**Concepto de Informática**

La Informáticaes la ciencia que estudia el análisis y resolución de problemas utilizando computadoras.

***Ciencia*** se relaciona con una metodología fundamentada y racional para el estudio y resolución de los problemas.

La ***resolución de problemas***aplicaciones en áreas muy diferentes tales como biología, comercio, control industrial, administración, robótica, educación, arquitectura, etc.

***Computadora*** máquina digital y sincrónica, con capacidad de cálculo numérico y lógico y de comunicación con el mundo exterior. Ayuda al hombre a realizar tareas repetitivas en menor tiempo y con mayor exactitud.

**Etapas de resolución de un problema por computadora Del Problema a la Solución**

Cuando se resuelve un problema del mundo real utilizando una computadora es necesario atravesar una serie de etapas.

**ANALISIS**

1er etapa: se sintetizan los requerimientos del problema y se simplifica el contexto y los datos a utilizar por el programa en la computadora.

**DISENO**

2da etapa: la descomposición funcional nos ayudará a reducir la complejidad, a distribuir el trabajo y en el futuro a reutilizar los módulos. Algoritmos.

**IMPLEMENTACION**

3er etapa: escribir algoritmos en un lenguaje de programación y elegir la representación de los datos.

**VERIFICACION**

4ta etapa: verificar que el programa conduce al resultado deseado, utilizando datos representativos del problema real

|  |  |
| --- | --- |
| ***Análisis y Diseño***  (*NO dependen del lenguaje*)  Entender el problema  Modelizarlo.  Modularizar.  Escribir los algoritmos. | ***Implementación***  (*depende del lenguaje*)  Codificación de los algoritmos en un lenguaje de programación.  Prueba en ejecución de cada módulo.  Prueba general del programa y verificación de que cumple los requisitos del problema. |

**Concepto de Algoritmo**

Definiremos algoritmo como la especificación rigurosa de la secuencia de pasos (instrucciones) a realizar sobre un autómata para alcanzar un resultado deseado en un tiempo finito.

**Concepto de Programa**

Definiremos programa como el conjunto de instrucciones u órdenes ejecutables sobre una computadora, que permite cumplir con una función específica (dichas órdenes están expresadas en un lenguaje de programación concreto).

**Programación Imperativa**

El modelo que siguen los lenguajes de programación para DEFINIR y OPERAR la información, permite asociarlos a un paradigma de programación particular.

En esta primera parte del curso trabajaremos bajo el paradigma imperativo/procedural.

Utilizaremos el lenguaje de programación: PASCAL

**Concepto de Estructura de control**

Todos los lenguajes de programación tienen un conjunto mínimo de instrucciones que permiten especificar el control del algoritmo que se quiere implementar.

**¿Para qué nos sirven las estructuras de control?**

Las estructuras de control permiten modificar el flujo de ejecución de las instrucciones de un programa.

SIMPLE IF

SELECCIÓN

MULTIPLE CASE

PRECONDICIONAL WHILE… DO

ITERATIVAS POSTCONDICIONAL REPEAT…UNTIL

REPETITIVA FOR

**CLASIFICACION DE ESTRUCTURAS DE CONTROL**

|  |  |
| --- | --- |
| **Estructura de control: Selección simple** | Puede ocurrir que en un problema real sea necesario elegir una alternativa entre 2 posibles. |
| **Estructura de control: Selección múltiple** | Puede ocurrir que en un problema real sea necesario elegir una alternativa entre varias posibles en función del problema a resolver. |
| **Estructuras de Control Iterativas** | Puede ocurrir que se desee ejecutar un bloque de instrucciones desconociendo el número exacto de veces que se ejecutan.    Para estos casos existen en la mayoría de los lenguajes de programación estructurada las **estructuras de control iterativas condicionales.**  Como su nombre lo indica las acciones se ejecutan dependiendo de la evaluación de la condición.  Estas estructuras se clasifican en **pre-condicionales** y **post-condicionales**. |
| **Estructura de Control: Iterativa precondicional** | Las **estructuras de control iterativas precondicionales** primero evalúan la condición y si es verdadera se ejecuta el bloque de acciones. Dicho bloque se puede ejecutar 0, 1 o más veces. |
| **Estructura de control: Iterativa repetitiva** | Puede ocurrir que se desee ejecutar un bloque de instrucciones conociendo el número exacto de veces que se ejecutan. Es decir ***repetir N veces un bloque de acciones.***  Este número de veces que se deben ejecutar las acciones es fijo y conocido de antemano. |
| **Estructura de Control: Iterativa postcondicional** | Las estructuras de control iterativas postcondicionales primero ejecutan el bloque de acciones y luego evalúan la condición. A diferencia de la estructura iterativa pre-condicional, el bloque de acciones se ejecuta 1 ó más veces.  Importante: el valor inicial de la condición debe ser conocido o evaluable antes de la evaluación de la condición.  Las estructuras de control iterativas postcondicionales primero ejecutan el bloque de acciones y luego evalúan la condición. A diferencia de la estructura iterativa pre-condicional, el bloque de acciones se ejecuta 1 ó más veces. |

**Definición de Tipo de Dato**

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

**Cada tipo de dato se caracteriza por presentar:**

* Un rango de valores posibles
* Un conjunto de operaciones realizables sobre ese tipo
* Una representación interna

***Los tipos de datos simples*** son aquellos que toman un único valor, en un momento determinado, de todos los permitidos para ese tipo. A su vez, los tipos simples, pueden clasificarse en:

* Tipos de datos definidos por el lenguaje (primitivos o estándar) son provistos por el lenguaje y tanto la representación como sus operaciones y valores son reservadas al mismo.
* Tipos definidos por el usuario, permiten definir nuevos tipos de datos a partir de los tipos primitivos.

***Los tipos de datos compuestos*** que son aquellos que pueden tomar varios valores a la vez que guardan alguna relación lógica entre ellos.

**Un tipo de dato string** es una sucesión de caracteres de longitud determinada.

**El tipo de dato registro** (record) permite agrupar un conjunto de campos, con igual o diferente tipo de dato, bajo un nombre único.

***Definidos por el lenguaje o estándar*** - esto significa que el conjunto de valores de ese tipo, las operaciones que se pueden efectuar y su representación están definidas y acotadas por el lenguaje

**El tipo de dato entero** es un tipo de dato simple y ordinal.

**El tipo de dato real** es una clase de dato numérico que permite representar números decimales. Es un tipo de dato simple.

**El tipo de dato lógico** permite representar datos que pueden tomar solamente uno de dos valores. Este tipo de dato se conoce también como tipo de dato boolean. Es un tipo de dato simple y ordinal.

**El tipo de dato carácter** es un tipo de dato simple y ordinal.

**El tipo de dato puntero** es un tipo de variable usada para almacenar la dirección en memoria dinámica de otra variable, en lugar de un dato convencional.

***Definido por el usuario*** es aquel que no existe en la definición del lenguaje, y el programador es el encargado de su especificación.

**Sintéticamente entonces un Tipo significa una clase de datos que tiene asociado:**

* + Un rango de valores posibles.
  + Una forma de representación.
  + Un conjunto de operaciones permitidas.
  + Un conjunto de condiciones de valores permitidos que se pueden verificar.

**Un tipo de dato subrango** es un tipo simple y ordinal que consiste de una sucesión de valores extraídos de un tipo ordinal base.

**Definicion de Modularizacion**

La tarea de Modularizar implica dividir un problema en partes. Se busca que cada parte realice una tarea simple y pueda resolverse de manera independiente a las otras tareas.

**PROCEDURE**: Conjunto de instrucciones que realiza una tarea específica y como resultado puede retornar 0, 1 o más valores. Puede ocurrir que un módulo Procedure contenga, además, de la declaración de tipos y variables propias, la definición de otros módulos. Se dice que el procedimiento contiene módulos anidados.

**FUNCTION**: Conjunto de instrucciones que realiza una tarea específica y como resultado retorna un único valor de tipo simple. También puede ocurrir que un módulo function contenga además, de la declaración de tipos y variables propias, la definición de otros módulos procedimientos y/o funciones.

**Variable global:** su declaración se hace en la sección de declaración del programa principal, es decir fuera de todos los módulos del programa y podrá ser usada en el programa y en todos los módulos del mismo. Por lo tanto, podrían ser utilizadas para la comunicación entre el programa y los módulos.

**Variable local al módulo:** su declaración se hace en un módulo particular y sólo podrá ser usada por ese módulo. Si este módulo contiene a su vez otros módulos, entonces esa variable podría ser también usada por todos los módulos interiores, si está declarada previo a ellos.

**Variable local al programa:** su declaración se hace antes de la sección de instrucciones ejecutables del programa y después de la declaración de los módulos del programa. Su uso se limita a la sección de instrucciones ejecutables.

**Parámetros:** El parámetro se define como una variable que representa un dato compartido entre módulos o entre un módulo y el programa principal. El uso de parámetros significa que los datos compartidos se deben especificar como parámetros que se trasmiten entre módulos. **Los datos compartidos se declararán como parámetros.**

|  |  |
| --- | --- |
| **POR VALOR:** Un parámetro por valor es un dato de entrada que significa que el módulo recibe una copia de un valor proveniente de otro módulo o del programa principal.  Con este dato el módulo puede realizar operaciones y/o cálculos, pero fuera del módulo ese dato NO reflejará cambios. | **POR REFERENCIA:** Un parámetro por referencia es un dato de salida o entrada/salida que contiene la dirección de memoria donde se encuentra la información compartida con otro módulo o programa que lo invoca.  El módulo que recibe este parámetro puede operar con la información que se encuentra en la dirección de memoria compartida y las modificaciones que se produzcan se reflejarán en los demás módulos que conocen esa dirección de memoria compartida. |
| datos de entrada al módulo | datos de salida  datos de entrada y salida del módulo |

**Desventajas de la comunicación a través de variables globales**

* Posibilidad de perder integridad de los datos, al modificar involuntariamente en un módulo alguna variable que luego deberá utilizar otro módulo. ¿Independencia? ¿Reusabilidad?
* Dificultad durante la etapa de la verificación
* Demasiadas variables en la sección de declaración
* Posibilidad de conflicto entre los nombres de las variables utilizadas por diferentes programadores
* Falta de especificación del tipo de comunicación entre los módulos
* Uso de memoria

|  |  |
| --- | --- |
| Variables Globales | Parametros |
| * Posibilidad de perder integridad de los datos, al modificar involuntariamente en un módulo alguna variable que luego deberá utilizar otro módulo * Posibilidad de conflicto entre los nombres de las variables utilizadas por diferentes programadores * Dificultad durante la etapa de la verificación * Demasiadas variables en la sección de declaración * Falta de especificación del tipo de comunicación entre los módulos * Uso de memoria | * Los módulos no pueden afectar involuntariamente los datos de otros módulos. ¡Independencia! ¡Reusabilidad! * Se reduce significativamente la cantidad de variables globales * Permite distinguir el tipo de comunicación * Los parámetros por referencia no utilizan memoria local * El uso de parámetros por valor puede significar una importante utilización de memoria local |

**Una estructura de datos** es un conjunto de variables (que podrían ser de distintos tipo) que poseen una relación lógica o conceptual entre sí y que se puede reconocer como un todo, bajo un nombre único.

**ESTRUCTURA DE DATOS – CLASIFICACIÓN**

Según sus elementos:

**Homogénea**: todos los elementos que la componen pertenecen a mismo tipo de dato.

**Heterogénea**: los elementos que la componen pueden ser de distinto tipo de dato.

Según su ocupación en memoria:

**Estática**: la cantidad de elementos que puede contener no varía durante el tiempo de ejecución del programa. Por lo tanto, la cantidad de memoria que necesita ocupar es siempre la misma durante todo el programa.

**Dinámica**: la cantidad de elementos puede variar, por lo tanto, puede variar la memoria ocupada en tiempo de ejecución.

Según su acceso:

**Directo:** se puede acceder a un elemento particular sin necesidad de pasar por otros elementos.

**Secuencial**: para llegar a un elemento, puede ser necesario pasar por otros elementos.

Según su linealidad:

**Lineal**: cada elemento puede tener 0 o 1 sucesor, 0 o 1 predecesor o ambos.

**No lineal**: cada elemento puede tener 0, 1 o más elementos que le preceden, 0, 1 o más elementos que le suceden o ambos casos.

**ESTRUCTURA DE DATOS REGISTRO - CARACTERÍSTICAS**

Según sus elementos es ***heterogénea***.

Cada campo que contiene el registro puede ser del mismo o de distinto tipo de dato.

Según el acceso a sus elementos es de ***acceso directo***.

Cada campo se puede referenciar directamente sin necesidad de referenciar los otros.

Según su ocupación en memoria es ***estática.***

Al declarar una variable de tipo registro, su tamaño en memoria puede ser determinado realizando la suma de sus campos, algo que no varia en tiempo de ejecución.

Según la relación entre sus elementos.

En esta estructura de datos podemos decir que no aplica el concepto.

No existe una relación de orden entre los campos, solamente el concepto que juntos componen.

**ESTRUCTURA DE DATOS REGISTRO - OPERACIONES**

Operaciones sobre la variable del tipo registro (***record***).

* + Asignación (**:=**)
  + Acceso a cada campo en particular (**.**)

Operaciones sobre los campos que componen el registro (***record***).

* + Como cada campo del registro puede tener su propio tipo de dato, entonces las operaciones que se puedan realizar sobre éstos campos son las permitidas para el tipo de dato que posea el campo correspondiente.

Única operación permitida sobre la variable del tipo registro (record).

* + Asignación (:=)

**ARREGLOS / ARRAYS**

Un arreglo es una estructura de datos que permite acceder a cada componente a través de índices, que indican la posición de cada componente dentro de la estructura de datos.

Un tipo de dato **Arreglo** es una colección de elementos que se guardan consecutivamente en la memoria y se pueden referenciar a través de índices.

**Características**

Esta estructura de datos reúne las siguientes características:

* Todos los elementos son del mismo tipo de datos, por eso es una estructura de datos **homogénea**.
* Los elementos o componentes pueden recuperarse en cualquier orden, indicando simplemente su posición, por eso es una estructura de datos de **acceso directo**. Como el acceso se hace a través del índice se la denomina también indexada.
* La memoria ocupada durante la ejecución del programa es fija, por eso se dice que es una estructura de datos **estática**.
* Dado que cada elemento tiene un elemento que le precede y uno que le sigue, esta estructura se denomina **lineal**.

**Tipos**

* Arreglos unidimensionales: vectores (los usaremos en este curso)
* Arreglos bidimensionales: matrices

**Operaciones**

* + Asignación de contenido a un elemento
  + Lectura / Escritura
  + Recorridos
  + Cargar datos en un vector
  + Agregar elementos al final
  + Insertar elementos
  + Borrar elementos
  + Buscar un elemento

|  |  |
| --- | --- |
| **ALOCACION EN MEMORIA**   * Las estructuras de datos hasta ahora vistas se almacenan estáticamente en la memoria física del ordenador. * 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 | **Alocación Dinámica (Heap)**   * Los espacios de memoria asignados a las variables dinámicas se reservan y se liberan durante la ejecución del programa. * No hay espacio de memoria reservado |
| **Inconvenientes de la configuración estática**   * Su rigidez, ya que estas estructuras no pueden crecer o decrecer 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**.** |

**TIPO DE DATO PUNTERO**

* Un puntero es un tipo de variable usada para almacenar la dirección en memoria dinámica de otra variable, en lugar de un dato convencional.
* Mediante la variable de tipo puntero (en stack) se accede a esa otra variable, almacenada en la dirección de memoria dinámica que señala el puntero. Es decir, el valor de la variable de tipo puntero es una dirección de memoria.
* Se dice que el puntero apunta o señala a la variable almacenada en la dirección de memoria (heap) que contiene el puntero. Lo que nos interesa es el dato contenido en esa variable apuntada. No hay que confundir la variable apuntada con el puntero.
* Cada variable de tipo puntero puede apuntar a un único tipo de dato (en Heap).

**Operaciones**

* Las variables dinámicas son por definición aquellas que se crean cuando se necesitan y se destruyen cuando ya han cumplido con su cometido.

**New (puntero)** creación

**Dispose (puntero)** el efecto es que se “rompe” el enlace entre p y p^.

* Asignación de un valor a una variable puntero
* Asignación de valor al objeto “referenciado” o “apuntado” por el puntero
* Acceso a la información del objeto “referenciado” o “apuntado” por el puntero
* Operaciones de comparación
* Eliminación de un objeto apuntado que no se necesita

**TIPO DE DATO LISTA – CONCEPTO**

* 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.

**COMPARACION ARRAGLOS VS LISTAS**

|  |  |
| --- | --- |
| **ARREGLOS** | **LISTAS** |
| HOMOGENEA | HOMOGENEA |
| ESTATICA | DINAMICA |
| LINEAL | LINEAL |
| ACCESO DIRECTO | ACCESO SECUENCIAL |
| * Los arreglos se almacenan en memoria estática. * La ocupación de memoria se resuelve en tiempo de compilación. * Ocupan posiciones consecutivas de memoria a partir de la posición inicial. | * Las listas se almacenan en memoria dinámica. * La ocupación de memoria se resuelve en tiempo de ejecución. * Se disponen aleatoriamente en memoria. Se relacionan lógicamente. |

**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

**OPERACIONES**

* Recorrer una lista
* Buscar un elemento en la lista
* Crear una lista vacía
* Agregar un elemento al principio de una lista
* Agregar un elemento al final de una lista
* Eliminar un elemento de la lista
* Insertar un nuevo elemento en una lista ordenada

**CORRECCION Y EFICIENCIA**

**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

**Legibilidad** (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 DE PROGRAMAS**

Un programa es correcto si cumple con las especificaciones del problema a resolver. Por esta razón, es que la especificación debe ser completa, precisa y no ambigua.

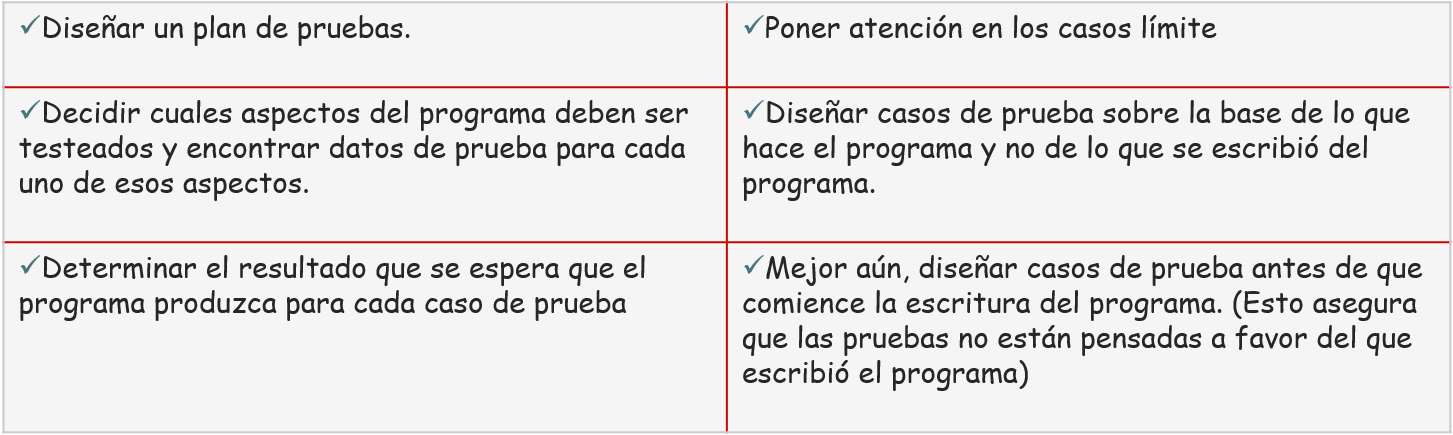
Para medir la corrección de un programa, el programador cuenta con diferentes técnicas:

* + - * + **TESTING**
        + **WALKTHROUGH**
        + **DEBUGGING**

Estas técnicas usadas complementariamente proveen evidencias para la corrección.

**Técnica de Testing**

La técnica de **Testing** es el proceso mediante el cual se proveen evidencias convincentes respecto a que el programa hace el trabajo esperado.

*  Las precondiciones, junto con las postcondiciones, permiten describir la función que realiza un programa, sin especificar un algoritmo determinado.
* Las precondiciones describen los aspectos que se consideran verdaderos antes que el programa comience a ejecutarse, por ejemplo: entradas de datos disponibles.
* Las postcondiciones describen los aspectos que deben cumplirse cuando el programa terminó.

**Técnica de Debugging**

La técnica de **Debugging** es el proceso mediante el cual se pueden identificar y corregir errores

**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.

**Técnica de Walkthroughs**

* 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 de Programas**

* Para cada problema se pueden tener varias soluciones algorítmicas correctas,
* Sin embargo, el uso de recursos (tiempo, memoria) de cada una de esas soluciones puede ser muy diferente

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.

**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
  + Tamaño
  + 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

El tiempo de ejecución de un programa puede calcularse de dos maneras:

* + - * + Análisis Empírico
        + Análisis Teórico

|  |  |
| --- | --- |
| Análisis Empírico | Análisis Teórico |
| 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. | 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.  **Ventajas**   * + 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. |

FOR = (suma de lo de adentro \* iteraciones) + 3 \* iteraciones + 2

WHILE = (suma de adentro x n) + n + 1

IF = (se suma de adentro + condiciones del if)