ALGORITMOS, DATOS Y PROGRAMAS

CONCEPTOS BASICOS

- **Informática:** la informática es la ciencia que estudia el análisis y resolución de problemas utilizando computadoras.
- **Etapas de la resolución de problemas:**
  1) Análisis del problema: se analiza el problema en su contexto con el mundo real. Deben obtenerse los requerimientos del usuario y el resultado es un modelo preciso del ambiente del problema y del objetivo a resolver.
  2) Diseño de una solución: el primer paso es la modularización del problema, es decir, la descomposición del mismo en partes que tendrán una función bien definida y datos propios. Estas partes deben estar comunicadas.
  3) Implementación (escritura del programa): un algoritmo es una especificación simbólica que debe convertirse en un programa real sobre un lenguaje de programación concreto. Debe ser traducido (automáticamente) al lenguaje de la máquina que le vaya a ejecutar, esto se denomina compilación, permite detectar y corregir los errores sintácticos que se cometan en la escritura.
  4) Verificación: una vez que se tiene un programa escrito y depurado de errores de sintaxis, debemos verificar que su ejecución conduce al resultado deseado, con datos representativos del problema real.
- **Algoritmo:** es la especificación rigurosa de la secuencia de pasos (instrucciones) a realizar sobre una autómata para alcanzar un resultado deseado en un tiempo finito.
    - Especificación rigurosa: que debe ser claro y univoco.
    - Autómata: sinónimo de computadora.
    - Tiempo finito: que debe comenzar y terminar.
- **Programa:** es un conjunto de instrucciones, ejecutables sobre una computadora, que permite cumplir una función específica (dichas ordenes están expresadas en un lenguaje de programación concreto).
    - **Componentes básicos de un programa:**
        - Instrucciones: representan las operaciones que ejecutara la computadora al interpretar el programa.
        - Datos: valores de información de las que se necesita disponer y en ocasiones transformar para ejecutar la función del programa.
    - **Características de los programas:**
      1) **Para la audiencia humana:**
          - Operatividad: el programa debe realizar la función para la que fue concebida.
          - 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.
      2) **Para la CPU de la computadora:**
          - Debe contener instrucciones validas
          - Deben terminar
          - No deben utilizar recursos inexistentes

- **¿Qué es programar?:** las tareas más importantes que tiene alguien que debe escribir un programa para resolver un problema sobre una computadora son:
    - Elegir la representación adecuada de los datos del problema.
    - Elegir el lenguaje de programación a utilizar, según el problema y la maquina a emplear.
    - Definir el conjunto de instrucciones cuya ejecución ordenada conduce a la solución.
- **Lenguaje de programación:** es un lenguaje formal diseñado para expresar procesos que pueden ser llevados a cabo por maquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento lógico y físico de una máquina, para expresar algoritmos de precisión o como modo de comunicación humana.
- **Ambiente de programación:** es el entorno en el cual el programador desarrolla sus aplicaciones. Hay dos ambientes reconocidos, uno visual donde se manipulan los objetos y se generan conexiones. Y otro de tipo código, donde se generan las instrucciones de programación a base de ir escribiendo y compilando.
- **Lenguaje fuertemente tipado:** significa que no permite violaciones de los tipos de datos, es decir, dado el valor de una variable de un tipo concreto, no se puede usar como si fuera de otro tipo distinto a menos que se haga una conversión.
- **Pre-condición y Post-condición:**
    - Precondición: es la información que se conoce como verdadera antes de iniciar el programa (o modulo). Por ejemplo, seria calcular el factorial de un número (exigirá que dicho numero sea mayor o igual a cero)
    - Postcondición: es la información que debería ser verdadera al concluir el programa (o modulo), si se cumple adecuadamente los pasos especificados. Por ejemplo, seria calcular el resultado de un factorial (exigirá que el resultado debe ser un entero y que debe ser mayor o igual a cero).

DATOS. TIPOS DE DATOS. ALCANCE DE LOS DATOS

- **Dato:** es una representación de un objeto del mundo real mediante la cual podemos modelizar aspectos que se quiere resolver con un programa sobre una computadora. Conceptualmente pueden ser constantes (no cambian su valor durante la ejecución) o variables (pueden cambiar su valor).
- **Tipo de dato:** es una clase de objetos de datos ligados a un conjunto de operaciones para crearlos y manipularlos. Se caracterizan por un rango de valores posibles, su representación interna y por un conjunto de operaciones realizables sobre este tipo. Se clasifican en:
    - Datos simples: son los que toman solamente un único valor y están acotados por el lenguaje, pueden ser de tipo numérico (integer, real), de tipo lógico (boolean) o de tipo carácter (char).
    - Datos compuestos: son los que toman más de un valor y están definidos por el usuario, por ejemplo, los registros, arreglos, listas, árboles o conjuntos.
- **Tipo de dato NUMERICO:**
    - Tipo de dato entero: es el más simple de todos. Dado que una computadora tiene memoria finita, la cantidad de valores enteros que se pueden representar sobre ellas son finitos, por esto se deduce que existe un número entero máximo y otro mínimo.
    - Tipo de dato real: permite representar números decimales. Consiste en definir cada número como un mantisa (parte decimal) y un exponente (parte fraccionaria).
- **Tipo de dato LOGICO:** permite representar datos que pueden tomar solamente uno de dos valores. También es llamado tipo de dato boolean. Dichos valores son: TRUE (verdadero) y FALSE (falso).

Se utilizan en situaciones donde se presentan dos alternativas de una solución. Los operados lógicos básicos son: NOT (negación), AND (conjunción) y OR (disyunción). El resultado de estos corresponde a las tablas de verdad.

- **Tipo de dato CARÁCTER:** representa un conjunto finito y ordenado de caracteres que la computadora reconoce. Contiene un solo carácter y los que reconoce la computadora son un conjunto estándar llamado ASCII, el cual permite establecer un orden de precedencia. Se debe tener en cuenta que no es lo mismo el valor entero 1 que el símbolo carácter '1'.
- **Utilidad de tener tipos de datos:** son útiles tenerlos porque generan flexibilidad (solo debemos modificar una declaración en lugar de una serie de declaraciones de variables), ayuda en la documentación (facilitando el entendimiento y la lectura del programa) y trasmiten seguridad (se reducen los errores de correspondencia entre el valor que se pretende asignar a una variable y el previamente declarado).
- **Variables:** una variable es una zona de memoria cuyo contenido va a ser de alguno de los tipos de datos. La dirección inicial de esta zona se asocia con el nombre de la variable.
    - Declaración: {Var
      nombreVariable : tipoVariable;}
    - Variables locales: solo pueden ser accedidas por el modulo en las que se la declara.
    - Variables globales: son las que pueden ser accedidas en cualquier parte del programa.
- **Ocultamiento y protección de datos:** conocido como Data Hidding, se utiliza para significar que todo dato que es relevante para un módulo debe ocultarse de los otros módulos. De esta manera se evita que en el programa principal se declares datos que solo son relevantes para algún modulo en particular y se protege la integridad de los datos.
- **Alcance de los datos:** así como se declaran módulos dentro del programa principal, pueden declararse módulos dentro de otros. Por lo tanto, deben aplicarse reglas que gobiernan el alcance de los identificadores:
    - El alcance de un identificador es el bloque de programa donde se lo declara.
    - Si un identificador declarado en un bloque es nuevamente declarado en un bloque interno, el segundo bloque es excluido del alcance de la primera sección.

ESTRUCTURAS DE CONTROL

Una estructura de control es el conjunto mínimo de instrucciones que permiten especificar el control que se quiere implementar. Este conjunto debe tener como mínimo una secuencia, una decisión, una selección, una repetición o una iteración.

- **Operaciones de entrada:**
    - Read: se usa para leer datos (por defecto desde teclado) y asignarlos a las variables correspondientes.
    - Readln: es igual al read pero hace que la siguiente sentencia comience leyendo en la línea de abajo.
- **Operaciones de salida:**
    - Write: se usa para mostrar el contenido de una variable (por defecto en pantalla). Si los datos son más de uno se separan por comas.
    - Writeln: igual al write pero hace que la siguiente sentencia salte directamente a la línea de abajo.
- **Estructura de control SECUENCIA:** es la más simple y está representada por una sucesión de operaciones en la que el orden de ejecución coincide con el orden físico de aparición de las instrucciones.
- **Estructura de control DECISION:** instrucción no secuencial que permite tomar decisiones en función de los datos del problema. Estructura {IF (condición) THEN}, si la condición es verdadera ejecuta las instrucciones correspondientes, si es falso puede ir por el {ELSE} y ejecutar las instrucciones dentro.

- **Estructura de control SELECCIÓN:** permite realizar una o más acciones, dependiendo de cuál de todas las condiciones evaluadas es verdadera. La variable debe ser de tipo ordinal. Estructura {CASE (variable) OF, posibilidades, ELSE, otras acciones}

- **Estructura de control REPETICION:** consiste repetir n veces un bloque de acciones. Este número n es fijo y conocido de antemano. Dentro de las acciones del ciclo no se puede modificar la variable *índice*. La variable de control debe ser de tipo ordinal (entero, boolean, char) y los incrementos, decrementos y testeos son implícitos. Estructura {FOR (índice:valor_inicial) TO/DOWNTO (valorfinal) DO}
- **Estructura de control ITERACION:** ejecuta un bloque de instrucciones desconociendo el número exacto de veces que puede ejecutarse. Se clasifican en pre y post condicionales.
    - Precondicionales: evalúa la condición y luego ejecuta el bloque de instrucciones. Se ejecuta cero a más veces, mientras la condición sea verdadera. Estructura {WHILE (condición) DO}
    - Postcondicionales: ejecuta el bloque de acciones y luego evalúa la condición. Se ejecuta de 1 a más veces, mientras la condición sea falsa. Estructura {REPEAT (bloque de acciones) UNTIL (condición)}

CALIDAD DE UN PROGRAMA

Factores que determinan la calidad del software.

- **Clasificación:** Se clasifican en tres grupos:

1) **Operatividad del producto:** características operativas.
    - Corrección: grado en que una aplicación satisface sus especificaciones y consigue los objetivos pedidos por el cliente.
    - Fiabilidad: grado que se puede esperar que una aplicación lleve a cabo las operaciones especificadas y con la precisión requerida.
    - Eficiencia: la cantidad de recursos hardware y software que necesita una aplicación para realizar las operaciones con los tiempos de respuesta adecuados.
    - Integridad: grado con que puede controlarse el acceso al software o a los datos a personal no autorizado.
    - Facilidad de uso: el esfuerzo requerido para aprender el manejo de una aplicación, trabajar con ella, introducir datos y conseguir resultados.
2) **Revisión del producto:** capacidad para soportar cambios.
    - Facilidad de mantenimiento: el esfuerzo requerido para localizar y reparar errores, se va a vincular ocn la modularización y con cuestiones de legibilidad y documentación.
    - Flexibilidad: el esfuerzo requerido para modificar una aplicación en funcionamiento.
    - Facilidad de prueba: el esfuerzo requerido para probar una aplicación de forma que cumpla con lo especificado en los requisitos.
3) **Transición del producto:**
    - Portabilidad: el esfuerzo requerido para transferir la aplicación a otro hardware o sistema operativo.
    - Reusabilidad: grado en que las partes de una aplicación pueden utilizarse en otras aplicaciones.
    - Interoperabilidad: el esfuerzo necesario para comunicar la aplicación con otras aplicaciones o sistemas informáticos.

- **Características que hacen a la calidad:**
    - 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 cuando cumple con la función especificada (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. Para medir si es correcto se debe probar con datos reales que permitan verificar su función.

- **Técnicas para verificar corrección:**
  - Testing (testeo): es el proceso de proveer evidencias convincentes respectos de si el programa hace el trabajo esperado. Las evidencias se pueden proveer de varias maneras:
    - Diseñando un plan de pruebas.
    - Poniendo atención en los casos límites.
    - Determinando el resultado que se espera que el programa produzca para cada caso de prueba.
    - Diseñar los casos de prueba antes de que comience la escritura del programa.
    - Cuando se descubre el error, se localiza y se corrige. Siempre que se corrige un error, se prueba el programa con el conjunto entero de casos de prueba.
  - Debugging: es el proceso de descubrir y reparar la causa 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 monitorear su comportamiento más cercanamente.

    Los errores pueden provenir de:
    - El diseño del programa puede ser defectuoso.
    - El programa puede usar un algoritmo defectuoso.
    - A veces el error es muy notorio y se reconoce fácilmente, otras se puede necesitar agregar sentencias de salidas adicionales que sirven como punto de control.
  - Walkthroughs: es el proceso de recorrer el programa ante una audiencia.
    - La lectura del programa a otra persona es un buen medio para detectar errores.
    - Esta persona no comparte preconceptos y esta predispuesta a descubrir errores u omisiones.
    - Cuando no se detecta el error, el programador trata de probar que no existe, pero mientras lo hace, el o la otra persona pueden detectar el error.
  - Verificación: es el proceso de analizar postcondiciones en función de las precondiciones establecidas.

- Las precondiciones y postcondiciones permiten describir la función que realiza un programa, sin especificar un algoritmo determinado.
- Las precondiciones describen los aspectos que deben considerarse antes de que el programa pueda comenzar a ejecutarse. Se consideran siempre volátiles.
- Las postcondiciones describen los aspectos que deben cumplirse cuando el programa termino.

EFICIENCIA

Es la métrica de calidad de los algoritmos, asociada con una utilización óptima de los recursos del sistema de cómputo donde se ejecutara el programa.
El análisis de eficiencia de un algoritmo estudia el tiempo de ejecución y el espacio de memoria que requiere.

- **Medición del tiempo de ejecución:** depende de:
  - 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 maquina empleadas en la ejecución.
  - El tiempo del algoritmo base.
- **Medición de la memoria utilizada:** solamente se puede calcular la memoria estática que utiliza. Se deben analizar las variables declaradas y el tipo correspondiente.
- **Métodos para medir la eficiencia respecto el tiempo de ejecución:**
  - Análisis empírico: es necesario ejecutar el programa y medir los recursos utilizados. Las desventajas es que puede dar una información pobre de los recursos consumidos, es completamente dependiente de la maquina donde se ejecuta, requiere implementar el algoritmo y ejecutarlo repetidas veces.
  - Análisis teórico: es necesario establecer una medida intrínseca de la cantidad de trabajo realizado por el algoritmo, esto permite comparar algoritmos y seleccionar la mejor implementación. Algunas características son:
    - Obtiene valores aproximados.
    - Es aplicable en la etapa de diseño.
    - Es independiente de la máquina donde se ejecute.
    - Permite analizar el comportamiento.
    - Se tienen en cuenta el número de operaciones elementales que emplea el algoritmo.
    - Cada operación elemental se ejecutara en una unidad de tiempo.
    - Se puede aplicar sin necesidad de implementar el algoritmo.
    - Una operación elemental será una asignación, una comparación o una operación aritmética.
- **Reglas para el cálculo del tiempo de ejecución:**
  - For.
  - For anidados.
  - Sentencias consecutivas
  - If / Else
- **¿Una solución eficiente es siempre correcta?**
  Sí, porque para verificar si es eficiente o no, primero tenemos que comprobar que sea correcto.

- **¿Desde el punto de vista de eficiencia, una solución recursiva es más o menos eficiente que una iterativa?**
  Es menos eficiente por causa del overhead (sobrecarga) de tiempo y memoria asociado con la llamada a subprogramas.

RECURSION

Una solución recursiva resuelve un problema por resolución de instancias más pequeñas del mismo problema, es decir, el problema es el mismo pero de menor tamaño. Esta solución se llama a si misma y tiene un caso base o caso degenerado.

- **Características:**
  - Como definir un problema en términos más pequeño del mismo tipo.
  - Como será disminuido el tamaño del problema en cada llamado.
  - Que instancia del problema servirá como caso base.

- **¿Crees que toda solución recursiva puede ser escrita en forma iterativa? ¿Por qué?**
  Cualquier cálculo recursivo puede ser expresado como una iteración, porque una solución iterativa se puede resolver de la misma manera que una recursiva pero sin respetar sus operaciones de auto invocación, caso base y alguna situación que garantice que se alcanzó el caso base.
- **¿El código de una solución recursiva es más corto que el de una solución iterativa?**
  El código de una solución recursiva ocupa más memoria y requiere más tiempo de ejecución que una solución iterativa, pero en cuanto a líneas de código es más corto uno recursivo.
- **¿Una solución recursiva es más legible que una solución iterativa?**
  Si, un problema recursivo posee una solución más legible.
- **¿Todos los algoritmos recursivos pueden ser resueltos también en forma iterativa?**
  Se puede escribir cualquier algoritmo recursivo en forma iterativa usando un bucle y el caso base como corte del bucle. Lo que no se puede hacer siempre es escribir una solución iterativa en forma recursiva.
- **¿Puede existir más de un caso base, puedes ejemplificar?**
  Si verdadero, ejemplo en las búsquedas dicotómicas y capicúa de vectores.

TIPOS DE DATOS DEFINIDOS POR EL USUARIO

Un tipo de dato def. por el usuario es aquel que no existe en la definición del lenguaje, y el programador es el encargado de su especificación. Tiene asociado un rango de valores posibles, una forma de representación, un conjunto de operaciones permitidas y un conjunto de condiciones de valores permitidas que se pueden verificar.

- **Declaración:** {Type *identificador: nombre
  Identificador = tipo;} *tipo: estándar o def. por el usuario
- **Ventajas de declarar estos tipos:**
  - Flexibilidad: en el caso de ser necesario modificar la forma en que se expresa el dato, solo se debe modificar una declaración en lugar de u conjunto de declaraciones de variables.
  - Documentación: se pueden usar como identificador de los tipos, nombres autoexplicativos, facilitando 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.

- **Tipo de dato ENUMERATIVO:** un dato enumerativo puede verse como una lista ordenada de valores posibles para las variables del tipo.
  Los valores que se definen en la lista no pueden ser de los tipos estándar, tampoco pueden repetirse valores entre identificadores distintos.
  No es posible realizar operaciones de entrada y salida, debido a esto estas variables son de uso interno, colaboran con la claridad del programa.
    - Declaración: {Type
      Identificador = (valor1,valor2,..,valorN);}
    - Funciones aplicables:
      - PRED: devuelve el valor anterior del enumerativo.
      - SUCC: devuelve el valor siguiente del enumerativo.
      - ORD: da la posición relativa del argumento dentro de la definición del enumerativo.
- **Tipo de dato SUBRANGO:** es un tipo de dato ordinal que consiste de una sucesión de valores de tipo ordinal tomado como base. Las operaciones de un subrango se heredan del tipo base. Facilita el chequeo de posibles errores y ayuda al mantenimiento del programa.
    - Declaración: {Type
      Identificador = set of *tipo_ordinal*;}
- **Tipo de dato CONJUNTO:** representa una colección de datos simples ordinales, sin elementos repetidos y sin ningún orden interno. La cantidad de elementos que contiene el conjunto puede estar limitada por la implementación en cada lenguaje o sistema operativo. En pascal estándar el número máximo de elementos es de 255. Un conjunto no admite operaciones de lectura y escritura.
    - Declaración: {Type
      Identificador = valorinicial .. valorfinal;}
    - Operaciones:
      - Asignación: se le puede asignar valores escribiendo sus elementos consecutivamente, encerrados entre corchetes y separados por comas.
      - Unión: se representa con el signo '+' y da como resultado otro conjunto donde aparece la suma de dos conjuntos. No se repiten elementos.
      - Intersección: se representa con el signo '*' y da como resultado otro conjunto donde aparecen los elementos comunes a dos conjuntos.
      - Diferencia: se representa con el signo '-' y da como resultado otro conjunto donde aparecen los elementos que están en el primer conjunto y no están en el segundo.
      - Pertenencia: se representa con el operador 'IN' y da como resultado un valor lógico. Verdadero si el elemento está en el conjunto, falso caso contrario.
- **Tipo de dato STRING:** es una sucesión de caracteres de un largo determinado que se almacenan en un área contigua de la memoria.
    - Declaración: {Type
      Identificador = string [longitud];}
      Longitud es el máximo número de cantidad de caracteres que puede contener el dato. Cuando no se especifica la longitud ese identificador puede tener como máximo 255 caracteres (en pascal).
    - Operaciones:
      - Asignación: se hace igual que si fuera una variable de tipo carácter, := . Si supera el máximo de caracteres declarado, se pierden los datos a partir de ahí y se dice que se "trunca".
      - Operadores relacionales: pueden compararse con =, <>, <=, >=. Estas operaciones realizan la comparación carácter por carácter.
      - Entrada y salida: admite las operaciones Read y Write de pascal.

MODULARIZACION – PARAMETROS

Modularizar significa dividir un problema en partes funcionalmente independientes, que encapsulen operaciones y datos. Cada una de estas partes se denomina modulo.
Se trata de separar en funciones lógicas con datos propios y datos de comunicación perfectamente especificados.

- **Descomposición:** tiene siempre un objetivo. Cuando se descompone un problema en subproblemas, deben ser de tal forma que:
    - Cada subproblema está en un mismo nivel de detalle.
    - Cada subproblema puede resolverse independientemente.
    - Las soluciones de los subproblemas deben combinarse para resolver el problema original.
- **Modulo:** es una tarea específica bien definida, que se comunica con otros módulos adecuadamente y cooperan para conseguir un objetivo en común. Cada módulo encapsula acciones o tareas, y debe representar los datos relevantes del subproblema a resolver.
- **Top down:** ir de lo general a lo particular. Dividir, conectar y verificar.
- **Ventajas:** importancia en el desarrollo de sistemas de software.
    - Mayor productividad: al dividir un sistema de software en módulos funcionalmente independientes, un equipo de desarrollo puede trabajar simultáneamente en varios módulos, reduciendo el tiempo de desarrollo global y aumentando las posibilidades de producción.
    - Reusabilidad: es la posibilidad de utilizar repetidamente el producto de software desarrollado. Naturalmente la descomposición funcional que ofrece la modularización favorece el re-uso.
    - Facilidad de mantenimiento correctivo: la división lógica de un sistema en módulos permite aislar los errores que se producen con mayor facilidad. Esto significa poder corregir los errores en menor tiempo y disminuir los costos de mantenimiento de los sistemas.
    - Facilidad del crecimiento del sistema: los sistemas de software reales crecen (aparecen con el tiempo nuevos requerimientos del usuario). La modularización permite disminuir los riesgos y costos de incorporar nuevas prestaciones a un sistema en funcionamiento.
    - Mayor legibilidad: mayor claridad para leer y comprender el código fuente. El ser humano maneja y comprende con mayor facilidad un número limitado de instrucciones directamente relacionados.

Pascal reconoce dos tipos de módulos: Procedimientos y Funciones.

- **Procedimiento (procedure):** es un conjunto de instrucciones que realizan una tarea específica y como resultado pueden retornar 0, 1 o más valores.
  Se invocan escribiendo su nombre seguido de los datos de comunicación.
  Permite parámetros de entrada y de entrada/salida.
  Permite operaciones de lectura y escritura.
- **Función (function):** es un módulo que realiza una única tarea y devuelve un solo valor de tipo simple.
  Obligatoriamente la última instrucción ejecutada deberá asignarle un valor al nombre de la función (del tipo especificado).
  Se pueden invocar dentro de un IF, o de un WHILE, o asignarla a una variable o dentro de un WRITE.
  Pueden recibir solo parámetros de entrada.
- **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.
- **Variable local:** su declaración se hace en un módulo particular, y solo podrá ser usadas 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 modulo y programa:** se hace a través de variables globales o el uso de parámetros. A través de variables globales presenta muchas desventajas como demasiados identificadores, conflictos de nombres de identificadores, posibilidad de perder integridad de los datos, entre otras. La solución a estos problemas es una combinación de ocultamientos de datos (data hidding) y uso de parámetros.
    - Data hidding significa que los datos exclusivos de un módulo no deben ser “visibles” o utilizables por los demás módulos.

        - El uso de parámetros significa que los datos compartidos se deben especificar como parámetros que se trasmiten entre módulos. En pascal pueden ser parámetros por valor o por referencia.
- **Pasaje de parámetros:** El concepto de pasajes de parámetros hace referencia a la manera que tienen los módulos de comunicarse, estos parámetros se envían desde la invocación del módulo (parámetros actuales) y se comunican con los parámetros formales (son los declarados en el encabezado del módulo). Pueden ser parámetros por valor o parámetros por referencia.
- **Parámetros por valor:** es un dato de entrada que significa que el modulo recibe una copia de un valor proveniente de otro modulo o del programa principal. Con este dato puede realizar operación y/o cálculos, pero fuera del módulo ese dato no reflejara cambios.
  Deben ser tratados como una variable local del módulo y utilizarlos puede significar una utilización importante de memoria.
- **Parámetros por referencia:** es un dato que contiene la dirección de memoria donde se encuentra la información compartida con otro modulo o programa que lo invoca. El modulo puede operar con la información que está dentro de la dirección y las modificaciones que se produzcan se reflejaran en los demás módulos que conocen esa dirección. Operan directamente sobre la dirección de la variable original, esto significa que no requiere memoria local.

ESTRUCTURAS 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.

- **Clasificación:**
    - Por el tipo de dato que la compone:
        - Homogénea: los datos que la componen son del mismo tipo.
        - Heterogénea: los datos que la componen son de distinto tipo.
    - Por la ocupación de memoria:
        - Estática: la cantidad de elementos que contiene es fija, es decir que la memoria que ocupa no varía con la ejecución.
        - Dinámica: la cantidad de elementos que contiene es variable, y por lo tanto la memoria que ocupa puede variar con la ejecución.
    - Por su acceso a los elementos:
        - Acceso secuencial: para acceder a un elemento particular se debe respetar un orden predeterminado, por ejemplo, pasando por todos los elementos que le preceden.
        - Acceso directo: también llamado indexado. Se puede acceder a un elemento particular directamente, sin necesidad de pasar por los anteriores a él, indicando su posición.
    - Por la relación entre sus elementos:
        - Lineales: a cada elemento le sigue uno y le precede uno, solamente.
        - No lineales: para cada elemento pueden existir, cero, uno o mas elementos que le suceden y preceden.

REGISTRO (RECORD)

Es una colección de valores, llamados campos, y cada uno tiene un identificador; estos campos son nombrados individualmente, como variables ordinales. Tiene tres características:

- Es heterogénea, los valores pueden ser de diferentes tipos.
- Es estática, la memoria que ocupa es fija.
- Es de acceso directo a sus componentes (campos).

- **Acceso a los campos:** para acceder se necesita especificar tanto el nombre del registro como el del campo que interesa. Esto se denomina calificar al campo. En Pascal seria {nombre-registro.nombre-campo;}
- **Operaciones:** la única operación permitida es la de asignación, siempre y cuando, sean del mismo tipo. El resto de operaciones deben ser aplicadas a cada campo del registro (campo a campo).

- **Sentencia WITH:** permite que la variable de tipo registro sea nombrada una vez, y luego los campos sean accedidos directamente.

ARREGLOS (ARRAY)

Es una colección de elementos que se guardan consecutivamente en la memoria y se pueden referenciar a través de su índice. Tiene algunas características:

- Es homogénea, los elementos almacenados son todos del mismo tipo.
- Es estática, la memoria ocupada es fija durante su ejecución.
- Es de acceso indexado, los elementos pueden recuperarse en cualquier orden, indicando la posición que ocupa.
- Es lineal, ya que cada elemento tiene un elemento que le precede y otro que le sucede.
- Un arreglo con un solo índice se denomina vector.
- La cantidad total de memoria depende de la cantidad de elementos y del tipo de dato.

- **Declaración:** {Type
  Vector = array [índice] of tipo_elementos;}
  Donde índice corresponde a cualquier tipo ordinal (entero, carácter, subrango) y los elementos pueden pertenecer a cualquier tipo de datos de alocación estática.

- **Operaciones:**
  - Asignación
  - Lectura/escritura
  - Recorridos
  - Cargar datos
  - Agregar elementos al final
  - Insertar elementos
  - Borrar elementos
  - Búsqueda de un elemento
  - Ordenación de los elementos
- **Recorridos parcial y total:** la operación de recorrido puede hacerse de dos maneras:
  - Recorrido parcial: implica analizar los elementos del vector hasta encontrar aquel que cumple con lo pedido. Puede ocurrir que se recorra todo el vector.
  - Recorrido total: implica analizar todos los elementos, lo que lleva a recorrer completamente la estructura.
- **Agregar elementos al final:** para esta operación hay que verificar si existe lugar en el arreglo, luego agregar el elemento y por ultimo aumentar su dimensión lógica.
- **Insertar elementos:** esta operación consiste en introducir dicho elemento en el interior de un arreglo, en una posición i dada, de tal forma que los elementos ubicados en las posiciones siguientes sean desplazados a las posiciones i+1 respectivamente, siempre y cuando haya lugar en el arreglo.
- **Borrar elementos:** hay que verificar que la posición sea validad, luego hacer el corrimiento (i-1) contrario al de insertar para reorganizar el vector y por ultimo disminuir la dimensión lógica.
- **Dimensión lógica y física del arreglo:**
  - Dimensión lógica: se determina cuando se cargar los elementos del arreglo. Indica la cantidad de posiciones de memoria ocupadas con contenido real. Puede ir variando dependiendo de si se insertan o borran elementos.
  - Dimensión física: se especifica en el momento de la declaración y determina su ocupación máxima de memoria. La cantidad de memoria no variara durante la ejecución del programa.
- **Búsqueda de un elemento:**
  - Búsqueda lineal o secuencial: es el proceso de buscar desde el principio de la estructura, analizando los elementos que contienen uno a uno hasta encontrarlo o hasta llegar al final. Requiere excesivo consumo de tiempo y es ineficiente en estructuras muy grandes.

- Búsqueda dicotómica: el vector tiene que estar ordenando y se basa en la estrategia de "divide y vencerás".
  Se compara el valor buscado (x) con el ubicado en el medio del vector (a). Si el elemento ubicado al medio del vector es igual a (x), termino la búsqueda. Sino, debería quedarse con la mitad del vector que conviene para seguir la búsqueda. Estos pasos se repiten hasta encontrar el elemento o se acaba el vector.
  Cada vez que se toma la mitad del arreglo, se va disminuyendo el tamaño del mismo.

- **Ordenación de elementos:** existen tres métodos:
  - Método de selección: para ordenar un vector de n elementos, a[1] .. a[n], busca el elemento menor según el criterio adoptado y lo ubica al comienzo. Para mantener la dimensión del vector lo intercambia con a[1]. A continuación se busca el segundo elemento menor del vector y se intercambia con a[2]. Continúa así sucesivamente buscando el próximo menor y lo coloca en su lugar, hasta que todos los elementos quedan ordenandos. Para ordenar un vector completo es necesario realizar n-1 pasadas por el vector.
  - Método de intercambio o burbujeo: procede de una manera similar al de selección, en el sentido de que realiza n-1 pasadas. Pero a diferencia del anterior, realiza correcciones locales de distancia 1, es decir, compara ítems adyacentes y los intercambia si están desordenados. Al final de la primera pasada, se habrán comparado n-1 pares y el elemento más grande habrá sido arrastrado, como una burbuja, hacia el final del vector. Al final de la pasada, el segundo máximo habrá sido ubicado en la posición n-1.

  - Método de inserción: ordena el vector de entrada insertando cada elemento a[i] entre los i-1 anteriores que ya están ordenados. Para realizar esto comienza a partir del segundo elemento, suponiendo que el primero ya está ordenando. Si los dos primeros elementos están desordenados, los intercambia. Luego, toma el tercer elemento y busca su posición correcta con respecto a los dos primeros.

PUNTEROS

Es un tipo de dato simple que contiene la dirección de otro dato. Mediante la variable de tipo puntero se accede a esa otra variable, almacenada en la dirección de memoria que señala el puntero.
Los punteros pueden apuntar solamente a variables dinámicas, datos que están almacenados en la heap (memoria estática). Ocupa 4 bytes de memoria (stack) para su representación.

- **Declaración:** {Type
  tipoPuntero = ^tipoVariableApuntada;}
- **Alocación estática (stack):** el espacio de memoria se reserva con anticipación y no cambia dirante la ejecución del programa. Esto permite una comprobación de tipos en tiempo de compilación. Presenta un inconveniente que es su rigidez, ya que no pueden crecer o decrecer durante la ejecución del programa.
- **Alocación dinámica (heap):** los espacios de memoria asignados a las variables dinámicas se reservan y se liberan durante la ejecución del programa. La ventaja es su flexibilidad, ya que pueden crecer o decrecer durante su ejecución.
- **Operaciones:** las variables dinámicas son por definición aquellas que se crean cuando se necesitan y se destruyen cuando ya han cumplido con su cometido.
  - Creación: NEW (puntero);
  - Destrucción: DISPOSE (puntero);
- **Operaciones frecuentes:**
  - Asignación de un valor a una variable puntero.
  - Asignación de valor al objeto "referenciado" por el puntero.
  - Acceso a la información del objeto "referenciado" por el puntero.
  - Eliminación de un objeto que no se necesita (dispose()).
  - Operaciones de entrada/salida (no podemos leer y escribir punteros. Si podemos leer y escribir los objetos que ellos referencian dependiendo del tipo apuntado).
  - Operaciones de comparación (pueden aparecer en expresiones relacionales como p=q y p<>q).

LISTAS

Es una colección de elementos homogéneos, con una relación lineal que los vincula. Los elementos que la componen no ocupan posiciones secuenciales o contiguas de memoria, es decir que los elementos pueden aparecer dispersos en la memoria, pero mantiene un orden lógico interno. Su memoria es dinámica. (Es homogénea. Dinámica. De acceso secuencial. Lineal).

- **Características:**
  - 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:**
  - Crear una lista vacía.
  - Agregar un elemento al principio de una lista.
  - Agregar un elemento al final de una lista.
  - Insertar un nuevo elemento en una lista ordenada.
  - Recorrer una lista.
  - Acceder al k-ésimo elemento de la lista.
  - Eliminar un elemento de la lista.
  - Combinar dos listas ordenadas formando una sola ordenada (Merge de Listas).
- **Declaración:** {Type

```
Lista=^nodo;
Nodo=record
        Dato:tipo_elem;
        Sig:lista;
        End;}
```

LISTAS CIRCULARES

Es una lista enlazada en la cual el último elemento apunta al primero de la lista. La ventaja que tiene es que cada nodo de la lista es accesible desde cualquier otro nodo de ella. La desventaja es que se pueden producir bucles o lazos infinitos.

LISTAS DOBLES

Son listas enlazadas que tienen dos enlaces, cuyos nodos se encuentras enlazados a otros dos nodos por medio de criterios diferentes. Se puede recorrer en dos sentidos (ordenes) diferentes. La ventaja es que puede accederse siguiendo cualquiera de los dos órdenes, sin utilizar espacio extra y la desventaja es que ocupan más memoria por nodo que una lista simple.

- **Declaración:** {Type

```
Lista=^nodo;
Nodo=record
        Dato:…;
        Sig1:lista;
        Sig2:lista;
        End;
listaDoble=record
        orden1:lista;
```

```
                orden2:lista;
                end;}
```

LISTAS DOBLEMENTE ENLAZADAS

Cada nodo tiene un "siguiente" y un "anterior" los cuales consideran el mismo criterio. Los enlaces permitirán recorrer la lista en dos sentidos de manera tal que un sentido da la inversa del otro.

- **Declaración:** {Type

```
        Lista=^nodo;
        Nodo=record          listaDoble=record
            Dato:…;              pri:lista;
            Ant:lista;           ult:lista;
            Sig:lista            end;}
            End;
```

MERGE DE LISTAS

Es el proceso de combinar dos listas en una sola con el criterio ordenando de menor a mayor

- **Operación merge:** consiste en generar una nueva estructura de datos (arreglos, listas) ordenada a partir de la mezcla de dos o más estructuras de datos previamente ordenadas. El mecanismo de Merge consiste en comparar los elementos de cada una de las estructuras que se combinan y guardar el más pequeño en la estructura resultante. Luego se deberá avanzar en aquella estructura que contenía el elemento con valor más pequeño.

ARBOL

Es una estructura de datos que satisface tres propiedades:

1) Cada elemento del árbol se relaciona con cero o más elementos, a quienes llama hijos.
2) Si el árbol no está vacío, hay un único elemento al cual se llama raíz y que no tiene padre (predecesor), es decir, no es hijo de nadie.
3) Todo otro elemento del árbol posee un único padre y es un descendiente de la raíz.

- **Características:**
  - Es homogénea, todos los elementos son del mismo tipo.
  - Es dinámica, puede aumento o disminuir el tamaño de memoria en su ejecución.
  - Es no lineal, cada elemento puede tener 0, 1 o más sucesores.
  - Es de acceso secuencial.
- **Longitud del camino:** cantidad de aristas entre la raíz y un nodo particular.
- **Profundidad del nodo:** es la longitud del camino único entre la raíz y el nodo.
- **Altura del árbol:** es el camino más largo de la raíz a una hoja.
- **Árbol binario:** son los árboles que en cada nodo tienen como máximo dos links o nodos.
- **Árbol ternario:** son los árboles que en cada nodo tiene como máximo tres links o nodos.
- **Árbol N-ario:** cuando los nodos tienen como máximo N links o nodos.

ARBOLES BINARIOS DE BUSQUEDA

Clase especial de árboles binarios en el que existe algún orden sobre los datos almacenados en ellos. Mantiene sus datos de tal manera que siempre es posible recuperarlo en el orden dado.
Cada nodo tiene un valor que (a) es más grande que el valor de todos los nodos del subárbol izquierdo y (b) es menor que el valor de todos los nodos del subárbol derecho.
La utilidad más importante es la búsqueda (el tiempo medio es O(log N)).

- **Declaración:** {Type
  Árbol=^nodo;
  Nodo=record
  hijoIzq:árbol;
  elem:…;
  hijoDer:árbol;
  end;}
- **Operaciones:**
  - Inicializar
  - Insertar nodo nuevo
  - Borrar un nodo
  - Recorrer el árbol
  - Buscar
  - Imprimir contenido

- **Recorrido de un árbol:** permite desplazarse a través de un árbol en forma tal que cada nodo sea visitado una sola vez. Los métodos que existen son:
  - Recorrido enOrden.
  - Recorrido preOrden.
  - Recorrido postOrden.
- **Borrado de un nodo:** se deben considerar diferentes situaciones:
  - Si el nodo es una hoja. Simplemente se borra.
  - Si el nodo tiene un hijo. El nodo puede ser borrado después de que su padre actualice el puntero al hijo del nodo que se quiere borrar.
  - Si el nodo tiene dos hijos. Al borrar dicho nodo es necesario mantener la coherencia de los enlaces, además de seguir manteniendo la estructura como un árbol binario de búsqueda. La solución consiste en sustituir la información del nodo que se borra por el de una de las hojas, y borrar a continuación dicha hoja (la mayor de las claves menores al nodo que se borra o la menor de las claves mayores al nodo que se borra).

TAD (TIPO ABSTRACTO DE DATOS)

Un tipo abstracto de datos (TAD) es un tipo definido por el usuario que:

- Tiene un conjunto de valores y un conjunto de operaciones.
- Cumple con los principios de abstracción, ocultación de la información y se puede manejar sin conocer la representación interna.

Es decir, los TADs ponen a disposición del programador un conjunto de objetos junto con sus operaciones básicas que son independientes de la implementación elegida.

- **Características:**
  - Especificación de la representación del tipo.
  - Especificación de las operaciones permitidas para ese tipo
  - Desarrollo interno de cada una de las operaciones permitidas, con independencia para el usuario de cómo se implementan.
  - Encapsulamiento de todo, de manera que el usuario no pueda manipular los datos del objeto excepto por el uso de las operaciones definidas en el punto anterior.
- **Ventajas:**
  - Permite una mejor conceptualización y modelado del mundo real. Mejora la representación y comprensión.

- La abstracción de datos provee un camino para poner juntos los componentes de software que están relacionados.
- El uso de un TAD es sencillo porque no es necesario recordar detalles de implementación, solo es necesario conocer las operaciones.
- El ocultamiento de la información permite garantizar la integridad de los valores de un TAD. Solo las operaciones definidas por el TAD pueden manipular la estructura de datos utilizada.
- La separación de la definición del tipo de su implementación permite cambiar el módulo de implementación sin necesidad de modificar los programas que lo utilizan.

- **Roles:**
  - El Constructor del Tad debe:
    - Declarar la interface del TAD. (el tipo y operaciones)
    - Definir la representación del tipo exportado e implementar las operaciones del tipo declaradas en la interface.

  - El Usuario del Tad debe:
    - Indicar el nombre del TAD a utilizar.
    - Declarar las variables correspondientes al TAD.
    - Utilizar SOLO las operaciones permitidas para el tipo, es decir , las operaciones definidas en la interface.

PILAS (TAD PILA)

Es una estructura de datos que organiza los datos de la siguiente manera:
A partir de una dirección de memoria, los datos se almacenan sucesivamente como si fueran una colección ordenada de elementos (cartas, platos, libros, camisas, etc). El orden está asociado al orden de llegada a la estructura.

- **Características:**
  - Es homogénea, ya que almacena elementos del mismo tipo.
  - Es dinámica, ya que permite agregar y sacar elementos durante la ejecución del programa.
  - Es de acceso LIFO/UEPS (Last In FirstOut): los elementos se recuperan en orden inverso al que fueron almacenados. En cualquier momento se puede recuperar el objeto que se encuentra al tope de la pila (es decir, el último que fue guardado).
  - Es lineal, cada elemento de la estructura posee un elemento que le precede y uno que le sigue, excepto el primero y el último.
- **Operaciones:**
  - Crear una pila: st_create
  - Agregar un elemento: st_push
  - Sacar un elemento: st_pop
  - Conocer cuál es el elemento que está al tope de la pila: st_top
  - Conocer la cantidad de elementos de la pila: st_length
  - Determinar si una pila está vacía: st_empty

COLAS (TAD COLA)

Es una estructura de datos que organiza los datos de la siguiente manera:
A partir de una dirección de memoria, los datos se almacenan sucesivamente manteniendo su orden de llegada (alumnos esperando para realizar una inscripción, personas frente a una ventanilla de cobro, etc). En cualquier momento se puede sacar el objeto que se encuentra primero en la estructura (es decir, el primero que fue guardado).

- **Características:**
  - Es homogénea, ya que almacena elementos del mismo tipo.
  - Es dinámica, ya que permite agregar y sacar elementos durante la ejecución del programa.

    - Es de acceso FIFO/PEPS (First In FirstOut): los elementos se recuperan en orden inverso al que fueron almacenados. En cualquier momento se puede recuperar el elemento que se encuentra en el frente de la cola (es decir, el primero que fue guardado).
    - Es lineal, cada elemento de la estructura posee un elemento que le precede y uno que le sigue, excepto el primero y el último.
- **Operaciones:**
    - Crear una cola: q_create
    - Agregar un elemento a la estructura: q_push
    - Sacar un elemento de la estructura: q_pop
    - Conocer cuál es el elemento que está en el frente de la cola: q_top
    - Conocer cuál es el elemento que está en el fondo de la cola: q_bottom
    - Conocer la cantidad de elementos de la cola: q_length
    - Determinar si una cola está vacía: q_empty

PROGRAMACION ORIENTADA A OBJETOS

- **Clase:** una clase se puede definir como un modelo que define los atributos y métodos comunes a todos los objetos que comparten las mismas características. Es una "plantilla genérica" para un conjunto de objetos de similares características. Compuesta por un conjunto de atributos, la estructura de los atributos y un conjunto de métodos para el comportamiento.
- **Objeto:** un objeto es una unidad que contiene atributos específicos (datos) y responde a los métodos (comportamiento) de su clase, para operar sobre esos atributos. Es una instancia de una clase y se comunican entre sí mediante mensajes.
- **Métodos:** Un método es una subrutina cuyo código es definido en una clase y puede pertenecer tanto a una clase, como es el caso de los métodos de clase o estáticos, como a un objeto, como es el caso de los métodos de instancia. Consiste generalmente de una serie de sentencias para llevar a cabo una acción.
- **Herencia:** Es un mecanismo que le permite a un objeto heredar propiedades de otra clase de objetos. La herencia permite a un objeto contener sus propios procedimientos o funciones y heredar los mismos de otros objetos.
- **Polimorfismo:** Se refiere a la propiedad por la que es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos. El único requisito que deben cumplir los objetos que se utilizan de manera polimórfica es saber responder al mensaje que se les envía.

PROGRAMACION CONCURRENTE

Concurrencia es la característica de los sistemas que indica que múltiples procesos/tareas pueden ser ejecutados al mismo tiempo y pueden cooperar y coordinarse para cumplir la función del sistema.

- **Comunicación de procesos en un entorno concurrente:**
    - Pasaje de mensajes → enviar y recibir.
        - Es necesario establecer un canal (lógico o físico) para transmitir información entre procesos.
        - También el lenguaje debe proveer un protocolo adecuado.
        - Para que la comunicación sea efectiva los procesos deben "saber" cuándo tienen mensajes para leer y cuando deben trasmitir mensajes.
    - Memoria compartida → depositar y sacar.
        - Los procesos intercambian información sobre la memoria compartida o actúan coordinadamente sobre datos residentes en ella.
        - Lógicamente no pueden operar simultáneamente sobre la memoria compartida, lo que obliga a bloquear y liberar el acceso a la memoria.
        - La solución más elemental es una variable de control que habilite o no el acceso de un proceso a la memoria compartida