CONCEPTOS BASICOS

Definiciones

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

Computadora: es una maquina digital y sincrónica, con cierta capacidad de cálculo numérico y lógico, controlada por un programa almacenado (software) y con posibilidad de comunicación con el mundo exterior. Su finalidad es ayudar al hombre a realizar tareas repetitivas en menor tiempo y con mayor exactitud.

Programa: es un conjunto de instrucciones, ejecutables sobre una computadora, que permite cumplir una función específica.

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 (en forma binaria). Conceptualmente pueden ser constantes (no cambian su valor durante la ejecución del programa), o variables (pueden cambiar)

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.

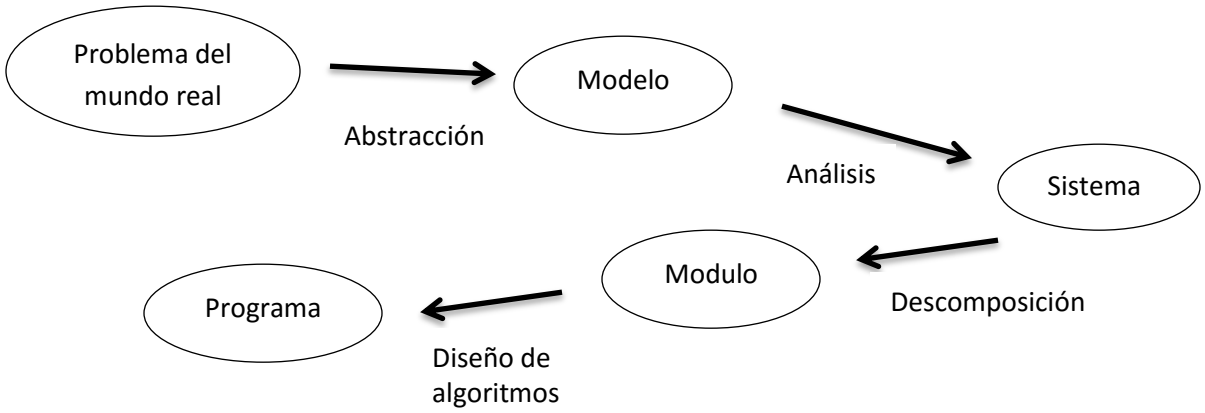
Precondición: es una información que se conoce como verdadera antes de iniciar el programa.

Poscondición: es una información que debiera ser verdadera al concluir un programa, si se cumple adecuadamente el requerimiento pedido.

Especificación de los problemas del mundo real

Especificación: es el proceso de analizar los problemas del mundo real y determinar en forma clara y concreta el objetivo que se desea.

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 especificacion rigurosa de la secuencia de pasos (instrucciones) a realizar sobre un autómata (sistema que realiza cosas automaticamente) para alcanzar un resultado deseado en un tiempo finito.

Especificacion rigurosa: debemos expresar un algoritmo en forma clara y univoca.

Tiempo finito: un algoritmo comienza y termina.

Las partes de un programa

Instrucciones: (o acciones) representan las operaciones que ejecutara 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 datos variables deben guardar 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 (diseñada 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 seguir o leer el código de programa ya escrito intentado 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 cumples las pre y pos condiciones del programa íntegramente.

Etapas en la resolución de problemas con la computadora

Análisis del problema: 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 y del objetivo a resolver. Un componente importante son los datos a utilizar y las trasformaciones de los mismos que llevan al objetivo.

Diseño de una solución: 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 y 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.

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 los 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. 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 ingeniera 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.

ALGORITMOS

Estructuras de control

Conjunto mínimo de instrucciones que permiten especificar el control del algoritmo que se quiere implementar. Este conjunto debe contener como mínimo:

1. **Secuencia:** representada por una sucesión de operaciones (asignaciones, etc.) en la que el orden de ejecución coincide con el orden físico de aparición de las instrucciones.
2. **Decisión:** instrucción no secuencial que permite tomar decisiones en función de los datos del problema. Estructura {IF (condición) THEN}, si la condición es verdadera ejecuta las instrucciones correspondientes, si es falsa se va por el {ELSE} y ejecuta la instrucciones.
3. **Selección:** permite realizar una o más acciones, dependiendo de cuál de todas las condiciones evaluada es verdadera. Estructura: {CASE variable OF}. La variable debe ser de tipo ordinal.
4. **Repetición:** consiste en repetir un número fijo y conocido de veces una o más acciones. Estructura: {FOR índice:valor_inicial/final TO/DOWNTO valor_final/inicial DO} (índice: debe ser de tipo ordinal (entero, boolean, char) y del tipo del identificador (en casi de tratarse de TDDU). No debe modificarse dentro del lazo. Los incrementos o decrementos y testeos son implícitos. Al terminar el ciclo, la variable índice no tiene un valor definido.
5. **Iteración:** ejecuta un bloque de instrucciones desconociendo el número exacto de veces que se ejecutan:
    a. **Precondicional:** evalúa la condición y luego ejecuta el bloque de instrucciones. Se ejecuta de 0 a más veces, mientras la condición sea verdadera. Estructura: {WHILE (condición) DO}
    b. **Postcondicional:** ejecuta el bloque de acciones y luego evalúan la condición. Se ejecuta de 1 a más veces, mientras la condición sea falsa. Estructura: {REPEAT bloque de acciones UNTIL (condición)}

Eficiencia de un algoritmo

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

Modularización

Los problemas del mundo real implican complejidad, extensión y modificaciones. A estos los tratamos de resolver con abstracción, descomposición e independencia funcional.

Modularizar significa dividir un problema en partes funcionalmente independientes, que encapsulen operaciones y datos. En otras palabras, separar en funciones lógicas con datos propios y datos de comunicación perfectamente especificados.

Cuando se realiza la descomposición debemos tener en cuenta que cada subproblema está en un mismo nivel de detalle, que cada uno se puede resolver independientemente de los demás y que las soluciones de estos subproblemas pueden combinarse para resolver el problema original.

Módulo: tarea específica bien definida, se comunican entre sí adecuadamente y cooperan para conseguir un objetivo común. Cada módulo encapsula acciones, tareas o funciones. En otras palabras, es un conjunto de instrucciones.

Metodología: TOP DOWN, ir de lo general a lo particular. Dividir, conectar, y verificar.

Características:

- Mayor productividad: al dividir un sistema de software en módulos funcionalmente independientes, un equipo de desarrollo puede trabajar simultáneamente en varios módulos, incrementando la productividad (es decir reduciendo el tiempo de desarrollo global del sistema).
- Reusabilidad: posibilidad de utilizar repetidamente el producto de software desarrollado. Naturalmente la descomposición funcional que ofrece la modularización favorece el re-uso.
- Facilidad de mantenimiento correctivo: la división lógica de un sistema en módulos permite aislar los errores que se producen con mayor facilidad. Esto significa poder corregir los errores en menor tiempo y disminuye los costos de mantenimiento de los sistemas.
- Facilidades de crecimiento del sistema: los sistemas de software reales crecen (aparecen con el tiempo nuevos requerimientos del usuario). La modularización permite disminuir los riesgos y costos de incorporar nuevas prestaciones a un sistema en funcionamiento.
- Mayor legibilidad: 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.

Forma de Modularizar:

- Procedimientos: modulo que realiza una tarea específica y retorna 0, 1 o más valores por parámetros. La comunicación es a través de parámetros por valor o referencia. Sin parámetros no devuelve nada. Se invoca escribiendo su nombre.

- Función: modulo que realiza una tarea específica y retorna siempre 1 valor de tipo simple (real, integer, char, boolean). Para devolver el resultado se asigna el nombre de la función como ultima instrucción. Se puede invocar dentro de un if, de un while, asignarla a una variable o dentro de un write. Puede recibir solo parámetros de entrada.

Parámetros: son datos que tienen como característica principal transferir información entre los módulos.
Parámetro por valor: un dato de entrada por valor es llamado parámetro IN y significa que el modulo recibe (sobre una variable local) un valor proveniente de otro modulo (o del programa principal). Con él puede realizar operaciones y/o cálculos, pero no producirá ningún cambio ni tampoco tendrá incidencia fuera del módulo.
Parámetro por referencia: la comunicación por referencia (OUT, INOUT) significa que el modulo recibe el nombre de una variable (referencia a una dirección) conocida en otros módulos del sistema. Puede operar con ella y su valor original dentro del módulo, y las modificaciones que se produzcan se reflejan en los demás módulos que conocen la variable.
Variable global: son aquellas que se declaran en la sección de declaración del programa principal y puede ser utilizada por cualquier módulo de este.
Variable local: son aquellos que se declaran y definen en la sección de declaración de un módulo individual.

TIPOS DE DATOS

Tipo de dato: es una clase de objeto ligado a un conjunto de operaciones para crearlo y manipularlo. Se caracterizan por tener un rango de valores posibles, un conjunto de operaciones realizables sobre este tipo y su representación interna.

Tipos de datos simples:

- Numérico (entero, real, etc.): conjunto de valores que se pueden representar de forma entera (integer) o de forma real (real: permite representar números con decimales).
  Las operaciones válidas para este tipo de dato son suma (+), resta (-), multiplicación (*), división (/), división entera (div: da como resultado el cociente de una división) y modulo (mod: da como resultado el resto de una división).
  Los operadores relacionales son igualdad (=), desigualdad (<>), y de orden (<, <=, >, >=).
- Lógico (boolean): puede tomar el valor verdadero (true) o el falso (false). Los operadores lógicos básicos son negación (not), conjunción (and) y disyunción (or).
- Carácter (char): conjunto finito y ordenado de caracteres que la computadora reconoce. Este tipo trabaja con un único carácter por vez. Se debe tener en cuenta que no es o mismo el valor entero 0 que el símbolo carácter '0'.

Otras operaciones
Read: se usa para leer datos (por defecto del teclado) y asignarlos a las variables correspondientes.
Write: se usa para mostrar el contenido de una variable, por defecto en pantalla.
Los diferentes tipos de datos se declaran en el programa a través de **variables:** zona de memoria, que los contiene. La dirección inicial de esta zona se asocia con el nombre de la variable. Una variable perteneciente a un tipo de dato predefinido por el lenguaje tiene asociado un rango de valores posibles, un conjunto de operaciones validad y una representación interna.
Constantes: su valor no cambia en la ejecución del programa. Se indica un valor en su declaración. Se declara con la palabra clave *const.* {Const
Nombre = valor;}
Variables: su valor cambia en la ejecución del programa. Se indica un valor en el programa. Se declara con la palabra clave *var.* {Var
Nombre: tipodedato;}

Tipos de datos definidos por el usuario: es aquel tipo de dato que no existe en la definición del lenguaje, donde el usuario es el encargado de determina su denominación, y el conjunto de valores y operaciones que dispondrá el mismo.
Estructura: {TYPE
Identificador (nombre) = tipo (estándar o definido por el usuario);}

Sus ventajas son:

- Flexibilidad: en el caso de ser necesario modificar la forma en que se representa el dato, solo 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.

1- **Tipo de dato ENUMERATIVO:** es una colección de datos cuyos valores son constantes simbólicas (identificadores que detallas los valores posibles del tipo) que solo permite operaciones de asignación o comparación, dado que se inicializa con valores.
Estructura: {TYPE
Identificador = (valor1, valor2, valor3, valor4, etc.);}
Características: lista ordenada de valores posibles para las variables asociadas al tipo. Los valores no pueden ser estándar, tampoco pueden repetirse entre identificadores distintos. Es un tipo de datos simple y ordinal. Las operaciones permitidas son asignación, comparación, sucesor (succ), predecesor (pred), ordinal (ord) y *case* identificador *of.*

2- **Tipo de dato SUBRANGO:** es un tipo ordinal, que consiste en una secuencia contigua de valores de algún tipo ordinal, llamado tipo pase del subrango.
Estructura: {TYPE
Identificador = valorInicial .. valorFinal;}
Es un tipo de dato simple y ordinal. Facilita el cheque de posibles errores, permite que el lenguaje verifique si los valores asignados estén dentro del rango establecido. Ayuda al mantenimiento del programa.

3- **Tipo de dato CONJUNTO:** es una colección de datos simples, todos del mismo tipo.
Estructura: {TYPE
Identificador = set of tipoOrdinal;} (integer, char, boolean, enumerativo: previamente declarado)
Cantidad máxima de elementos 255. Las operaciones permitidas son asignación, unión (+), diferencia (-), intersección (*), pertenencia (IN), comparación y *case* identificador *of*.

4- **Tipo de dato STRING:** es una sucesión de caracteres que se almacenan en un área contigua de la memoria y puede ser leído o escrito.
Estructura: {TYPE
Identificador = string[k];} donde k es la longitud máxima del string
Identificador = string;} se supone como 255 caracteres como máximo
Se almacena en un área contigua de la memoria. No es un dato simple. Las operaciones validas son asignación (truncamiento de string), comparación (evalúa las longitudes, si son iguales evalúa el contenido), concatenación (agrega un string a continuación de otro (+) [cad1:=”ho”+”la”]), copiar (retorna un substring del string original, a partir de una posición y de largo igual a cantidad. El resultado es otro string), longitud (retorna la cantidad de caracteres del string. El resultado es un numero entero [length (string)]), concatenar (concatena dos string en un tercero [concat (string1, string2, string3)], mayúsculas (convierte a mayúsculas todos los caracteres que lo componen [strupper(string1)], minúsculas (convierte a minúsculas todos los caracteres que lo componen [strlower(string1)].

ESTRUCTURA DE DATOS

Es un conjunto de variables (no necesariamente del mismo tipo) relacionadas entre sí de diversas formas.

Clasificación:

- **Por tipo de dato:** Simples (entero, carácter, etc.) Compuestas (registro, pilas, etc.).
- **Por contenido:** Homogéneas (pilas, colas, etc.) Heterogéneas (registros, etc.).
- **Por alocación en memoria:** Estáticas (arreglos, etc.) Dinámicas (listas, arboles, etc.).
- **Por tipo de acceso:** Indexado (arreglos, etc.) Secuencia (listas, etc.).

Estructuras de datos compuestas

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

1) es una estructura heterogénea.

2) 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.

3) es una estructura estática.

```
Estructura: {TYPE
        Identificador = RECORD
                Campo1:tipo;
                Campo2:tipo;
        End;}
```

Pilas: es una colección ordenada de elementos, con tres características básicas:

1) Es una estructura homogénea.
2) Los elementos pueden recuperarse en orden inverso al que fueron almacenados (LIFO: last in first out)
3) Es una estructura dinámica.

Colas: es una colección ordenada de elementos, con tres características básicas:

1) Es una estructura homogénea.
2) Los elementos pueden recuperarse en el orden en que fueron almacenados (FIFO: first in first out).
3) Es una estructura dinámica.

Datos compuestos indexados

Arreglos: es una colección ordenada e indexada de elementos, con las siguientes características:

1) Es una estructura homogénea.
2) Los elementos pueden recuperarse en cualquier orden, simplemente indicando la posición que ocupan dentro de la estructura; por este motivo es una estructura indexada.
3) Es una estructura dinámica.

```
Estructura: {TYPE
        Vector = array [rango] of tipo;}
```

Rango: integer, char, boolean, enumerativo.
Tipo: integer, char, boolean, enumerativo, real, registro.

Vectores: es un arreglo lineal, tiene solo un índice, o sea una sola dimensión.

Matrices: 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 homogéneos, con un orden interno y en el que se necesitan dos índices para referenciar un único elemento de la estructura.

```
Estructura: {TYPE
        Matriz = array [fila,columna] of tipo;}
```

Fila, Columna: integer, char, boolean, enumerativo.
Tipo: integer, char, boolean, enumerativo, real, registro.

Índice: variable ordinal, permite un acceso a los elementos (ubica/direcciona). El desplazamiento se realiza desde la posición inicial.

Tiene dimensión física (tamaño fijo, declarado) y dimensión lógica (tamaño máximo ocupado). La dimL debe ser siempre menor a la dimF.

RECURSIVIDAD

La recursividad se da cuando se obtienen soluciones a problemas en los que una función o procedimientos se llaman a sí mismo para resolver el problema, entonces se obtiene subprogramas recursivos.
Resuelve problemas por resolución de instancias más pequeñas del mismo problema, es decir, el problema siempre es el mismo pero de menor tamaño. Tiene un caso base o degenerado. En cada llamada el tamaño del problema se achica.

Todo problema recursivo tiene una solución iterativa, pero no siempre sucede la viceversa. NO es más eficiente que una solución secuencial. Muchas veces se ejecutan más lento y no usan de manera eficiente la memoria.

ANALISIS 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 más 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 volumen de la información a manejar en la memoria.

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 cual 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. Tiene la ventaja de ser muy fácil de implementar, pero no tiene en cuenta algunos factores como la velocidad de la máquina y los datos con los que se ejecuta el algoritmo.

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 max (t1, t2) donde t1 es el tiempo de ejecución de S1, y t2 es el tiempo de ejecución S2.

Regla 4: para sentencias consecutivas.

Eficiencia en algoritmos recursivos: el verdadero uso de la recursividad es como una herramienta para resolver problemas para los cuales no hay una solución iterativa sencilla. En estos casos la recursividad permite obtener una expresión clara del algoritmo en base a la definición del problema original.
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.

Algoritmo de búsqueda

Es el proceso de ubicar información particular en una colección de datos, por ejemplo una estructura con un componente igual a x, y en caso de encontrarlo retornar alguna información relacionada con el ítem, por ejemplo su posición.

Búsqueda lineal: es el proceso de buscar desde el principio de la estructura, analizando los elementos que contiene uno a uno hasta encontrarlo o hasta llegar al final. Requiere excesivo consumo de tiempo. Es ineficiente en estructuras muy grandes.

Búsqueda binaria: sobre un vector ordenando 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.
Pasos para realizarla, calcula la primer y última posición para obtener el medio, luego comprar si el elemento buscado es igual al que se encuentra en el medio, en caso de no ser el buscado, si el elemento buscado es menor se continua la búsqueda en la primera mitad del vector, en caso contrario sobre la segunda mitad. El proceso termina cuando se encuentra el elemento buscado o cuando ya no hay elementos por comparar.

Algoritmos de ordenación

Es el proceso por el cual un grupo de elementos se puede ordenar.

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]. Continua 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. Pero a diferencia del anterior, 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, como una burbuja, hacia el final del vector. Al final de la pasada, el segundo máximo habrá sido ubicado en la posición n-1.

Método de inserción: ordena el 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 ya está ordenando. Si los dos primeros elementos están desordenados, los intercambia. Luego, toma el tercer elemento y busca su posición correcta con respecto a los dos primeros.

DATOS COMPUESTOS ENLAZADOS: LISTAS, ARBOLES, GRAFOS

Listas como estructura de datos

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: 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 la lista puede representarse de la siguiente manera: {|dato|indicador|}
Donde dato es la información que se maneja, el que puede estar constituido por un conjunto de elementos de algún tipo previamente definido y el campo indicador es una variable de tipo puntero, cuyo contenido es la dirección del próximo elemento de la estructura.

Operaciones con listas:

- Recorrer una lista
- Acceder al k-esimo 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 listar para conocer su posición dentro de ella
- Borrar un elemento de la lista

Listas circulares: es una lista enlazada en la cual el último elemento apunta al primero o principio de la lista.
Las ventajas que tiene es que cada nodo de la lista es accesible desde cualquier otro nodo de ella. Y las operaciones de concatenación y división de las listas son más eficaces.

La desventaja es que se pueden producir bucles o lazos infinitos. Una forma de evitar esto es disponer de un nodo especial denominado cabecera que se diferencia de otros nodos por tener un valor especial.

Listas doblemente enlazadas: en estas 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. Ocupan más memoria por nodo.

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.

Árbol binario: es un árbol donde cada nodo no tiene más de dos hijos. Se dice que esta ordenado cuando cada nodo del árbol es mayor que los elementos 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).

INTRODUCCION A TIPOS ABSTRACTOS DE DATOS (TAD)

Abstracciones de datos: la utilidad fundamental de la abstracción y la modelización de objetos del mundo real es la posibilidad de reusar soluciones que respondan a dicho modelo en diferentes problemas.

Modulo, interfaz e implementación

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

Interfaz: es una especificación de las funcionalidades del módulo y puede contener las declaraciones de tipos y variables que deban ser conocidas externamente. Se la conoce como parte pública.

Implementación: abarca el código de los procedimientos que concretan las funcionalidades internas. Se la conoce como parte privada.

Encapsulamiento de datos

También llamado empaquetamiento. 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.

Diferencias entre tipo de dato y tipo de dato abstracto

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 objetos, excepto por el uso de las operaciones definidas en el punto anterior.
- La forma de programación utilizada corresponde a la clásica ecuación de Wirth (1984)
    - Programa = Datos + Algoritmos

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. Se realiza pensando en la solución del problema en su conjunto y se aplican las técnicas de modularización a fin de reducir la complejidad del problema original.

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.

Sin embargo, esto puede mejorarse si se refina la ecuación anterior:

- Algoritmos = Algoritmo de datos + Algoritmo de control

donde se entiende como “algoritmo de datos” a la parte 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.
Para implementar un TAD se requiere especificar un módulo en el que se separan la interfaz (visible) de la implementación (oculta), para lograr el ocultamiento y la protección de los datos.

Ventajas del uso de TAD respecto de la programación convencional

La independencia facilita la re-usabilidad del código.
El programa o modulo que referencia al TAD, lo utiliza como “caja negra”, esto permite que las modificaciones internas del TAD no afecten a quienes lo utilizan, siempre que su interfaz no se modifique.
Se diferencian dos etapas:

- En el momento de diseñar y desarrollar el TAD no interesa conocer la aplicación que lo utilizara. Esto permite concentrarse únicamente en la implementación del nuevo tipo.
- En el momento de utilizar el TAD no interesa saber cómo funcionara internamente, solo bastara 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 y abstracción de operaciones. Este tipo de datos se denomina TDL (tipo de dato lógico) pues existe una conexión lógica, conocida por el usuario entre los tipos. 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. De hecho, no existe lo que se conoce como encapsulamiento del dato abstracto, salvo en la mente del programador o usuario.

Requerimientos y diseño de un TAD

Disponer de un TAD brinda la posibilidad de tener código reusable donde se represente la estructura de datos y también su comportamiento. Esto requiere:

- Poder encapsular la especificación visible y la implementación de las operaciones.
- Poder declarar tipos protegidos de modo que la representación interna este 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, implica escoger estructuras de datos adecuadas para representar la información.
- 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 permiten inicializar la estructura de datos 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.

INTRODUCCION AL CONCEPTO DE ARCHIVOS

Un 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.

Operaciones básicas sobre archivos:

- Vincular o relacionar el archivo físico (que está en memoria secundaria) con el nombre lógico que se utilizara dentro del algoritmo.
- Abrir un archivo para su 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 algunos existentes.

Técnicas de organización y acceso a un archivo:

Según el modo en que se organizan:

- Acceso secuencial: permite acceder a los registros o elementos uno tras otro y en el orden físico en que están guardados. Consiste de un conjunto de registros almacenados consecutivamente de manera que para acceder al registro n-ésimo del mismo se debe acceder a los n-1 registros anteriores.
- Acceso directo: permite obtener un registro determinado sin necesidad de haber accedido a sus predecesores. Consiste de un conjunto de registros donde el ordenamiento físico no necesariamente corresponde con el ordenamiento lógico. Los registros se recuperan accediendo por su posición dentro del archivo. Presenta la ventaja que se puede obtener cualquier elemento del archivo en cualquier orden. Y presenta el inconveniente de tener que determinar la posición donde se encuentra cada elemento.
- Secuencial indizado: utiliza estructuras de datos auxiliares para permitir un acceso pseudo directo a los registros de un archivo. Tienen la ventaja de tener un acceso mucho más rápido que los secuenciales, pero necesitan más espacio para mantener la estructura del índice.

Buffers: es una memoria intermedia entre un archivo y un programa, donde los datos residen provisoriamente hasta ser almacenados definitivamente en memoria secundaria o donde los datos residen una vez recuperados de dicha memoria secundaria. Los buffers ocupan una zona de memoria RAM de la computadora.

Operaciones básicas sobre archivos en Pascal

Declaración de archivos:

```
{Type
        Archivo: file of tipo_de_dato;
Var
        Arch:archivo;}
```

Relación con el sistema operativo:

El formato de la instrucción que permite establecer al correspondencia entre el nombre físico y el lógico es:

```
Assign (nombre_logico, nombre físico);
```

Apertura y creación de archivos:

```
Reset (nombre_logico); {abre un archivo existente como de lectura y escritura}
Rewrite (nombre_logico); {abre el archivo por primera vez, solo para escritura}
```

Cierre de archivos:

```
Close (nombre_logico);
```

Lectura y escritura de archivos:

```
Read (nombre_logico, variable); {leer}
Write (nombre_logico, variable); {escribir}
```

Operaciones adicionales:

- Para recorrer un archivo desde el primer registro hasta el final:

Eof (nombre_logico); retorna verdadero si la posición corriente dentro del archivo referencia a la marca de fin, falso en caso contrario.

- Para retornar el numero de elementos del archivo:

Filesize (nombre_logico); retorna un numero entero.

- Para la posición actual del apuntador en el archivo:

Filepos (nombre_logico); retorna un numero entero.

- Para ir a una posición especifica en el archivo:

Seek (nombre_logico, posición);

Algoritmos clásicos sobre archivos

Modificar el contenido actual: involucra un archivo de datos previamente generado, y consiste en modificar sus datos, recorriendo el archivo desde el primer elemento hasta el último.

Agregar nuevos elementos:

Actualización de un archivo maestro con uno o varios archivos detalle: se denomina archivo maestro a aquel archivo que resume un determinado conjunto de datos, en tanto que se denomina archivo detalle a aquel que agrupan información que se utilizara para modificar el contenido del archivo maestro.

Corte de control**:** consiste en la generación de reportes (informes impresos) a partir de información contenida en un archivo, siguiendo para ellos algunas pautas definidas. Dado que "corte de control" es un caso práctico; para definirlo se debe recurrir a la descripción de sus principales características de uso.

Merge o unión de varios archivos: es la operación por la cual se mezclan 2 o más archivos secuenciales ordenandos en uno solo que tendrá la totalidad de los elementos. Cuando se exige que el archivo resultante este ordenando por algún criterio, será necesario intercalar elementos de un archivo y de otro según ese orden. Sino se puede hacer un append.

Eliminar elementos de un archivo

Cuando un registro deja de tener información significativa, el mismo debe ser eliminado. Se puede quietar elementos de dos maneras posibles:

1. Mediante borrado físico: consiste en eliminar efectivamente el elemento del archivo recuperando el espacio que el mismo ocupa. Tiene la ventaja que el archivo ocupa en todo momento el espacio estrictamente necesario en el disco rígido. Y tiene la desventaja que ante el borrado de cada elemento se debe efectuar un "corrimiento" de los elementos posteriores al que se desea quitar.
2. Mediante borrado lógico: consiste en indicar que la zona que ocupa el registro que se quiere eliminar, se encuentra disponible para que cualquier otro elemento la ocupe. La ventaja del método es su velocidad de procesamiento. La desventaja radica en que el espacio sobre el disco rígido no se recupera, solo puede ser reutilizado por el mismo archivo.