***Etapas para la resolución de un problema mediante la computadora:***

Dado un problema de la vida real:

* 1era etapa 🡪 Análisis: se sintetizan los requerimientos del problema, y se crea un Modelo que simplifique el contexto del problema y de los datos a representar.
* 2da etapa 🡪 Diseño: A partir del análisis se diseña una solución, descomponiendo funcionalmente.
* 3era etapa 🡪 Implementación: se escriben los algoritmos en un lenguaje de programación y se elige la representación de datos.
* 4ta etapa 🡪 Verificación: implica estudiar si el sistema cumple con los requerimientos especificados.

Análisis y Diseño, No dependen del lenguaje, sirven para entender el programa, modularizarlo y escribir los algoritmos.

Implementación y Verificación, dependen del lenguaje, sirven para codificar los algoritmos en un lenguaje, probar en ejecución a cada módulo y prueba general del programa, verificando si cumple con lo pedido.

Programa: Es un conjunto de instrucciones u ordenes ejecutables sobre una computadora, que cumplen una tarea especifica. 🡪 Programa = Algoritmo + Datos.

Algoritmo: Es una secuencia de instrucciones que realizan una tarea en un tiempo finito.

Datos: Es una representación de un objeto, aspecto o situación del mundo real, los cuales se modelizan a programas, pueden ser constantes o variables.

***Tipos de Datos:***

Un tipo de dato es una clase de objetos de datos ligados a un conjunto de operaciones para crearlos y manipularlos; se caracterizan por tener un rango de valores posibles, un conjunto de operaciones realizables sobre el tipo y una representación interna. Y se separan en dos tipos:

* Tipos de datos definidos por el Lenguaje: Son provistos por el lenguaje, su representación y operaciones son reservados al mismo.
* Tipos de Datos Definidos por el Usuario: Permiten definir nuevos tipos de datos a partir de tipos simples.
* Tipos de Datos Simples: En un momento dado de la ejecución pueden contener solo un valor posible.
* Tipo de Datos Ordinales: Para cada valor de los posibles para el tipo, se puede decir cuál es su inmediato anterior y posterior.
* Tipo de Datos Numéricos: es un conjunto de valores que puede representarse de dos formas, (Tipo Entero, Tipo Real)

-El entero es un tipo de dato simple y Ordinal, dependiendo del compilador y de la maquina donde se corra, tiene un rango de [-; ]. Operaciones (+, -, \*, div, mod, operaciones relacionales)

-El Real permite representar números decimales, los elementos de este tipo se representan con cierta cantidad de bits para la parte entera y otra para la decimal. Operaciones (+, -, \*, /, div, mod, operaciones relacionales)

* Tipo de Datos Lógicos: permiten representar datos que pueden tomar solo uno de dos valores (Boolean), se utiliza en situaciones donde se representan dos alternativas de una condición. Los operadores lógicos son (Not, And, Or).
* Tipos de Datos Carácter: representan elementos de un conjunto finito ordenado de caracteres que la computadora reconoce. Es de tipo simple y ordinal mediante la tabla ASCII. Admiten operaciones de comparación (=, <>, >=)

***Variables:***

Una variable es una zona de memoria cuyo contenido será un tipo de datos, su dirección inicial está asociada al nombre de la variable.

Su valor puede cambiar durante la ejecución del programa, se indica un valor en el programa, se declaran con la palabra clave Var.

Contrario a esto, Las constantes no cambian su valor durante la ejecución del programa, se debe indicar un valor en su declaración, se declaran con la palabra clave Const.

***Tipos de Lenguajes:***

Fuertemente Tipados, exigen especificación de los tipos de datos de cada variable y verifican que sea un tipo correspondiente a su definición en la ejecución.

Auto Tipados, determinan el tipo acorde al primer valor que se le asigne a la variable.

Dinámicamente Tipados, permiten que una variable tome valores de distinto tipo durante la ejecución de un programa.

***Estructuras de Control:***

Permiten cambiar el flujo de ejecución de un programa

* Secuencia: es la forma de ejecución más simple, está representada por una sucesión de operaciones, en la que el orden de ejecución coincide con el orden físico de aparición de instrucciones. (Read, Write)
* Decisión: se usa cuando es necesario tomar decisiones en función de los datos del problema. (If, Else)
* Selección: si tengo que ver qué valor toma un determinado dato entre varios, dependiendo de ese valor puedo realizar diferentes acciones. (Case Of)
* Repetición: es una extensión natural de la secuencia, consiste en repetir un número fijo y conocido de veces una o más acciones. (For do)
* Iteración: Puede ocurrir que se desee ejecutar un bloque de instrucciones un número desconocido de veces, existen estructuras iterativas Pre-condicionales (evalúan la condición y luego ejecutan el bloque) y Post-condicionales (Ejecutan el bloque de acciones y luego evalúan la condición). (While do, Repeat Until)

Case: Permite realizar una o más acciones dependiendo de cuál de todas las condiciones evaluadas es verdadera.

* La variable de decisión debe ser de tipo ordinal
* Puede haber más de un valor en cada una de las entradas
* Un valor debe aparecer en una sola de las entradas
* Deben incluirse todas las posibilidades.

For: Permite repetir un numero conocido de veces un bloque de acciones

* La variable de control “i” debe ser de tipo ordinal (Integer, boolean, Char)
* No debe modificarse dentro del bloque de acciones
* Los incrementos o decrementos y testeos son implícitos
* Al terminar el ciclo, la variable índice no tiene un valor definido (Su uso se limita a la repetición).

While: Evalúan la condición y luego ejecutan el bloque de acciones en caso de que sea verdadera.

* Realiza las acciones mientras la condición sea verdadera
* Se ejecuta 0,1 o más veces.

Repeat Until: ejecutan el bloque de acciones y luego evalúan la condición.

* Realiza acciones mientras la condición es falsa (o hasta que la condiciones se hace verdadera)
* Se ejecuta 1 o más veces (como mínimo 1 vez)

***Modularización***: ya que los problemas del mundo real implican: Complejidad, Extensión, y Modificaciones/Situaciones de cambio. Los tratamos de resolver con: Abstracción y Descomposición funcional.

Modularizar significa dividir un problema es partes funcionalmente independientes, que encapsulen operaciones y datos.

Abstracción: La descomposición tiene siempre como objetivo obtener:

Alta Cohesión 🡪 medida del grado de identificación de un módulo con una función concreta

Bajo Acoplamiento 🡪 medida de la interacción de los módulos que constituyen un programa.

Cuando se descompone un problema en subproblemas, es de la forma tal que:

* Cada subproblema está en un mismo nivel de detalle.
* Cada subproblema puede resolverse lo más independiente posible.
* Las soluciones de los subproblemas pueden combinarse para resolver el problema original.

Los módulos son un conjunto de instrucciones que cumplen una tarea especifica bien definida, se comunican entre si adecuadamente y cooperan para conseguir un objetivo común.

***Ventajas de la modularización:***

* Mayor Productividad: Al dividir en módulos independientes, un equipo de desarrollo puede trabajar simultáneamente en varios módulos, incrementando la productividad.
* Reusabilidad: Hace posible utilizar repetidamente el producto desarrollado. Naturalmente la descomposición funcional que ofrece la modularización favorece el reuso.
* Facilidad de Mantenimiento Correctivo: La división lógica 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 del sistema.
* Facilidad para el crecimiento del sistema: La modularización permite disminuir los riesgos y los costos de incorporar nuevas prestaciones a un sistema en funcionamiento.
* Mayor Legibilidad: Un efecto de la modularización es una mayor claridad para leer y comprender el código fuente. Es más fácil comprender un número limitado de instrucciones directamente relacionada.

***Módulos:*** Se separan en Procedimientos y Funciones, y cada uno tiene:

* Un encabezado que permite mostrar qué hace.
* Un cuerpo que oculta cómo resuelve su tarea.
* Datos propios (Locales) y Datos compartidos.

PROCEDURES:

1. Es un módulo que realiza tareas y puede devolver 0,1 o más valores.
2. Se invocan escribiendo su nombre.
3. En general devuelve el resultado a través de Parámetros.
4. No es aconsejable introducir operaciones de Lectura/Escritura.
5. Permite parámetros de entrada y de entrada salida.

FUNCTION:

1. Es un módulo que realiza una única tarea y devuelve siempre un solo valor de tipo simple.
2. Para devolver el resultado se asigna al nombre de la función como una instrucción.
3. No es aconsejable introducir operaciones de Lectura/Escritura.
4. Se puede invocar dentro de una condición (If / While), o asignarla a una variable, o dentro de un write.
5. Pueden recibir solo parámetros de entrada.

Flujo de Control una vez ejecutado un módulo: Tanto en procedimientos como en funciones, el flujo de control del programa vuelve al punto inmediatamente después de donde se realizó la llamada una vez que se ha ejecutado el procedimiento o la función. La diferencia es que las funciones pueden retornar un valor, mientras que los procedimientos no lo hacen.

***Alcance de una variable:*** es el contexto de donde una variable puede ser referenciada o nombrada y ésta es reconocida.

Variable Global: Su declaración se hace en la sección de declaración del programa principal (Fuera de todos los módulos del programa) y podrá ser usada en el programa y en todos los módulos de este.

Variable Local: Su declaración se hace en un módulo particular y solo podrá ser usada por ese modulo. Si este módulo contiene a su vez otros módulos, entonces esa variable puede ser también usada por todos los módulos interiores.

***Comunicación entre módulos:*** Los datos entre los módulos y el programa se comunican mediante:

* Variables Globales: No es Aconsejable. Puede traer efectos colaterales, perjudica la legibilidad.
* Parámetros: Indican de manera explícita que datos utilizara el módulo.

La comunicación externa de un módulo con el resto del sistema es aconsejable que se produzca a través de datos de entrada y datos de salida a través de parámetros.

PARÁMETROS POR VALOR:

* Un dato de entrada por valor es llamado parámetro IN y significa que el módulo recibe (Sobre una variable local) un valor proveniente de otro modulo (o del programa principal).
* Con él se pueden realizar operaciones y cálculos, pero no producirá ningún cambio ni tampoco tendrá incidencia fuera del módulo.

PARÁMETROS POR REFERENCIA:

* La comunicación por referencia significa que el módulo recibe la dirección de memoria de una variable conocida en el punto de invocación.
* Dentro del módulo se puede operar con el valor original contenido en esa dirección de memoria, y las modificaciones que se produzcan se reflejan en los demás módulos que conocen a la variable. Por ello no requiere memoria Local.
* En el encabezado del módulo se distinguen por tener la palabra VAR.

***Tipos de datos definidos por el usuario:*** Nos da la capacidad de especificar y manejar datos no estándar, indicando valores permitidos, operaciones validad y su representación interna. Ventajas:

* Aumento de la riqueza expresiva del lenguaje, con mejores posibilidades de abstracción de datos.
* Mayor seguridad respecto de las operaciones que se realizan sobre cada clase de datos.
* Limites preestablecidos sobre los valores posibles que pueden tomar las variables que corresponden al tipo de dato.
* Generan flexibilidad en el caso de ser necesario modificar la forma en la que se representa un dato, solo se debe modificar una declaración (En el Type) en lugar de un conjunto de declaraciones de variables.
* Facilitan la documentación, se pueden usar como identificadores de los tipos nombres autoexplicativos, facilitando el entendimiento y lectura del programa.

SUBRANGO:

* Consiste en una sucesión de valores de un tipo ordinal tomado como base.
* Es un tipo de datos Simple
* Es un tipo de datos Ordinal.

CONJUNTOS:

* Colección de datos simples, cuyo contenido deben ser datos del tipo Ordinal, sin repetición.
* Es un tipo de datos Compuesto.
* No es un tipo de datos Ordinal.
* Se pueden tener conjuntos de valores enteros, boolean y char.
* En pascal, no pueden contener más de 255 elementos.
* No permite operaciones de Lectura – Escritura.
* Permite operaciones de asignación (: =), unión (+), intersección (\*). Pertenencia (IN) y diferencia (-), comparación (<=, <>, =).

STRING:

* Es una sucesión de caracteres de un largo determinado, que se almacenan en un área contigua de la memoria.
* Es un tipo de dato Compuesto.
* No es un dato Simple.
* Si no se indica la longitud máxima, se supone como 255 caracteres como máximo.
* Se pueden comparar tipos de datos string.
* Comparación por igualdad: se evalúan si las longitudes de los strings a comparar son iguales, si es así copara el contenido, si las longitudes no son iguales no se compara el contenido.
* Comparación por distinto: Se evalúa si las longitudes de los string son iguales, si no es así se devuelve verdadero, Si las longitudes son iguales se compara el contenido.
* Comparación por menor: Se utiliza el orden lexicográfico.

***Tipo de dato Estructurado:*** Permite al programador definir un tipo al que se asocian diferentes datos que están lógicamente relacionados y asociados bajo un nombre único.

Una Estructura de datos es un conjunto de variables relacionadas entre sí y que se puede operar como un todo, bajo un nombre único. Se clasifican según:

* Datos que se pueden almacenar en la estructura: el tipo de datos que componen a una estructura de datos pueden ser homogéneos y heterogéneos.
* Homogénea: Los datos que la componen son todos del mismo tipo.
* Heterogénea: Los datos que la componen son de distinto tipo.
* Ocupación de memoria: Las estructuras de datos pueden ocupar memoria Estática y Dinámica.
* Estática: la cantidad de elementos que contiene es fija, es decir que la cantidad de memoria que ocupa no varía durante la ejecución del programa.
* Dinámica: La cantidad de elementos que contiene es variable, y por lo tanto la cantidad de memoria ocupada puede cambiar durante la ejecución del programa.
* Acceso a sus elementos: Las estructuras de datos se puede acceder a sus elementos mediante acceso directo o secuencial.
* Acceso Secuencial: para acceder a un elemento particular se debe respetar un orden predeterminado.
* Acceso Directo: Se puede acceder a un elemento particular, directamente, sin necesidad de pasar por los anteriores a él.
* Organización de sus datos: la estructura de datos puede tener un orden línea o no lineal.
* Lineal: está formada por ninguno, uno o varios elementos que guardan una relación de adyacencia ordenada donde cada elemento le sigue uno y le precede uno, solamente.
* No Lineal: Para un elemento dado pueden existir 0,1 o más elementos que le suceden y 0, 1 o más elementos que le preceden.

REGISTROS:

* Es una estructura de que permite agrupar datos de diversos tipos con alguna clase de correspondencia lógica. Cada uno de estos datos que componen un registro es llamado “Campo”.
* Se pueden realizar operaciones de, asignación (: =), Entrada y Salida (read, write), comparación (=, <>) Todas las operaciones se realiza campo a campo.
* Es de tipo: Heterogénea, Estática, Acceso Directo, Lineal.

ARREGLOS:

* Es una estructura de datos que permite acceder a cada componente a través de una variable índice, que da la posición de la componente dentro de la estructura de datos.
* Los elementos de un arreglo pueden pertenecer a cualquier tipo de datos de alocación estática (Enteros, Reales, Caracteres, Registros, otro arreglo).
* El índice de un arreglo es una variable de tipo ordinal, permite acceder a cualquier elemento de manera directa, representa un desplazamiento desde la posición inicial del arreglo.
* Es de tipo: Homogénea, Estática, Acceso directo indexado, Lineal.
* Cuando se trabaja con esta estructura se debe tener en consideración la dimensión física y lógica del arreglo.
* Existes los arreglos de una dimensión (Vectores) y Bidimensionales (Matrices) las cuales requieren dos índices para acceder al componente deseado.

Alocación estática y dinámica: Mecanismos para la reserva de la memoria.

Alocación estática: Las variables reservan memoria en su declaración, permite al lenguaje hacer validaciones de tipo en tiempo de compilación. Genera rigidez lo cual es una desventaja ya que no permite que las estructuras varíen su dimensión.

Alocación Dinámica: permite declarar variables dinámicas o referenciadas. En pascal existe un tipo de datos PUNTERO que permite generar referencias a variables dinámicas.

PUNTEROS: Es un tipo de variable usada para almacenar la dirección de memoria de otra variable en lugar de un valor convencional.

* Es un tipo de dato simple que contiene la dirección de otro dato.
* Los punteros pueden apuntar solamente a variables dinámicas, es decir, a datos que están almacenados en memoria dinámica (HEAP).
* Cada variable de tipo puntero puede apuntar a un único tipo de dato.
* Un dato apuntado, no tienen memoria asignada, es decir no existe inicialmente espacio reservado en memoria para este dato.
* Una variable de tipo puntero ocupa una cantidad de memoria fija, independiente del tipo de dato al que apunta.
* Para poder emplear variables dinámicas es necesario emplear un tipo de dato que permita referenciar nuevas posiciones de memoria que no han sido reservadas necesariamente, y que se van a crear y destruir en tiempo de ejecución.
* Las variables dinámicas son por definición aquellas que se crean cuando se necesitan y se destruyen cuando ya han cumplido con su cometido.
* Para crear y destruir una variable referenciada (Reservar y liberar memoria) se utilizan las sentencias (NEW y DISPOSE).
* A una variable de tipo puntero se le permite asignar dos valores: Nil y otra variable de tipo puntero.

Dispose: elimina la conexión que existe entre la variable puntero y su dirección y luego libera esa dirección; por lo tanto, esa dirección podría ser reasignada luego (Por medio del new a otro puntero o al mismo).

Nil: Solo elimina la conexión que existe entre la variable puntero y su dirección; la memoria sigue ocupada y no puede ser reasignada, pero tampoco puede ser accedida.

Operaciones no permitidas en Punteros:

* No se permite leer una variable de tipo puntero. (Read(p))
* No se permite escribir una variable de tipo puntero. (Write(p))
* No se permite asignar el contenido a una variable puntero de manera manual (P: =AABC).
* No se permite comparar por menor o mayor la dirección de una variable puntero (‘If p > ABDE’; ‘If p >q’).

LISTAS:

* Es una estructura de datos que nos permite almacenar datos de una forma organizada, al igual que los vectores, pero, a diferencia de estos, esta estructura es dinámica, por lo que no tenemos que saber "a priori" los elementos que puede contener.
* Contiene elementos homogéneos, con una relación lineal que los vincula, es decir, cada elemento tiene un único predecesor y sucesor (Excepto el primero y el ultimo).
* Los elementos que la componen no ocupan posiciones secuenciales o contiguas de memoria, Es decir que pueden aparecer dispersos en la memoria, pero mantienen un orden lógico interno.
* Es de tipo: Homogénea, Dinámica, De acceso secuencial, Lineal.
* Están compuestas por nodos que se conectan por medio de enlaces o punteros, y debe existir un puntero que indique donde comienza la lista. El ultimo nodo tiene su puntero con valor NIL indicando que no hay más nodos en la lista.
* Cuando se necesita espacio adicional, nuevos nodos pueden ser alocados y agregados a la estructura (New), y cuando existen nodos que ya no se necesitan, pueden ser borrados, liberando memoria (Dispose).

Listas vs Vectores:

* Espacio: Se refiere a la cantidad de memoria utilizada por la estructura de datos. Suponiendo que tienen la misma cantidad de datos, los arreglos ocupan menor memoria que las listas (por los enlaces).
* Tiempo: Se refiere al tiempo que toma recuperar un dato de la estructura. En una lista al ser de acceso secuencial requiere un tiempo que no es constante, depende el número del elemento al que se quiere acceder. Y en un arreglo con acceso directo requiere un tiempo fijo.
* Datos: Se refiere a como se almacenan los datos. En las listas siempre hay lugar, distintos casos para agregar (al principio, final o según un orden). En los Arreglos no siempre hay lugar, distintos casos para agregar, siempre debo validar si hay espacio.
* Parámetros: por valor o por referencia:

Por Valor: de una lista se copia solo el puntero inicial. En un arreglo se realiza una copia de todo.

Por Referencia: de una lista se copia el valor inicial para poder ser modificado, del arreglo se copia la dirección en donde se encuentra almacenado.

***Calidad de un programa:***

Los programas deben atender a diferentes criterios que hacen a su calidad como:

* Corrección: (¿Hace lo que se pide?) El grado en que una aplicación satisface sus especificaciones y consigue los objetivos encomendados por el cliente.
* Eficiencia: (¿Cómo se usan los recursos del sistema?) Atiende a cómo el programa usa la memoria y el tiempo que consume su ejecución.
* Integridad de los datos que maneja: (¿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 localizas los fallos?) El esfuerzo requerido para localizar y reparar errores se vincula a la modularización, legibilidad y documentación.
* Flexibilidad: (¿Puedo añadir nuevas opciones?) El esfuerzo requerido para probar 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: (¿Podre usarlo en otra maquina?) El esfuerzo requerido para transferir la aplicación a otro hardware o sistema operativo.
* Reusabilidad: (¿Podre utilizar alguna parte del software en otra aplicación?) Grado en que partes de una aplicación pueden utilizarse en otras aplicaciones.
* Interoperabilidad: (¿Podrá comunicarse con otras aplicaciones?) El esfuerzo necesario para comunicar la aplicación con otras aplicaciones.
* Legibilidad: El código fuente de un programa debe ser fácil de leer y entender. Esto obliga a acompañar a las instrucciones con comentarios adecuados.
* Documentados: Todo el proceso de análisis y diseño del programa y su solución debe estar documentado mediante texto/gráficos para favorecer la comprensión, modificación y la adaptación a nuevas funciones.

***Corrección y Eficiencia:***

Un programa es correcto cuando cumple con la función especificada, esto significa que cumple con los requerimientos propuestos. Para determinar cuáles son esos requerimientos se debe tener una especificación completa, precisa y no ambigua del problema a resolver antes de escribir el programa.

Técnicas para medir corrección:

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

¿Cómo se proveen evidencias?:

* Diseñar de un plan de pruebas.
* Decidir cuales aspectos del programa deben ser testeados y encontrar datos de prueba para cada uno de esos aspectos.
* Determinar el resultado que se espera que el programa produzca para cada caso de prueba.
* Poner atención a los casos límites.
* Diseñar casos de prueba sobre la base de lo que hace el programa y no de lo que se escribió del programa.
* Diseñar casos de prueba antes de que comience la escritura del programa.

Durante este proceso es importante analizar las postcondiciones en función de las precondiciones establecidas en el programa.

* Debugging: es el procedo 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, como:

* El diseño del programa puede ser defectuoso.
* El programa puede usar un algoritmo defectuoso.
* Walkthroughs: consiste en recorrer el programa ante una audiencia.
* La lectura de un programa a otra persona provee un buen medio para detectar errores.
* Esta persona no comparte preconceptos y esta predispuesta a descubrir errores u omisiones.

Una vez que se cuenta con una solución correcta es necesario medir cuantos recursos se utilizan. (Tiempo de ejecución y Memoria utilizada)

Eficiencia de programas:

Se define la eficiencia como una métrica de calidad de los algoritmos, asociada con una utilización optima 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.

A blue background with black text

Description automatically generated

Para calcular la memoria estática utilizada, puedes seguir estos pasos:

1. Identifica todas las variables y estructuras de datos que has declarado en tu programa.
2. Determina el tamaño en bytes de cada tipo de dato utilizado en esas variables y estructuras (Tabla).
3. Suma todos los tamaños de las variables y estructuras para obtener la memoria estática utilizada
4. Si tu programa utiliza arreglos, debes tener en cuenta el tamaño de cada elemento del arreglo multiplicado por la cantidad de elementos en el mismo.
5. el tamaño del puntero no depende del tipo de dato al que apunte. Todos los punteros ocuparán el mismo tamaño, independientemente de si apuntan a enteros, caracteres, estructuras o cualquier otro tipo de dato.

Para calcular la memoria dinámica, recordamos que esta se gestiona mediante punteros y funciones específicas para la asignación y liberación de memoria (new, dispose).

1. Identificar las variables a las que se le reserva memoria con un new
2. Sumar el tamaño de los datos a los que apunta
3. Restar las variables que se liberan con un dispose.

medición del tiempo de ejecución en un programa: Depende de distintos factores

* Los datos de entrada al programa
* Tamaño
* Contenido
* La calidad del código generado por el compilador utilizado
* El tiempo del algoritmo base

Para algunos programas el tiempo de ejecución se refiere al tiempo de ejecución del, pero caso. En estos casos se obtiene una cota superior del tiempo de ejecución para cualquier entrada. Puede calcularse de dos maneras:

Análisis Empírico: es necesario ejecutar el programa y medir el tiempo empleado en la ejecución. Este análisis obtiene valores exactos para una maquina y unos datos determinados.

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, es independiente de la maquina donde se ejecute, permite analizar el comportamiento.

Consideraciones para el cálculo teórico:

1. Considerar el número de operaciones elementales que emplea el algoritmo.
2. Hay que considerar que una operación elemental utiliza un tiempo constate para su ejecución, independientemente del tipo de dato con el que trabaje.
3. Suponer que cada operación elemental se ejecutara en una unidad de tiempo
4. Suponer que una operación elemental es una asignación, una comparación o una operación aritmético/lógica simple.
5. Los comentarios, declaraciones y operaciones de entrada/salida no se consideran al realizar el calculo

Reglas generales para el cálculo de tiempo de ejecución:

1. For: 1 to N 🡪 Se debe calcular la cantidad de operaciones elementales que se ejecutan dentro del FOR y multiplicarla por la cantidad de veces que se ejecuta la instrucción for.

Además, La instrucción For realiza:

-Asignación inicial i: =1 (1)

-Testeo de i <= n (n+1)

-Incrementos de i: = i + 1 (2n)

En general 3n+2, siendo n la cantidad de repeticiones.

1. For anidados 🡪 Se aplica la regla del for,
2. While/Repeat 🡪 Se debe calcular la cantidad de operaciones elementales que se ejecutan dentro del while y multiplicarla por la cantidad de veces que se ejecuta, como no se conoce esa cantidad se considera el peor caso (n)
3. If then /Else 🡪 En el caso de una sentencia If then else, debe calcularse la cantidad de operaciones que se realizan en cada parte y se debe elegir aquella que consuma más tiempo, mayor cantidad de operaciones elementales.

***Organización de Computadoras:*** ☹

Arquitectura: Son aquellos atributos visibles al programador, conjunto de instrucciones, numero de bits usados para la representación de datos, mecanismos de Entrada/Salida, técnicas de direccionamiento.

Organización: Indica como son implementados esos atributos (transparentes al programador), señales de control, interfaces, tecnología de memoria.

Funciones de las computadoras:

* Procesamiento de datos
* Almacenamiento de datos
* Movimiento de datos
* Control
* Jugar al Minecraft

Arquitectura Von Neumann: tiene como aspectos importantes la utilización del sistema binario, las instrucciones y datos residen en memoria y la memoria es direccionable por localidad sin importar el dato almacenado.

tiene 5 componentes principales

* Unidad de entrada: provee las instrucciones y los datos.
* Unidad de memoria: donde se almacenan datos e instrucciones.
* Unidad aritmético-lógica: procesa los datos.
* Unidad de control: dirige la operación.
* Unidad de salida: se envían los resultados.

Central Prossesing Unit (CPU):

Un sistema de cómputo está constituido por 3 subsistemas, estas componentes se comunican entre si mediante el Sistema de Interconexión:

1. CPU
2. Memoria
3. Entrada/Salida

Buses: El CPU intercambia información mediante los buses, los cuales son un camino de comunicación entre dos o más dispositivos, están agrupados en canales (un bus de 32 bits son 32 canales separados de un solo bit cada uno).

* Bus de Direcciones: indica el origen o el destino de los datos
* Bus de Datos: transporta datos o instrucciones
* Bus de control: conjunto de señales que sincroniza las actividades con los periféricos del sistema

Arithmetic Logic Unit (ALU):

Realiza las operaciones lógicas y matemáticas a partir de las instrucciones definidas en el programa en ejecución. Recibe operandos a procesar y devuelve tanto resultados como banderas de condición (flags)

Ciclo de Instrucción:

El procesamiento requerido para una sola instrucción se llama ciclo de instrucción y se divide en dos etapas: ciclo de búsqueda (1,2) y ciclo de ejecución (3,4)

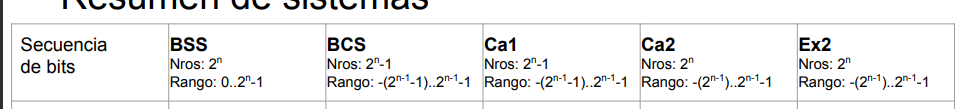
1. Buscar la instrucción en la memoria principal
2. Decodificar la instrucción
3. Ejecutar la instrucción
4. Almacenar o guardar resultados

Accesos a memoria o E/S:

1. Calculo dirección instrucción: determina la dirección de la siguiente instrucción a ejecutarse.
2. Búsqueda instrucción: lee la instrucción de su posición de memoria a la CPU.
3. Decodificación de la instrucción: analiza la instrucción para determinar el tipo de operación a realizar y los operandos que se usaran.
4. Calculo dirección operando: si la operación implica la referencia a un operando en la memoria o e/s, entonces se determina la dirección.
5. Búsqueda del operando: busca el operando en la memoria o e/s.
6. Operación sobre los datos: ejecuta la instrucción.
7. Calculo dirección resultado.
8. Almacenamiento resultado.

Números enteros:

Rango: para un sistema de representación dado, el rango contempla el intervalo de valores representables



Operaciones Aritméticas:

Flags: son bits que el procesador establece de modo automático acorde al resultado de cada operación realizada.

* Z(zero): vale 1 si todos los bits del resultado de una operación son cero.
* N(negativo): vale 1 si el bit más significativo del resultado también es 1.
* C(carry): flag de acarreo, no se puede representar un resultado con esos bits.
* V(overflow): flag de desbordamiento, indica una condición fuera de rango en Ca2.

Punto Fijo:

En los sistemas de punto fijo se considera que todos los números a representar tienen exactamente la misma cantidad de dígitos y la coma fraccionaria esta siempre ubicada en el mismo lugar.

Punto Flotante:

Es una forma de representar un numero con una mantisa y un exponente. La mantisa M puede ser fraccionaria, la base B es implícita y no necesita almacenarse. MxB^E