica **escoger las estructuras de datos adecuadas** para representar la información. Dicha selección está **estrechamente relacionada con la complejidad de la implementación de las operaciones**.
- Las **operaciones a proveer** para el nuevo tipo **y el grado de parametrización** de las mismas.

Estas **operaciones** pueden clasificarse en:

- Operaciones para **crear o inicializar objetos** que permite inicializar la estructura de dato o generar la situación inicial necesaria para **crear un objeto nuevo**.
- Operaciones para **modificar los objetos del TAD** que permiten agregar o quitar elementos del TAD.
- Operaciones que permiten **analizar los elementos del TAD**.

Archivos

Archivo: Es una estructura de datos que **guarda en un dispositivo de almacenamiento secundario** de una computadora **una colección de elementos** del mismo tipo. Es una **estructura** de datos **Homogénea y Dinámica.**

Se le debe asignar a una variable un nombre lógico, que referenciará indirectamente al archivo físico, esta variable será un archivo lógico.

Operaciones Básicas

- **Vincular o relacionar el archivo físico**(que está en memoria secundaria) **con el nombre lógico** que se utilizará dentro del algoritmo.
- **Abrir** un archivo para s u procesamiento.
- **Cerrar** un archivo, luego de operar con el mismo.
- **Leer** los elementos contenidos en un archivo.
- **Escribir nuevos elementos** en un archivo **o modificar alguno** existentes.

Organización y acceso a los datos

Acceso Secuencial : En un archivo con acceso secuencial, para acceder al elemento n-ésimo deberá accederse a los n – 1 elementos anteriores.

Acceso Directo: El acceso directo permite referenciar o cambiar elementos en cualquier orden.

Operaciones frecuentes en el procesamiento de datos

- **Merge:** Es la operación por la cual **se mezclan 2 o más archivos secuenciales** ordenados en uno sólo que tendrá la totalidad de los elementos. Cuando se exige que el **archivo resultante esté ordenado por algún criterio**, será necesario **intercalar elementos** de un archivo y de otro **según ese orden**. **Sino** se puede hacer un **append**.
- **Modificación de un archivo maestro con uno de detalle.**
- **Altas , bajas y modificaciones**