Informática: Es la ciencia (Relaciona con una metodología fundamentada y racional para el estudio y resolución de problemas) que estudia el análisis y resolución de problemas (Uso de herramientas informáticas) utilizando computadoras (Maquina 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).

- *Objetivo*: Resolver problemas del mundo real utilizando la computadora.

PASOS PARA TRABAJAR:

1. Poseer un problema.
2. A partir del problema, obtener un modelo del mismo. MODELIZACION.
3. Pasar del modelo del problema a la resolución. MODULARIZACION.
4. Basándose en los módulos propuestos IMPLEMENTAR un PROGRAMA.
5. Utilizar la computadora para la resolución.

Una vez que se tiene la descomposición en funciones/procesos o módulos, debemos diseñar su implementación: requiere escribir el programa y elegir los datos a representar.

PROGRAMA = ALGORITMO + DATOS

PROGRAMA= Las instrucciones (que también se han denominado acciones) representan las operaciones que ejecutará la computadora al interpretar el programa. Un conjunto de instrucciones forma un algoritmo **+** Los datos son los valores de información de los que se necesita disponer y en ocasiones transformar para ejecutar la función del programa.

ALGORITMO:

- Alcanzar el resultado en tiempo finito (un algoritmo comienza y termina). Implícito el número de instrucciones debe ser también finito.
- Especificación rigurosa (algoritmo expresado en forma clara y unívoca).
- Escribir el algoritmo en un lenguaje “entendible” y ejecutable por la máquina.

DATO:

- Es una representación de un objeto del mundo real mediante la cual podemos modelizar aspectos del problema que se quiere resolver con un programa sobre una computadora. Puede ser constante o variable.

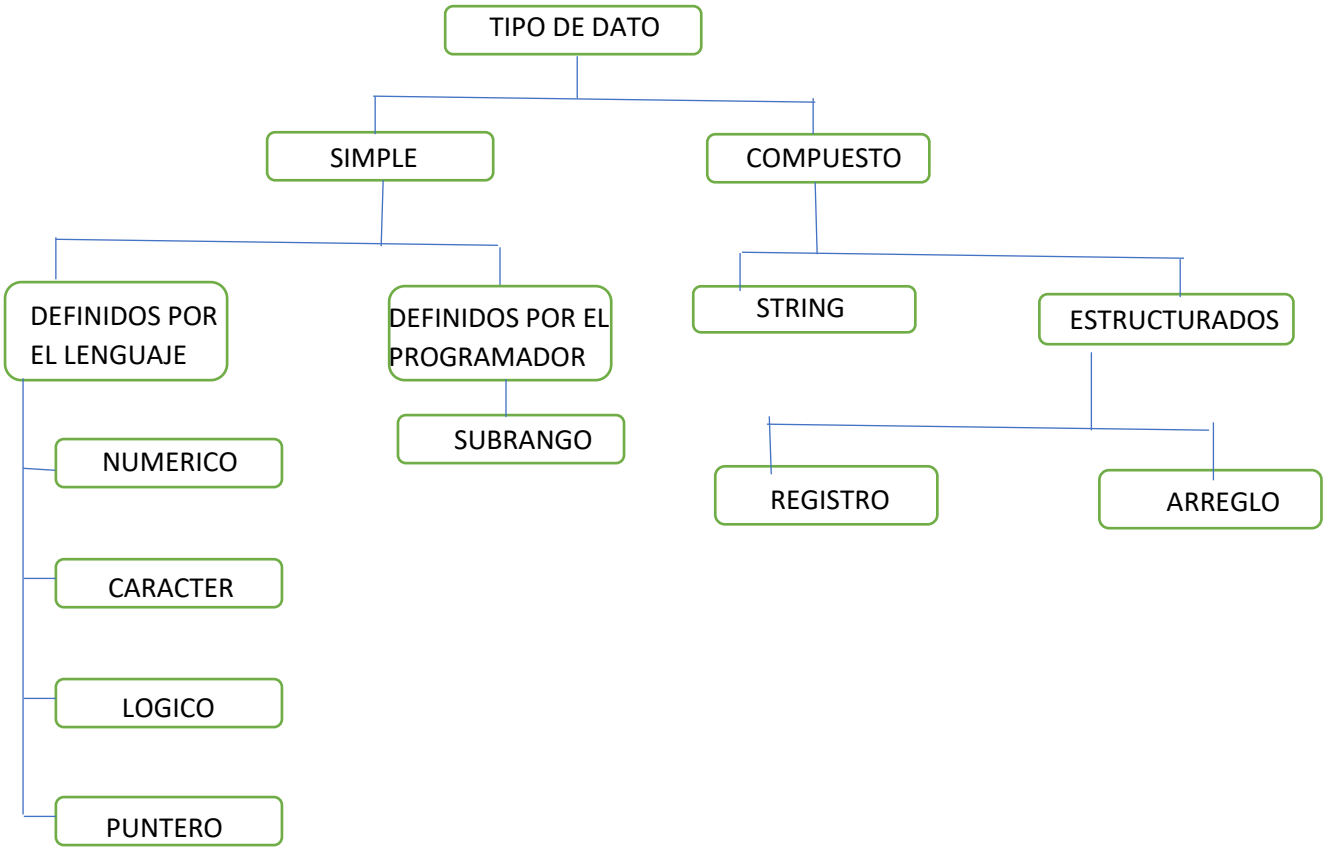
Características para el desarrollador:

- Operatividad: El programa debe realizar la función para la que fue concebido.
- Legibilidad: El código fuente de un programa debe ser fácil de leer y entender. Esto obliga a acompañar a las instrucciones con comentarios adecuados.
- Organización: El código de un programa debe estar descompuesto en módulos que cumplan las subfunciones del sistema.
- Documentados: Todo el proceso de análisis y diseño del problema y su solución debe estar documentado mediante texto y/o gráficos para favorecer la comprensión, la modificación y la adaptación a nuevas funciones.

TIPO DE DATO

- Clase de objeto de datos ligados a un conjunto de operaciones para crearlos y manipularlos.
- Posee un rango de valores posibles.
- Un conjunto de operación interna dentro de la computadora.

Clasificación: SIMPLES (aquellos que toman un único valor, en un momento determinado, de todos los permitidos para ese tipo) y COMPUESTOS (pueden tomar varios valores a la vez que guardan alguna relación lógica entre ellos, bajo un único nombre).



DEFINIDOS POR EL LENGUAJE

- Son provistos por el lenguaje y tanto la representación como sus operaciones y valores son reservadas al mismo.

DEFINIDOS POR EL PROGRAMADOR

- Permiten definir nuevos tipos de datos a partir de los tipos simples.

NUMERICO

- Representa el conjunto de números que se pueden necesitar. Estos números pueden ser enteros (más simples) o reales (números que pueden requerir decimales).

LOGICO

- Permite representar datos que pueden tomar dos valores V o F.

CARÁCTER

- Representa un conjunto finito y ordenado de caracteres que la computadora reconoce. Contiene un solo carácter.

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

Una vez declarado un tipo podemos asociar al mismo ***variables***, es decir *nombres simbólicos que pueden tomar los valores característicos del tipo*.

VARIABLE

- Es una zona de memoria y el contenido va a ser algún tipo de dato. Referenciada la zona con el nombre de la variable. Puede cambiar su valor durante el programa.

CONSTANTE

- Es una zona de memoria y el contenido va a ser algún tipo de dato. Referenciada la zona con el nombre de la variable. NO puede cambiar su valor durante el programa.

CLASIFICACION DE LOS LENGUAJES DE ACUERDO A LA DECLARACION DE LAS VARIABLES

Fuertemente Tipado (strongly typed)

- 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.

Auto Tipados (self typed)

- Clase de lenguaje, que verifica el tipo de las variables según su nombre.

Dinámicamente Tipados (dinamically typed)

- Clase de lenguaje que permiten que una variable tome valores de distinto tipo durante la ejecución de un programa.

PRE-CONDICION

- Es la información que se conoce como verdadera antes de iniciar el programa (ó módulo).

POST-CONDICION

- Es la información que debería ser verdadera al concluir el programa (ó módulo), si se cumplen adecuadamente los pasos especificados.

OPERACIONES DE ENTRADA/SALIDA

- READ: Tomar datos desde un dispositivo de entrada (por defecto desde teclado) y asignarlos a las variables correspondientes.
- WRITE: Mostrar el contenido de una variable, por defecto en pantalla. Permite informar un texto, contenido de una variable, la combinación de ambos o el resultado de alguna cuenta.

ESTRUCTURA DE CONTROL

- Permiten especificar y modificar el flujo de ejecución de las instrucciones del algoritmo que se quiere implementar con un conjunto mínimo de instrucciones.

1. SECUENCIA

Representada por una sucesión de operaciones en la que el orden de ejecución coincide con el orden físico de aparición de las instrucciones.

2. DESICIÓN

Tomar decisión en función de los datos del problema, es entre dos alternativas es la que se representa simbólicamente TRUE o FALSE. IF (condición) then

3. SELECCIÓN

Evalúa la variable y a partir de ello realiza una acción. La variable de decisión debe ser de tipo ordinal. Puede haber más de un valor en cada una de las entradas. Un valor puede aparecer en una sola de las entradas. Deben incluirse todas las posibilidades.

CASE car OF

‘a’..’z’: write(“El carácter leído es una minúscula”);

‘0’..’9’: write(“El carácter leído es un número”);

Else erite(“El carácter leído es un símbolo”);

End.

4. REPETICION

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. La variable de control debe ser de tipo ordinal. No debe modificarse dentro del lazo. Al terminar el ciclo, la variable índice no tiene un valor definido. FOR i:=1 to 5 do begin

5. ITERACIÓN

Consiste en ejecutar un bloque de instrucciones desconociendo el número exacto de veces que se ejecutan. Son estructuras de control iterativas condicionales.

- PRE-CONDICIONAL: Evalúan la condición y si es verdadera se ejecuta el bloque de acciones. Dicho bloque se pueda ejecutar 0, 1 ó más veces. **Importante**: el valor inicial de la condición debe ser conocido o evaluable antes de la evaluación de la condición. WHILE (condición) DO BEGIN
- POST-CONDICIONAL: Ejecutan las acciones luego evalúan la condición y ejecutan las acciones mientras la condición es falsa. Dicho bloque se pueda ejecutar 1 ó más veces. **Importante**: el valor inicial de la condición debe ser conocido o evaluable antes de la evaluación de la condición. REPEAT BEGIN (acción) END UNTIL (condición)

TIPO DE DATOS DEFINIDOS POR EL USUARIO

- Aquel que no existe en la definición del lenguaje, y el programador es el encargado de su especificación. Aumenta la riqueza expresiva del lenguaje con mejor posibilidad de abstracción de datos. Mayor seguridad respecto de las operaciones sobre los datos. Limites preestablecidos sobre los valores posibles que pueden tomar las variables. TYPE identificador = tipo (puede ser estándar o uno definido por el lenguaje)

Ventajas:

- **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.

SUBRANGO

- Tipo ordinal que consiste de una sucesión de valores de un tipo ordinal (predefinido o definido por el usuario) tomado como base. VARIABLE = valor1 .. valor2;

Operaciones permitidas: Asignación. Comparación. Todas las operaciones del tipo base.

Facilita el chequeo de posibles errores, pues permite que el lenguaje verifique si los valores asignados se encuentran dentro del rango establecido. Ayuda al mantenimiento del programa.

STRING

- Sucesión de caracteres de un largo determinado, que se almacena en un área contigua de memoria. Ocupa una cantidad de memoria fija. VARIABLE = string[cant. MaxCaract];

Si las cadenas que se comparan son de igual longitud y contienen los mismos símbolos, en el mismo orden, el resultado de la operación es verdadero. Si tienen distinta longitud el resultado de la comparación es falso.

MODULARIZACION

- Es dividir un problema en partes funcionalmente independientes, que encapsulen operaciones y datos. Resolver problemas con: Abstracción, Descomposición e Independencia Funcional.

1. Cada subproblema está a un mismo nivel de detalle.
2. Cada subproblema puede resolverse independientemente.
3. Las soluciones de los subproblemas puede combinarse para resolver el problema original.

Metodología:

- Mayor productividad: Al dividir el problema, un equipo de desarrollo puede trabajar simultáneamente en varios módulos incrementando la productividad (reduciendo el tiempo de desarrollo global del sistema).
- Reusabilidad: Posibilidad de utilizar repetidamente el producto de software desarrollado.
- Facilidad de Mantenimiento: La división lógica permite corregir los errores en menor tiempo y disminuye los costos de mantenimiento de los sistemas.
- Facilidad de crecimiento: Modularizar permite disminuir los riesgos y costos de incorporar nuevas prestaciones a un sistema en funcionamiento.
- Legibilidad: Mayor claridad para leer y comprender el código fuente.

MODULO

- Tarea específica bien definida, se comunican entre sí adecuadamente y cooperan para conseguir un objetivo común. Cada módulo encapsula acciones, tareas o funciones. Hay que representar los objetos relevantes del problema a resolver.

PROCEDURE

- Conjunto de instrucciones que realizar una tarea específica y retorna 0, 1 o más. Procedure nombre; var..; begin … end;

FUNCTION

- Conjunto de instrucciones que realizan una tarea específica y retorna un único valor de tipo simple.
  Function nombre: tipo; var:…; begin ….. end;
- Asignando a una variable el valor. Dentro de un IF , WHILE. Dentro de un WRITE.

VARIABLES GLOBALES

- Problemas: Demasiados identificadores. No se especifica la comunicación deseada 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.

- Solución: combinación de ocultamiento de datos (Data Hiding) (los datos exclusivos de un módulo NO deben ser "visibles" o utilizables por los demás módulos) y uso de parámetros (los datos compartidos se deben especificar como parámetros que se trasmiten entre módulos).

PARAMETROS

- Comunicación externa de un módulo con el resto del sistema (puede no existir) se produce a través de datos de entrada y datos de salida.

Por Valor

- Un dato de entrada por valor IN, modulo recibe un valor con el cual realiza operaciones y/o cálculos, pero no producirá ningún cambio ni tampoco tendrá incidencia fuera del módulo. (nombre: tipo)

Por Referencia

- El módulo recibe (IN/OUT) el nombre de una variable (referencia a una dirección) conocida, 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. (var nombre: tipo)

TIPO DE DATO ESTRUCTURADO

- Permite al programador definir un tipo al que se asocian diferentes datos que tienen valores lógicamente relacionados y asociados bajo un nombre único.

ESTRUCTURA DE DATOS

- Es un conjunto de variables (que podrían ser de distinto tipo) relacionadas entre sí y que se puede operar como un todo, bajo un nombre único.

CLASIFICACION – ESTRUCTURA DE DATOS

- ELEMENTOS
  - ***Homogénea***: Todos los datos son del mismo tipo.
  - ***Heterogénea***: Los datos son de tipos diferentes.
- ACCESO
  - ***Secuencial***: Para acceder a un elemento particular se debe respetar un orden predeterminado, ejemplo, pasando por todos los que le preceden por ese orden.
  - ***Directo***: Se puede acceder a un elemento articular, directamente, sin necesidad de pasar por los anteriores a él, ejemplo, referenciando una posición.
- TAMAÑO

- ***Estática***: La cantidad de memoria que ocupa no varía durante la ejecución del programa.
- ***Dinámica***: La ocupación de memoria varia durante la ejecución del programa.

o LINEALIDAD
- ***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***: Si para un elemento dado puede existir 0, 1 o más elementos que le suceden y 0, 1 o más elementos que le preceden.

REGISTRO

- Tipo de dato estructurado que permite agrupar diferentes clases de datos en una estructura única.
- Heterogénea, Estática (los tipos de los campos que la componen se detallan en la definición entonces se sabe cuanto ocupan en la declaración), Campos (Información que conforma al registro, posee un identificador, pueden ser nombrados como variables ordinarias).

ARREGLO

- Un arreglo (ARRAY) es una estructura de datos compuesta que permite acceder a cada componente por una variable índice (Indexada), que da la posición de la componente dentro de la estructura de datos (Acceso Directo). Es Homogenea y Estática (tamaño de la estructura se conoce desde su declaración).
- Es una colección de elementos que se guardan consecutivamente en la memoria y se pueden referenciar a través de un índice.

TIPOS DE ARREGLOS

o Vectores
o Matrices
o Tensores

RECORRIDO

- Consiste en recorrer el vector de manera total o parcial, para realizar algún proceso sobre sus elementos.

a. *Recorrido Total*: implica analizar **todos los elementos** del vector, lo que lleva a recorrer completamente la estructura.
b. *Recorrido Parcial*: implica analizar los elementos del vector, **hasta encontrar** aquel que cumple con lo pedido. Puede ocurrir que se recorra todo el vector.

DIMENSION LOGICA

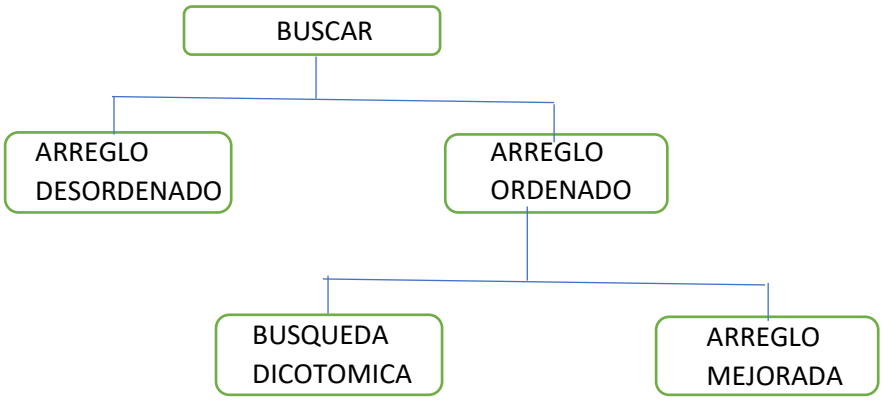
- Se determina cuando se cargan contenidos a los elementos del arreglo. Indica la cantidad de posiciones de memoria ocupadas con contenido real.

DIMENSION FISICA

- Se especifica en el momento de la declaración y determina su ocupación máxima de memoria. La cantidad de memoria no variará durante la ejecución del programa.

ARREGLOS - BUSCAR

Dada una colección de elementos, es posible extraer dos tipos de información:
- Información relativa al conjunto de datos de dicha colección.
- Información detallada de algún ítem en particular.



BUSCAR - ARREGLOS DESORDENADOS

- Se aplica cuando los elementos de la estructura no tienen un orden.
- Requiere excesivo consumo de tiempo en la búsqueda.
- El número de comparaciones es en promedio (dimL + 1) / 2.
- Es indeficiente a medida que el tamaño del arreglo crece.

BUSCAR - ARREGLOS ORDENADOS - BUSQUEDA DICOTOMICA

- Se aplica cuando los elementos de la estructura no tienen orden, caso contrario hay que ordenarlos primero.
- El numero de comparaciones es en promedio (1+LOG2(demL + 1))/2
- A medida que el arreglo crece la cantidad de comparaciones tiene a LOG2(dimL+1)/2

ARREGLOS - ORDENACION

- El proceso por el cual, un grupo de elementos puede ser ordenado se conoce como algoritmo de ordenación.

ARREGLOS - ORDENACION - METODO DE SELECCION

1. En cada pasada se elige un mínimo.
2. Se ubica al mínimo en la posición correspondiente, la primera vez en el primer lugar, la segunda vuelta en el segundo lugar, y así sucesivamente.
3. Utiliza (dimL-1) pasadas, donde n es la dimensión del arreglo.

Supongamos que:

- C es el número de comparaciones
- M es el número de intercambios
- N dimensión del arreglo

SELECCIÓN	INTERCAMBIO	INSERCIÓN
C = [n(n-1)]/2	C = n(n-1)/2	n-1 <= C <= n(n-1)/2
M = 3(n-1)	M = 3n (n-1)/2	2(n-1) <= M <= (n+4).(n-1)/2

TIPO DE DATO PUNTERO

- Un puntero es un tipo de variable usada para almacenar la dirección en memoria de otra variable, en lugar de un dato convencional.
- Mediante la variable de tipo puntero se accede a esa otra variable, almacenada en la dirección de memoria que señala el puntero. Es decir, el valor de la variable de tipo puntero es una dirección de memoria.
- Los punteros pueden apuntar solamente a variables dinámicas, es decir, a datos que están almacenados en memoria dinámica (heap).

Dos tipos de Alocación de Memoria

Alocación Estática (stack) --> variables estáticas

- El espacio de memoria se reserva con anticipación y no cambia durante la ejecución del programa.
- Esto permite una comprobación de tipos en tiempo de compilación

Inconvenientes de la configuración estática

- Su rigidez, ya que estas estructuras no pueden crecer o decrecer durante la ejecución del programa.

Alocación Dinámica (heap) --> variables dinámicas ó referenciadas

- Los espacios de memoria asignados a las variables dinámicas se reservan y se liberan durante la ejecución del programa.

Ventajas de la configuración dinámica:

- Su flexibilidad, ya que las estructuras "dinámicas" pueden crecer o decrecer durante la ejecución del programa.

TYPE TipoPuntero= ^TipoVariableApuntada;

New (p); Read(p^.nombre, p^.apellido, p^.edad, p^.altura);	Dispose (p);

- Copia de valores de un puntero a otro: p^ := q^

LISTA ENLAZADA

- Colección de elementos homogéneos, con una relación lineal que los vincula, es decir que cada elemento tiene un único predecesor (excepto el primero), y un único sucesor (excepto el último).
- Los elementos que la componen no ocupan posiciones secuenciales o contiguas de memoria. Es decir, pueden aparecer dispersos en la memoria, pero mantienen un orden lógico interno.
- Es Homogénea, Dinámica, Lineal y de Acceso Secuencial.

CARACTERISTICAS

- Están compuesta por nodos
- Los nodos se conectan por medio de enlaces o punteros
- Cuando se necesitan agregar nodos a la estructura, se solicita espacio adicional
- Cuando existen nodos que ya no se necesitan, pueden ser borrados, liberando memoria
- Siempre se debe conocer la dirección del primer nodo de la lista (puntero inicial) para acceder a la información de la misma
- El último nodo de la lista se caracteriza por tener su enlace en Nil

LISTAS	LISTAS ENLAZADAS

Program Ej1;
Type
  Lista= ^Nodo;
  Nodo= Record
  Datos: integer; {contenido}
  sig : Lista; {dirección del siguiente nodo}
End;
 	
Var
  L: Lista; {Memoria estática reservada para el puntero inicial}
	Type
	  info = …;
	  Lista = ^ nodo;
	  nodo = record
	  Datos : info;
	  Sig: Lista;
	  End;
	
	Var
	  P : Lista;

LISTA ENLAZADA - RECORRIDO

Type
  cadena50 = string[50];
  persona= record
  nom : cadena50;
  edad : integer
  end;
 	
  lista= ^reg;
  reg = record
  datos : persona;
  sig : lista;
  end ;
	Procedure recorrido (pri : lista);
	Begin
	  while (pri <> NIL) do begin
	  write (pri^.datos.nom,
	  pri^.datos.edad) ;
	  pri:= pri^.sig
	  end;
	end;

LISTA ENLAZADA - BUSQUEDA DE UN ELEMENTO

Type
  cadena50 = string[50];
  persona= record
  nom : cadena50;
  edad : integer
  end;
 	
  lista= ^reg;
  reg = record
  datos : persona;
  sig : lista;
  end ;
	function buscar (pri: lista; x: cadena50): boolean;
	Var
	  encontre : boolean;
	begin
	  encontre := false;
	  while (not encontre) and (pri <> NIL) do
	  if x = pri^.datos.nom then
	  encontre := true
	  else pri := pri^.sig;
	  buscar := encontre
	End;

LISTA ENLAZADA - AGREGAR UN ELEMENTO AL PRINCIPIO DE LA LISTA

```
Program Ej1;
Type
   Lista= ^Nodo;
   Nodo= Record
      Datos: integer;
      Sig: Lista;
   End;

Var
L : Lista;
n : integer;
```

```
Procedure AgregarAdelante (var L:lista; num:integer);
Var
   nue:Lista;
Begin
   New(nue);
   nue^.datos:=num;
   nue^.sig:=L;
   L:=nue;
End;
```

```
Begin {prog. ppal}
   L:=nil;
   readln (n);
   While (n<>0) do Begin
      AgregarAdelante (L, n);
      readln (n);
   End;
End.
```

LISTA ENLAZADA - AGREGAR UN ELEMENTO AL FINAL DE LA LISTA

```
Program Ej1;
Type
   Lista= ^Nodo;
   Nodo= Record
      Datos: integer;
      Sig: Lista;
   End;

Var
L : Lista;
n : integer;
```

```
procedure AgregarAlFinal (var pri: lista; num: integer);
   var act, nue : lista;
begin
   new (nue);
   nue^.datos:= num;
   nue^.sig := NIL;
   if pri <> Nil then begin
      act := pri ;
      while (act^.sig <> NIL ) do
         act := act^.sig ;
         act^.sig := nue ;
   end
   else
         pri:= nue;
end;
```

```
Begin {prog. ppal}
   L:=nil;
   readln (n);
   While (n<>0) do Begin
      AgregarAdelante (L, n);
      readln (n);
   End;
End.
```

LISTA ENLAZADA - BORRAR UN ELEMENTO AL FINAL DE LA LISTA

```
Procedure BorrarElemento ( var pri: lista; num:integer);
   var ant, act: lista;
begin
   act := pri;
   ant := pri;
   {Recorro mientras no se termine la lista y no encuentre el elemento a borrar}
   while (act <> NIL) and (act^.datos <> num) do begin
      ant := act;
      act := act^.sig
   end;
   if (act <> NIL) then begin
      if (act = pri) then
         pri : = act^.sig;
      else     ant^.sig:= act^.sig;
      dispose (act);
   end;
end;
```

LISTA ENLAZADA - INSERTAR UN ELEMENTO EN LA LISTA

```
Procedure InsertarNodo ( var pri: lista; num: integer);
var ant, nue, act: lista;
begin
   new (nue);
   nue^.datos := num;
   act := pri;
   ant := pri;
   {Recorro mientras no se termine la lista y no encuentro la posición correcta}
   while (act<>NIL) and (act^.datos < num) do begin
      ant := act;
      act := act^.sig ;
   end;
   if (ant = act) then
      pri  := nue {el dato va al principio}
   else
      ant^.sig := nue; {el dato va entre otros dos o al final}
   nue^.sig := act ;
end;
```

LISTA ENLAZADA - ANALISIS COMPARATIVO

- Espacio: se refiere a la cantidad de memoria consumida por una estructura de datos dada.

Si se supone igual cantidad de datos en las dos estructuras se puede afirmar que:

- Vectores: son más económicos.
- Listas: requieren espacio extra para los enlaces.

- Tiempo: se refiere al tiempo que toma almacenar o recuperar datos, entre otros.
- Parámetros.

CORRECCION Y EFICIENCIA

CALIDAD DE UN PROGRAMA

- **Corrección (¿Hace lo que se le pide?)** El grado en que una aplicación satisface sus especificaciones y consigue los objetivos encomendados por el cliente.
- **Fiabilidad (¿Lo hace de forma fiable todo el tiempo?)** El grado que se puede esperar que una aplicación lleve a cabo las operaciones especificadas y con la precisión requerida
- **Eficiencia (¿Qué recursos hardware y software necesito?)** La cantidad de recursos hardware y software que necesita una aplicación para realizar las operaciones con los tiempos de respuesta adecuados

- **Integridad (¿Puedo controlar su uso?)** El grado con que puede controlarse el acceso al software o a los datos a personal no autorizado
- **Facilidad de uso (¿Es fácil y cómodo de manejar?)** El esfuerzo requerido para aprender el manejo de una aplicación, trabajar con ella, introducir datos y conseguir resultados
- **Facilidad de mantenimiento (¿Puedo localizar los fallos?)** El esfuerzo requerido para localizar y reparar errores → Se va a vincular con la modularización y con cuestiones de legibilidad y documentación.
- **Flexibilidad (¿Puedo añadir nuevas opciones?)** El esfuerzo requerido para modificar una aplicación en funcionamiento
- **Facilidad de prueba (¿Puedo probar todas las opciones?)** El esfuerzo requerido para probar una aplicación de forma que cumpla con lo especificado en los requisitos
- **Portabilidad (¿Podré usarlo en otra máquina?)** El esfuerzo requerido para transferir la aplicación a otro hardware o sistema operativo
- **Reusabilidad (¿Podré utilizar alguna parte del software en otra aplicación?)** Grado en que las partes de una aplicación pueden utilizarse en otras aplicaciones
- **Interoperabilidad (¿Podrá comunicarse con otras aplicaciones o sistemas informáticos?** El esfuerzo necesario para comunicar la aplicación con otras aplicaciones o sistemas informáticos
- **Legibilidad** : El código fuente de un programa debe ser fácil de leer y entender. Esto obliga a acompañar a las instrucciones con comentarios adecuados. Relacionado con la presentación de documentación.
- **Documentación**: todo el proceso de análisis y diseño del problema y su solución debe estar documentado mediante texto y/o gráficos para favorecer la comprensión, la modificación y la adaptación a nuevas funciones.

CORRECCION

- Un programa es correcto si se realiza de acuerdo a sus especificaciones.
  - Por esta razón es muy importante una completa especificación.

CORRECCION - TESTING

- La técnica de Testing es el proceso mediante el cual se proveen evidencias convincentes respecto a que el programa hace el trabajo esperado.
- Cuando se tiene el plan de pruebas y el programa, el plan debe aplicarse sistemáticamente.

¿Como se proveen evidencias?

- Diseñar un plan de pruebas.
- Poner atención en los casos límite
- Decidir cuales aspectos del programa deben ser testeados y encontrar datos de prueba para cada uno de esos aspectos.
- Diseñar casos de prueba sobre la base de lo que hace el programa y no de lo que se escribió del programa.

- Determinar el resultado que se espera que el programa produzca para cada caso de prueba
- Mejor aún, diseñar casos de prueba antes de que comience la escritura del programa. (Esto asegura que las pruebas no están pensadas a favor del que escribió el programa)

CORRECCION - DEBUGGING

- La técnica de Debugging es el proceso de localización del error.

Puede involucrar:

- el diseño y aplicación de pruebas adicionales para ubicar y conocer la naturaleza del error.
- el agregado de sentencias adicionales en el programa para poder monitorear su comportamiento más cercano.

Los errores pueden provenir de varios caminos, por ejemplo:

- El diseño del programa puede ser defectuoso.
- El programa puede usar un algoritmo defectuoso.

CORRECCION - WALKTHROUGS

- La técnica de Walkthroughs consiste en recorrer el programa ante una audiencia.

- La lectura de un programa a alguna otra persona provee un buen medio para detectar errores.
- Esta persona no comparte preconceptos y está predispuesta a descubrir errores u omisiones.
- A menudo, cuando no se puede detectar un error, el programador trata de probar que no existe, pero mientras lo hace, puede detectar el error, o bien puede que el otro lo encuentre.

EFICIENCIA

- Se define 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 programa, principalmente la memoria utilizada y el tiempo de ejecución empleado.
- Una vez que se obtiene un algoritmo y se comprueba que es correcto, es importante determinar cuántos recursos, como tiempo de ejecución o espacio en memoria, se requiere para la solución del problema.

Medición de la Memoria utilizada en un programa

- Se puede calcular únicamente la cantidad de memoria estática que utiliza el programa.
- Se analizan las variables declaradas y el tipo correspondiente.

Medición del Tiempo de ejecución de un programa

Depende de distintos factores:

- Los datos de entrada al programa
    i) Tamaño
    ii) Contenido
- La calidad del código generado por el compilador utilizado
- La naturaleza y rapidez de las instrucciones de máquina empleadas en la ejecución del programa
- El tiempo del algoritmo base.

El tiempo de ejecución de un programa debe definirse como una función de la cantidad de datos de entrada.

Para algunos programas, el tiempo de ejecución se refiere al tiempo de ejecución del "peor" caso. En estos casos, se obtiene una cota superior del tiempo de ejecución para cualquier entrada.

Ejemplos: Problema de búsqueda secuencial en vectores y listas.

CALCULAR TIEMPO DE EJECUCION - ANALISIS EMPIRICO

- Para realizar un análisis empírico, es necesario ejecutar el programa y medir el tiempo empleado en la ejecución

Inconveniente: este análisis tiene varias limitaciones porque puede dar una información pobre de los recursos consumidos:

- Obtiene valores exactos para una máquina y unos datos determinados
- Es completamente dependiente de la máquina donde se ejecuta
- Requiere implementar el algoritmo y ejecutarlo repetidas veces.

CALCULAR TIEMPO DE EJECUCION - ANALISIS TEORICO

- Para realizar un análisis teórico, es necesario establecer una medida intrínseca de la cantidad de trabajo realizado por el algoritmo. Esto nos permite comparar algoritmos y seleccionar la mejor implementación.

- Obtiene valores aproximados
- Es aplicable en la etapa de diseño de los algoritmos, uno de los aspectos fundamentales a tener en cuenta.
- Se puede aplicar sin necesidad de implementar el algoritmo.
- Independiente de la máquina donde se ejecute
- Permite analizar el comportamiento

Consideraciones generales para tener en cuenta al hacer el cálculo teórico:

- Considerar el número de operaciones elementales que emplea el algoritmo.

- Considerar que una operación elemental utiliza un tiempo constante para su ejecución, independientemente del tipo de dato con el que trabaje.
- Suponer que cada operación elemental se ejecutará en una unidad de tiempo (dejando de lado la magnitud).
- Suponer que una operación elemental es una asignación, una comparación o una operación aritmética simple.

Reglas Generales para el cálculo del tiempo de ejecución

Regla 1: For

Regla 2: For anidados

Regla 3: sentencias consecutivas

Regla 4: If / else

Los comentarios, declaraciones y operaciones de entrada/salida (Read / Write), no se consideran al realizar el cálculo.

EFICIENCIA EN SOLUCIONES MODULARIZADAS

- Si se tiene un programa con módulos, es posible calcular el tiempo de ejecución de los distintos procesos, uno a la vez, partiendo de aquellos que no llaman a otros. Debe haber al menos un módulo con esa característica.
- Después puede evaluarse el tiempo de ejecución de los procesos que llamaron a los módulos anteriores y así sucesivamente.

¿Existe una relación directa entre eficiencia y corrección? ¿Una solución eficiente es siempre correcta? Por qué.

Para que un algoritmo sea eficiente primero se debe haber comprobado que es correcto. Si, no podría ser eficiente sin primero ser correcta.

¿Podrías relacionar el concepto de Eficiencia con los siguientes ítems?

- Cantidad de líneas del programa
- Modularización
- Uso de variables locales
- Uso de determinadas estructuras de control
- Excesiva documentación de los algoritmos

- Cantidad de líneas del programa: No se puede establecer una relación directa
- Modularización: facilita el analisis de eficiencia, dado que puede calcularse el tiempo de ejecución de los distintos procesos uno a la vez, sin embargo una solución modularizada no es necesariamente más eficiente que una sin modularizar
- Las variables locales tienden a mejorar la legibilidad pero no hacen al programa más eficiente
- Las estructuras de control no afectan el tiempo de ejecución
- Un algoritmo bien documentado puede ser menos eficiente que otro

¿Cuáles crees que serían los ítems que influyen sobre el tiempo de ejecución de un programa y por qué?

a. Cantidad de datos de entrada
b. Cantidad de líneas de código
c. Cantidad de iteraciones presentes en el programa
d. Cantidad de variables declaradas

A. Si un programa se va a ejecutar solo con entradas “pequeñas” , la velocidad de crecimiento puede ser menos importante

B. No hay una relación directa

C. Las iteraciones multiplican el tiempo de ejecución del bloque

D. No lo afecta

Análisis comparativo de las estructuras de datos vistas

¿Qué diferencias conceptuales existe en las estructuras de datos vectores y listas?

- Las dos estructuras son homogéneas.
- El acceso a los elementos en el vector es directo a través de un índice, en cambio en la lista es secuencial recorriendo los elementos de la lista
- Las dos estructuras son lineales
- Los vectores almacenan memoria continua en memoria y las listas se almacenan aleatoriamente, siguiendo un orden lógico
- En vectores la ocupación en memoria se resuelve al momento de la compilación y en listas la ocupación en memoria se resuelve al momento de la ejecución
- Los vectores poseen acceso directo, en cambio las listas son de acceso secuencial

Compara y explica detalladamente la operación de inserción y borrado en vectores y listas.

En los vectores al momento de insertar un elemento primero debe comprobarse que haya espacio, entonces se mueven todos los elementos desde la dimensión lógica hasta la posición donde se desea insertar el elemento, finalmente se asigna el elemento a la posición

En las listas no se controla el espacio en memoria, en cambio debe reservarse un lugar de alojado usando new() y generar el nuevo nodo, requiere menos accesos a memoria dado que

al momento de la inserción simplemente se ajustan los punteros del nodo anterior (si no es el primero) y el puntero al siguiente elemento

Al borrar un elemento en un vector debe asignarse a cada elemento su siguiente desde la posición del elemento que deseo borrar hasta la dimensión lógica, finalmente reducir la dimensión lógica

Al borrar un elemento de una lista simplemente deben ajustarse los punteros para que el puntero anterior al nodo borrado (si no es el primero) apunte al nuevo nodo y el nuevo nodo apunte al de la posición actual

¿Cómo diferenciaría la búsqueda de un elemento en un vector y en una lista?

En los vectores hay que controlar que mientras se busca el elemento la dimensión lógica no exceda la dimensión física y avanzar aumentando una variable que sirva de índice

En las listas debe controlarse que no se llegue al último elemento de la lista y se recorre a través de los punteros al siguiente elemento, utilizando una variable auxiliar de tipo “lista”

¿Cómo es el acceso a un elemento conociendo su posición en un vector y en una lista?

En un vector el acceso es directo, vector[pos] mientras que en las listas el acceso es secuencial, es decir deben recorrer los componentes previos para llegar a la posición deseada

¿A igual cantidad de elementos en un vector y una lista qué diferencias hay respecto de la ocupación de memoria?

La diferencia es a favor de los vectores, ya que estos podrían almacenarse directamente en memoria en cambio en las listas se almacenará un puntero por cada componente además del elemento a guardar.