**Algoritmos, Datos y Programas.**

**Clase 1: Conceptos básicos**

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

**Etapas para resolver un problema**

1. **Análisis**: representa la abstracción del problema real en un modelo que permitiría su resolución por computadora. Se analiza el problema en su contexto del mundo real. Deben obtenerse los requerimientos del usuario. El resultado del análisis del problema es un modelo preciso del ambiente del problema y del objetivo a resolver.

\*Esta etapa no depende del lenguaje de programación.

1. **Diseño:** en esta etapa pensaremos en la forma de descomponer las partes que integran la solución. Suponiendo que el problema es computable, a partir del modelo se debe diseñar una solución. En el paradigma procedural, esta etapa involucra entre otras tareas la **modularización** del problema, considerando la descomposición del mismo y los datos necesarios para cumplir su objetivo.

*{La descomposición funcional de todas las acciones que propone el modelo nos ayudará a reducir la complejidad, a distribuir el trabajo y en el futuro a re-utilizar los módulos.}*

Esta etapa involucra la especificación de los algoritmos:

Cada uno de los módulos del sistema diseñado tiene una función que podemos traducir en un algoritmo, que puede no ser único. La elección del algoritmo adecuado para la función del módulo es muy importante para la eficiencia posterior del sistema de software.

\*Esta etapa no depende del lenguaje de programación.

1. **Implementación:**requiere escribir algoritmos en un lenguaje de programación y elegir la representación de los datos. Se debe probar en ejecución cada uno de los módulos, y luego una prueba del programa completo.

Esta etapa involucra la escritura de programas:

Un algoritmo es una especificación simbólica que debemos convertir en un programa escrito en un lenguaje de programación concreto.

\*Esta etapa depende del lenguaje de programación.

1. **Verificación:** una vez que se tiene un programa escrito y depurado de errores de sintáxis, debemos verificar que su ejecución conduce al resultado deseado, con datos representativos del problema real.

**Los programas deben tener las siguientes características:**

1. Para la audiencia humana:

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

1. Para la CPU de la computadora:

* Debe contener instrucciones válidas.
* Deben terminar.
* No deben utilizar recursos inexistentes.

**Algoritmo**:es la especificación rigurosa de la secuencia de pasos (instrucciones) a realizar sobre un autómata para alcanzar un resultado deseado en un tiempo finito.

\*El algoritmo debe ser claro y unívoco.

\*El algoritmo debe comenzar y terminar. El número de instrucciones debe ser finito.

\*Si el autómata es una computadora, debemos escribir el programa en un lenguaje entendible y ejecutable por ésta.

**Dato*:***es una representación de un objeto del mundo real mediante la cual podemos modelizar aspectos del problema que se quiere resolver con un programa sobre una computadora.

Representar datos requiere una transformación desde el mundo real a alguna forma de **representación binaria** que pueda ser interpretada por la computadora.

* **Constantes**: datos que no cambiandurante la ejecución del programa*.*
* **Variables***:* datos quedurante la ejecución del programa pueden cambiar.

**Programa**: Es un 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).

Un programa es **correcto** si cumple lo solicitado en los **requerimientos*.***

Los **programas** se escriben en un **lenguaje de programación** siguiendo un conjunto de reglas sintácticas y semánticas.

Componentes básicas de un programa:

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

Programa= instrucciones + datos.

**¿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 máquina a emplear.
* ***Definir el conjunto de instrucciones*** (en el lenguaje elegido)cuya ejecución ordenada conduce a la solución.

***Pre-condición***es la información que se conoce como verdadera antes de iniciar el programa (ó módulo).

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

**Tipos de Datos.**

Un tipo de dato es una clase de datos ligados a un conjunto de operaciones para crearlos y manipularlos, y se caracterizan por:

* Un rango de valores posibles.
* Un conjunto de operaciones realizables sobre ese tipo.
* Una representación interna.
* Los tipos de datos **simples** toman un único valor, en un momento determinado, de todos los permitidos para ese tipo.
* 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.
* Numérico. (entero o real)
* Carácter.
* Booleano.
* Definidos por el usuario: permiten definir nuevos tipos de datos a partir de los tipos simples. (subrango)
* Los tipos de datos **compuestos** pueden tomar varios valores a la vez que guardan alguna relación lógica entre ellos. (string, conjunto)

**Tipo de dato numérico.**

-El 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 ella son finitos, por esto se deduce que existe un número entero máximo y otro mínimo.

-El tipo de dato real es una clase de dato numérico que permite representar números decimales. Consiste en definir cada número como una mantisa (parte decimal) y un exponente (posición de la coma).

Operaciones:

* + suma (+) 🡪 enteros, reales.
  + resta (-) 🡪 enteros, reales.
  + multiplicación (\*)🡪 enteros, reales.
  + división (/) 🡪 reales
  + división entera (div) y módulo (mod)🡪 enteros

El orden de precedencia para la resolución, ya conocido, es:

**1)**operadores \*, / **2)** operadores +, - **3)** operadores div y mod.

* Además de los operadores matemáticos mencionados, el tipo de dato numérico posee **operadores relacionales** que permiten comparar valores pertenecientes al mismo tipo de dato. Dichas relaciones son la **igualdad** (=), **desigualdad** (<>) y de **orden** (<, <=, >, >=). El **resultado** es del tipo de dato lógico. (Verdadero o Falso).

**Tipo de dato lógico.**

El tipo de dato lógico permite representar datos que pueden tomar solamente uno de dos valores. Este tipo de dato también es llamado tipo de dato boolean.

Dichos valores son:

**verdadero** (*true*) ; **falso** (*false*)

 Se utiliza en situaciones donde se representan dos alternativas de una condición.

Los **operadores lógicos** o **booleanos** básicos son:

**negación** (*not*) ; **conjunción** (*and*); **disyunción** (*or*).

El resultado de estas operaciones es el correspondiente a las conocidas tablas de verdad.

Existe un orden de precedencia para los operadores lógicos:

**1)*or* 2)*and* 3)*not***

**Tipo de dato carácter**

El tipo de dato carácter representa un conjunto finito y ordenado de caracteres que la computadora reconoce.Un dato de tipo carácter contiene solo un carácter.

Los caracteres que reconocen las computadoras no son estándar. Sin embargo, este conjunto de valores se normalizó, entre otros, por un estándar llamado ASCII, el cual permite establecer un **orden de precedencia** entre los mismos.

**Caracteres especiales**:‘!’, ‘#’, ‘$’, ‘%’, …

**Dígitos:** ‘0’, ‘1’, ‘2’, …, ‘8’, ‘9’

**Letras mayúsculas:** ‘A’, ‘B’, ‘C’, …, ‘Y’, ‘Z’

**Letras minúsculas:** ‘a’, ‘b’, ‘c’, …, ‘y’, ‘z’

- Se debe tener en cuenta que no es lo mismo el valor entero 0 que el símbolo carácter ‘0’.

-Dos valores de tipo carácter se pueden comparar por =, <>, >, <, >=, <=. El resultado de cualquiera de ellos es un valor de tipo de dato lógico.

Los diferentes tipos de datos deben especificarse y a esta especificación dentro de un programa se la conoce como **declaración**. Una vez declarado un tipo podemos asociar al mismo ***variables***, es decir ***nombres simbólicos*** que pueden tomar los valores característicos del tipo.

**Variables:** Una variable es una zona de memoria cuyo contenido va a ser de alguno de los tipos mencionados anteriormente. La dirección inicial de esta zona se asocia con el nombre de la variable. Llamaremos identificadores a los nombres descriptivos que se asocian a los objetos (constantes y variables) para abstraer dentro del programa su dirección real en memoria y su valor.En Pascal, los identificadores están formados por letras, dígitos en cualquier orden y algunos símbolos especiales (excepto el primer carácter que debe ser una letra). En **Pascal**, las constantes deben ser declaradas antes de ser usadas. La declaración de constantes **en Pascal** se realiza mediante la palabra clave **const**, de la forma:

**Const** *nombre* = *valor*

Donde nombre es el identificador que representa el nombre de la constante. El tipo de dato de la constante queda definido implícitamente por el tipo de dato de valor.

\*Las variables en Pascal se declaran utilizando la palabra clave **var**, de la forma:

**var**

*nombre\_de\_variable* :*tipo\_de\_variable;*

o, si hay varias variables del mismo tipo, se declaran separadas por coma y se le asocia el tipo de dato, una sola vez. Por ejemplo:

**var**

*nombre\_de\_variable\_1, …, nombre\_de\_variable\_n*: *tipo\_de\_variable;*

* **Lenguaje fuertemente tipado:**Algunos lenguajes, como Pascal, exigen que se especifique a qué tipo pertenece cada una de las variables. Verifican que el tipo de los datos asignados a esa variable se correspondan con su definición.

No se permite violación de datos. 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.

* **Lenguaje auto-tipado:** clase de lenguajes que verifica el tipo de las variables según su nombre.
* **Lenguaje dinámicamente tipado:** lenguajes que permiten que una variable tome valores de distinto tipo durante la ejecución de un programa.

**Clase 2: Estructuras de control.**

**Operaciones de entrada:**

Read: se usa para leer datos (por defecto desde teclado) y asignarlos a las variables correspondientes.

Readln: es igual a la instrucción Read pero hace que la siguiente sentencia Read comience leyendo en la siguiente línea.

**Operaciones de Salida:**

Write: se usa para mostrar el contenido de una variable, por defecto en pantalla. Pueden ser de tipo entero, real, char. Los datos a mostrar si son más de uno deben ir separados por coma.

Writeln: es igual a la instrucción Write, pero hace que la siguiente sentencia Write muestre el contenido de la variable en la siguiente línea.

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

* **Estructura de control: decisión.**

En un algoritmo representativo de un problema real es prácticamente imposible que todo sea secuencial. Es necesario tomar ***decisiones*** en función de los datos del problema.

La estructura básica de decisión entre ***dos*** alternativas es la que se representa simbólicamente*:*

If (condición) thenbegin

…

End;

* **Estructura de control: selección.**

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.

Case variable decisión of

posibilidad1: Acciones;

posibilidad2: Acciones;

…………………...

else otras acciones

End;

* **Estructura de control: repetición.**

Es una extensión natural de la secuencia. Consiste en ***repetir N veces un bloque de acciones.*** Este número de veces que se deben ejecutar las acciones es fijo y conocido de antemano.

forindice := valorinicialtovalorfinaldobegin

Acciones

End**;**

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). Los incrementos ó decrementos y testeos son implícitos.

* **Estructura de Control: iteración.**

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.

* **Pre-condicionales**: Las estructuras iterativas pre-condicionales primero evalúan la condición y si es verdadera se ejecuta el bloque de acciones. Dicho bloque se puede ejecutar 0, 1 ó más veces.Importante: el valor inicial de la condición debe ser conocido o evaluable antes de la evaluación de la condición.

While(condición)do begin

Acciones

End**;**

* **Post-condicionales:** Las estructuras iterativas post-condicionales primero ejecutan el bloque de acciones y luego evalúan la condición. Dicho bloque se puede ejecutar 1 o más veces.

repeat

Acciones

until( condicion);

**Clase 3: Calidad de un programa. Corrección. Eficiencia**

**Calidad de un programa**

Factores que determinan la calidad del software

Se clasifican en tres grupos:

1. **Operatividad del producto**: características operativas

* ***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

1. **Revisión del producto**: capacidad para soportar cambios

* ***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

1. **Transición del producto**: adaptabilidad a nuevos entornos

* ***Portabilidad***(¿Podré usarlo en otra máquina?) El esfuerzo requerido para transferir la aplicación a otro hardware o sistema operativo
* ***Reusabilidad***(¿Podré utilizar alguna parte del software en otra aplicación?) Grado en que las partes de una aplicación pueden utilizarse en otras aplicaciones
* ***Interoperabilidad***(¿Podrá comunicarse con otras aplicaciones o sistemas informáticos?) El esfuerzo necesario para comunicar la aplicación con otras aplicaciones o sistemas informáticos

**Características que hacen a la calidad de un programa:**

* **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.**Un programa bien documentado será fácil de leer y mantener.**

Cuando se realiza el mantenimiento de un programa no sólo se actualiza el código, sino también los comentarios del programa. Debe quedar claro que los comentarios no son un agregado al programa sino que son parte del mismo.

**CORRECCIÓN**

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.
* Para medir si un programa es correcto se debe probar con datos reales que permitan verificar su función.

**EFICIENCIA**

Definiremos *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.

Se deben tener en cuenta:

* Cantidad de operaciones que se ejecutan
* Estructuras de control utilizadas

Y a partir de este análisis

* ¿Cual de los programas resulta más eficiente?
* ¿En qué basa la respuesta?

**Clase 4: Tipos de datos definidos por el usuario.**

Un aspecto muy importante en los lenguajes de programación es la capacidad de especificar y manejar datos no estándar, indicando valores permitidos, operaciones válidas y su representación interna. Esto permite:

* + - 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.
    - Límites preestablecidos sobre los valores posibles que pueden tomar las variables que corresponden al tipo de dato.

Un **tipo de dato definido por el usuario** es aquel que no existe en la definición del lenguaje, y el programador es el encargado de su especificación. Este mismo 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.

En Pascal, los tipos deben ser declarados antes de ser usados.La declaración de tipos se hace a través de la palabra clave TYPE de la siguiente forma:

TYPE identificador = tipo;

**Identificador**: Nombre con que se conocerá al tipo de dato en el programa.

**Tipo**: Puede ser un tipo estándar o alguno de los tipos de datos definidos por el usuario.

**Ventajas de declarar tipos:**

* Flexibilidad: en el caso de ser necesario modificar la forma en que se representa el dato, sólo se debe modificar una declaración en lugar de un conjunto de declaraciones de variables.
* Documentación**:** se pueden usar como identificador de los tipos, nombres autoexplicativos, facilitando de esta manera el entendimiento y lectura del programa.
* Seguridad**:** se reducen los errores por uso de operaciones inadecuadas del dato a manejar, y se pueden obtener programas más confiables.

**Tipo de dato definido por el usuario: Subrango.**

Algunas veces el tipo de dato puede tomar alguno de todos los valores que nos brinda un tipo ordinal, es decir un subrango de los mismos.

Un tipo de dato subrango es un tipo ordinal que consiste de una sucesión de valores de un tipo ordinal tomado como base. Las operaciones de un tipo de dato subrango se heredan del tipo base.

DECLARACION:

**Program ejemplo;**

**Type** mayúsculas **=** ‘A’.. ‘Z’;

siglo\_pasado**=** 1900 .. 1999;

**Var**  letra: mayúsculas;

año: siglo\_pasado;

¿Por qué es útil el tipo subrango?

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

**Tipo de dato compuesto: Conjunto**

Desde el punto de vista informático un tipo conjunto representará 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. Se puede tener conjunto de enteros, caracteres ó subrango. El número máximo de elementos de un conjunto en Pascal es 255. Un conjunto NO admite operaciones de lectura y escritura.

DECLARACION:

**Program ejemplo;**

**Type** identificador**= set of** tipo-ordinal;

**Var** nombre**:** identificador;

* **Operaciones**
* **Asignación:** A una variable de tipo conjunto se le puede asignar valores escribiendo sus elementos consecutivamente, encerrados entre corchetes y separados por comas, en la sección de instrucciones ejecutables.

**Begin**

ConjLetras:= [‘a’, ’b’];

conjLetras2:= [‘a’ .. ’z’];

conjLetras3:= [ ]; *{conjunto vacio}*

ConjEdad:= [1, 3, 20, 55];

**End;**

* **Unión:**se representa con el signo + y da como resultado otro conjunto. En este conjunto resultado aparecen los elementos de los dos conjuntos y aquellos elementos repetidos aparecen una vez.

**Begin**

ConjLetras:= [‘a’ , ’b’] + [‘j’ , ’k’];

conjLetras2:= conjLetras + [‘l’ , ’m’];

**End;**

* **Intersección:** se representa con el signo \* y da como resultado otro conjunto. En el conjunto resultado aparecen solamente los elementos comunes a los dos conjuntos.

Begin

conjLetras := [‘a’ , ’b’] + [‘j’ , ’k’];

conjLetras2 := [‘a’] + [‘l’ , ’m’];

conjLetras3 := conjLetras \* conjLetras2;

End;

* **Diferencia:** se representa con el signo - y da como resultado otro conjunto. Este conjunto resultado contiene los elementos que están en el primer conjunto y no están en el segundo.

Begin

conjLetras := [‘a’ , ’b’ , ’c’];

conjLetras2 := [‘b’ , ’d’ , ’e’];

conjLetras3 := conjLetras - conjLetras2;

End;

* **Pertenencia:** se representa con el operador **in** y da como resultado un valor lógico. Esta operación devuelve verdadero si el elemento está en el conjunto y falso en caso contrario.

Begin

conjLetras := [‘a’, ’ b’ , ’c’];

res:= ‘a’ in conjLetras;

End;

Se pueden usar los operadores **relacionales** para determinar si un conjunto está incluido en otro (<=), si son distintos (<>) ó iguales (=).

**Tipo de dato compuesto: string**

En la mayoría de los lenguajes existe un tipo estándar o definible por el usuario que es la cadena de caracteres llamado string. Vimos que un carácter es una letra, número ó símbolo.

Cuando se trabaja con el tipo de dato string, se tienen k caracteres tratados como una sola variable (donde k es la longitud del string).

Un tipo de dato **string** es una sucesión de caracteres de un largo determinado, que se almacenan en un área contigua de la memoria.

DECLARACION

**TYPE** identificador **=string**[longitud];

Longitud es el número máximo de cantidad de caracteres que puede contener el dato. En Pascal cuando no se especifica la longitud ese identificador podrá contener como máximo 255 caracteres.

**Type** hilera**= string** [10];

**Var**h1, h2, h3**:**hilera;

**Operaciones**

* **Asignación:** para asignar valor a una variable de tipo de dato string se hace igual que si fuera una variable de tipo carácter, **:=**.

Si se le asigna mayor cantidad de caracteres que lo declarado como longitud máxima, los últimos a partir de esa longitud se pierden y se dice que la hilera de caracteres “se trunca”.

**Program** uno**;**

**Type** hilera**= string** [20];

**Var**cad1: hilera;

**Begin**

cad1:= ‘buenos días!’;

* **Operadores Relacionales:** los strings pueden compararse por =, <>, <=, =>**.**
* Si las cadenas que se comparan son de igual longitud **y** contienen los mismos símbolos, en el mismo orden, el resultado de la operación es verdadero.
* Estos operadores realizan la comparación carácter por carácter. Si tienen distinta longitud el resultado de la comparación es falso.
* **Operaciones de Entrada y Salida: read y write**

El tipo de dato string admite las operaciones Read y Write de Pascal.

Si la cadena ingresada supera la longitud declarada para el dato string entonces serán descartados los caracteres que se encuentran mas a la derecha.

**Clase 5: Modularización y comunicación entre módulos**

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

Cuando se descompone un problema en subproblemas, deben ser de forma tal 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

**Descomposición**

Los **Módulos** representan tareas específicas bien definidas que deben comunicarse entre sí adecuadamente y cooperar para conseguir un objetivo común. Se parte de la descomposición de problemas grandes a problemas pequeños.

* Cada módulo encapsula acciones o tareas.
* Cada módulo debe representar los datos relevantes del subproblema a resolver.

**TOP DOWN**

Ir de lo general a lo particular. Dividir…conectar… y verificar.

**Ventajas de la modularización:**

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 así el tiempo de desarrollo global del sistema y aumentando las posibilidades de producción de software
* **Reusabilidad:** Un objetivo fundamental de la Ingeniería de Software es la reusabilidad, es decir la posibilidad de utilizar repetidamente el producto de software desarrollado.

Naturalmente la descomposición funcional que ofrece la modularización favorece el reuso*.*

* **Facilidades 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.
* **Facilidades de crecimiento del sistema:** Los sistemas de software reales crecen, es decir 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:**Un efecto de la modularización es una mayor claridad para leer y comprender el código fuente. El ser humano maneja y comprende con mayor facilidad un número limitado de instrucciones directamente relacionadas.

**Recursos de los lenguajes de programación para especificar la modularización**

* Subroutine; Module; Procedure (provisto por Pascal); Function (provisto por Pascal); Package;Class, etc.

**Modularización: procedimiento**

Un procedimiento se puede definir como un conjunto de instrucciones que realizan una tarea específica y como resultado puede retornar 0, 1 o más valores.

**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 sólo podrá ser usada por ese módulo. Si este módulo contiene a su vez otros módulos, entonces esa variable puede ser también usada por todos los módulos interiores.

**¿COMO SE COMUNICAN LOS DATOS ENTRE MODULOS Y PROGRAMA?**

Por variables globales y parámetros. El uso de variables globales para la comunicación entre módulos presenta las siguientes desventajas:

* Demasiados identificadores.
* No se especifica la comunicación deseada entre los módulos.
* Conflictos de nombres de identificadores utilizados por diferentes programadores.
* Posibilidad de perder integridad de los datos, al modificar involuntariamente en un módulo datos de alguna variable que luego deberá utilizar otro módulo.

La solución a estos problemas ocasionados por el uso de variables globales es una combinación de ocultamiento de datos (Data Hiding) y uso de parámetros. El ocultamiento de datos 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.

La comunicación externa de un módulo con el resto del sistema puede no existir, pero si existe esa comunicación debería hacerse a través de los parámetros (datos de entrada y datos de salida).La comunicación entre módulos usando parámetros puede ser por valor o por referencia.

* **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.

Un parámetro por valor debe ser tratado como una variable local del módulo. La utilización de este tipo de parámetros puede significar una utilización importante de memoria.

* **Parámetro por referencia**: es un dato 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.

Los parámetros por referencia operan directamente sobre la dirección de la variable original, en el contexto del módulo que llama. Esto significa que no requiere memoria local.

El número y tipo de los argumentos utilizados en la invocación a un módulo deben coincidir con el número y tipo de parámetros del encabezamiento del módulo.

* Al modularizar es muy importante obtener independencia funcional de cada módulo. Esto disminuye el acoplamiento con el resto del sistema y por lo tanto reduce el impacto de las fallas y modificaciones.
* La comunicación entre módulos debe acotarse a intercambio de datos por argumentos y parámetros. Siempre que se pueda deben utilizarse parámetros por valor.
* Los procedimientos son subprogramas que interactúan en el espacio de datos del módulo que los invoca a través de parámetros por referencia.
* En todos los casos deben evitarse las variables globales.

**Modularización: funciones.**

Una función se puede definir como un módulo que realiza una única tarea y devuelve un solo valor de tipo simple.

* La lista de parámetros es **opcional** (como en los procedimientos)
* Tipo: es el tipo del dato que devolverá la función. Debe ser de tipo simple.
* La última instrucción ejecutada deberá **obligatoriamente** asignarle un valor al nombre de la función (del tipo especificado)
* Respecto de las operaciones de lectura y escritura, no es aconsejable introducirlas como parte de la función.
* Se pueden invocar: dentro de un if, o de un while, o asignarla a una variable o dentro de un write.
* Pueden recibir sólo parámetros de entrada.
* Las **funciones** pueden pensarse como operadores definidos por el usuario, que **reciben** variables (parámetros por valor) y **producen** un resultado único.

**Clase 6: Estructura de datos Registro.**

**Tipo de dato estructurado:** permite al programador definir un tipo de dato al que se asocian diferentes datos que tienen valores lógicamente relacionados y asociados bajo un nombre único.

Una **estructura de datos** es un conjunto de variables (que podrían ser de distinto tipo) relacionadas entre sí y que se puede operar como un todo, bajo un nombre único.

**Las estructuras de datos pueden clasificarse:**

* De acuerdo al tipo de dato que la compone:
* **Homogéneas**: los datos que la componen son todos del mismo tipo.
* **Heterogéneas**: los datos que la componen son de distinto tipo.
* De acuerdo a la ocupación de memoria:
* **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 esvariable, y por lo tanto la cantidad de memoria ocupada puede cambiar durante la ejecución de un programa.
* De acuerdo al acceso a sus elementos:
* **De acceso secuencial:** para acceder a un elemento particular se debe respetar un orden predeterminado, por ejemplo, pasando por todos los elementos que le preceden.
* **De acceso directo: s**e puede acceder a un elemento particular, directamente, sin necesidad de pasar por los anteriores a él, por ejemplo, indicando una posición.
* De acuerdo a la relación entre sus elementos:
* **Lineales**: a cada elemento le sigue uno y le precede uno, solamente.
* **No lineales:** para cada elemento pueden existir 0, 1 ó mas elementos que le suceden y 0, 1 ó mas que le preceden.

El **registro** es uno de los tipos de datos estructurados, que permiten agrupar diferentes clases de datos en una estructura única. Una manera natural y lógica de agrupar la información de cada empleado en una sola estructura es declarar un tipo REGISTRO asociando el conjunto de datos de cada empleado. Cada dato que compone el Registro se denomina campo.

Un registro es una estructura de datos que cumple con:

* Se identifica el nombre del tipo como registro (RECORD).
* Se especifica el nombre y tipo de los campos individuales que componen el tipo. La lista de campos se encierra entre las palabras claves record y end.
* Cada uno de los campos tiene un identificador. Los campos pueden ser nombrados individualmente, como variables ordinarias.
* Los campos pueden ser de cualquier tipo predefinido o definidos por el usuario (estáticos).
* Para acceder a los campos de un registro, se necesita especificar tanto el nombre del registro como el del campo que interesa. Esto se denomina calificar al campo.

En Pascal se hace de la siguiente forma: Nombre-Registro.nombre-campo.

Dado que el Registro es un tipo de dato estructurado, las operaciones deberán aplicarse a cada uno de los campos que lo componen. Como los campos de un registro son variables de algún tipo, las operaciones posibles sobre un campo son las permitidas para el tipo de dato correspondiente.

La única **operación** permitida en registros es la de Asignación, siempre y cuando, los mismos estén definidos del mismo tipo. No pueden realizarse comparaciones entre registros completos, es decir para saber si dos registros son iguales, se debe evaluar cada uno de los campos.

Cuando se trabaja con registros, hay ocasiones en que el acceso a los campos a través de la calificación suele ser tediosa.

Para solucionar este inconveniente, el lenguaje Pascal provee una sentencia **WITH** que permite que la variable de tipo registro sea nombrada una vez, y luego los campos del registro sean accedidos directamente.

**Clase 7: Estructura de datos: Arreglos.**

Un arreglo (ARRAY) es una estructura de datos compuesta que permite acceder a cada componente por una variable índice, que da la posición de la componente dentro de la estructura de datos. Es una colección de elementos que se guardan consecutivamente en la memoria y se pueden referenciar a través de un índice.

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 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**.
* Precisamente por ser **estática**, permite el acceso rápido a sus componentes a través de la **variable índice**( que tiene que ser de tipo ordinal ) y que puede verse como el **desplazamiento desde la posición inicial de comienzo de la estructura**.
* Un arreglo con un solo índice se denomina ***vector.***
* La cantidad total de memoria depende de la cantidad de componentes y del tipo de dato de cada uno.

DECLARACION.

La declaración del tipo vector **debe indicar la cantidad de elementos y el tipo de datos de los elementos.**

Vector =**Array [índice] of** tipo\_elementos;

**Índice**:

Es posible indexar los elementos por un índice que corresponde a **cualquier tipo ordinal**:

* Entero
* Carácter
* subrango

**Elementos**:

Los elementos de un arreglo pueden pertenecer a **cualquier tipo de datos de alocación estática**:

* Entero, Real, Lógico, Carácter
* String
* Registros
* Otro arreglo

**Operaciones:**

* + Asignación
  + Lectura / Escritura
  + **Recorridos**: La operación de Recorrido en un vector consiste en recorrer el vector de manera total o parcial, para realizar algún proceso sobre sus elementos.La operación de Recorrido Total, implica analizar todos los elementos del vector, lo que lleva a recorrer completamente la estructura. La operación de Recorrido Parcial, implica analizar los elementos del vector, hasta encontrar aquel que cumple con lo pedido. Puede ocurrir que se recorra todo el vector.
  + Cargar datos en un vector
  + Búsqueda secuencial de un elemento
  + Agregar elementos al final
  + Insertar elementos
  + Borrar elementos
  + Búsqueda dicotómica de un elemento
  + Ordenación de los elementos
* **Dimensión Física del vector:** se especifica en el momento de la declaración; determina su ocupación de memoria. La cantidad de memoria no variará durante la ejecución del programa.
* **Dimensión Lógica del vector:** se determina cuando se cargan contenidos a los elementos del arreglo. Indica la cantidad de posiciones de memoria ocupadas con contenidos cargados desde la posición inicial en forma consecutiva.

**Operaciones de Búsquedas en Vectores.**

El proceso de ubicar información particular en una colección de datos es conocido como método de búsqueda. La colección de datos puede estar ordenada, o sin orden.

**Búsqueda lineal o secuencial:** (Arreglos sin orden)

* Se aplica cuando los elementos no tienen orden.
* Requiere excesivo consumo de tiempo en la localización del elemento.
* Número medio de comparaciones (dimL + 1) /2
* Es ineficiente a medida que el tamaño del arreglo crece.

**Búsqueda en arreglos ordenados:**

* Recorrido secuencial:
* Se aplica cuando los elementos tienen orden.
* La búsqueda comienza desde el principio y se avanza por la estructura de manera secuencial y de a uno hasta que encuentro el número buscado o hasta que encuentro uno mayor.
* Búsqueda Dicotómica :
* Se aplica cuando los elementos tienen orden.
* Se compara el valor buscado (x) con el ubicado en el medio del vector (a):

1. Si el elemento ubicado al medio del vector es igual a x, entonces la búsqueda termina.
2. Si no es el valor buscado, debería quedarse con la mitad del vector que conviene, para seguir la búsqueda. Este paso se repite tantas veces hasta que se acaba el vector o encuentro el valor.

* Cada vez que se toma la mitad del arreglo, se va disminuyendo el tamaño del mismo.
* El proceso termina cuando encuentro el elemento, o cuando el vector se hace tan pequeño que no quedan más elementos, y por lo tanto se puede deducir que el elemento no se encuentra en el vector.

**Ordenación de Vectores.**

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

* **Método de Selección**: Supongamos que se quiere ordenar el arreglo de menor a mayor:

\*en cada pasada se elige un mínimo (por ejemplo) y se lo ubica en el lugar que le corresponde.

\* n-1 pasadas.

Se pregunta por cada elemento: ¿Quién es el menor de mi derecha?

-Si el menor es el actual, se pasa a analizar el siguiente elemento, ya que es el más chico, no hay otros más chicos en su derecha.

-Si el mayor es el actual, se debe guardar el menor de su derecha en una variable auxiliar. El actual pasa a la posición del menor y el auxiliar va a pasar a la posición del actual.

* **Método de Intercambio:** Se comparan e intercambian datos adyacentes según corresponda

\*n-1 pasadas

\*Si se quiere ordenar el vector de menor a mayor:

\*Al final de cada pasada el mayor viaja y queda al fondo del arreglo (no es necesario re - compararlo)

Respecto del análisis teórico, el método de intercambio es “peor” que el de selección.

Para cada elemento, se pregunta ¿Cómo soy respecto de mi vecino, mayor o menor?

Si el actual es menor se pasa al siguiente elemento

Si el actual es mayor, se debe guardar el siguiente en un auxiliar; el auxiliar pasaría a la posición del actual; y el actual pasaría a la posición del siguiente.

* **Método de Intercambio con centinela:**

Podemos mejorar la eficiencia del método haciendo uso de una variable lógica o centinela

\* Si no hay intercambio, el arreglo ya está ordenado. Esta variable lo que hace es decir si hubo intercambio o no. Si hubo intercambio es que no está ordenado todavía.

* **Método de Inserción:**

Se parte de una lista de dos ítems y se ordena.

En cada pasada se agrega un ítem y se inserta en la posición correspondiente en el arreglo ordenado.

**Estructura de datos: Matrices.**

Una matriz es una estructura de datos compuesta que permite acceder a cada componente utilizando dos índices (fila y columna).Estos índices permiten ubicar un elemento dentro de la estructura. El primer índice indica la fila y el segundo índice indica la columna. Un tipo de dato Matriz es una colección ordenada e indexada de elementos.

**Características importantes:**

* Homogénea
* Estática
* Indizada
* Lineal

Matriz =**Array**[fila,columna] **of** tipo\_elementos;

Fila, columna: es posible indexar los elementos por un índice que corresponde a **cualquier tipo ordinal**:

* Entero
* Carácter
* subrango

Tipo de elementos: Los elementos de un arreglo pueden pertenecer a **cualquier tipo de datos**:

* Enteros
* Reales
* Caracteres
* Registros
* String
* Otro arreglo

Operaciones:

* Cargar una matriz
* Recorrer una matriz mostrando sus elementos
* Buscar un elemento determinado
* Eliminar una fila ó columna en una matriz
* Insertar una fila ó columna en una matriz

**Clase 8: Recursión.**

* El diseño de una solución involucra: El desarrollo de los algoritmos y la definición de las estructuras de datos
* Las Metodologías para resolver problemas:

1. Diseño Top Down
2. Recursión
3. Abstracción de Datos

* Una solución recursiva resuelve un problema por resolución de instancias más pequeñas del mismo problema.
* Esta solución se llama a sí misma
* Existe un caso distinto que se resuelve de manera particular y directa (sin recursión). Este caso se denomina Caso Base o Caso Degenerado

**Características de la Recursión**

Cuando se construye una solución recursiva se deben tener en cuenta:

* ¿Cómo definir el problema en términos de un problema más pequeño del mismo tipo?
* ¿Cómo será disminuido el tamaño del problema en cada llamado recursivo?
* ¿Qué instancia del problema servirá como **caso base**?
* Algunos problemas pueden resolverse con la misma facilidad usando tanto recursión como iteración.
* Las funciones y procedimientos no recursivos pueden ejecutarse más rápidamente y utilizar la memoria de manera más eficiente que los recursivos.
* Usada de manera apropiada, la recursión es una herramienta muy poderosa para la resolución de problemas para los cuales no es sencillo encontrar una solución no recursiva.

**Clase 9: Punteros.**

**Tipos de Alocación de Memoria**

* **Alocación Estática** (stack): variables estáticas.

\* 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.

Inconvenientes de la configuración estática:

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

* **Alocación Dinámica** (heap): variables dinámicas o referenciadas.

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

Ventajas de la configuración dinámica:

\*Su flexibilidad, ya que las estructuras “dinámicas” pueden crecer o decrecer durante la ejecución del programa.

**Puntero:** es un tipo de dato simple que contiene la dirección de otro dato.

* Un puntero es un tipo de variable usada para almacenar la dirección en memoria de otra variable, en lugar de un dato convencional.
* Mediante la variable de tipo puntero se accede a esa otra variable, almacenada en la dirección de memoria que señala el puntero. Es decir, el valor de la variable de tipo puntero es una dirección de memoria.
* Se dice que el puntero apunta o señala a la variable almacenada en la dirección de memoria 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.
* 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.
* Una variable de tipo puntero ocupa 4 bytes de memoria (stack) para su representación interna en Pascal
* Declaración: **TYPE** PunteroAEntero=^Integer;
* Un puntero puede apuntar a una variable de cualquier tipo, incluso tipos estructurados.
* Un dato referenciado o apuntado, como los ejemplos vistos, no tienen memoria asignada, o lo que es lo mismo no existe inicialmente espacio reservado en memoria para este dato.
* Para poder emplear variables dinámicas es necesario usar el tipo de dato PUNTERO que permite referenciar nuevas posiciones de memoria que no han sido declaradas a priori y que se van a crear y destruir en tiempo de 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.
* En Pascal la creación y destrucción de variables dinámicas se realiza mediante los siguientes procedimientos:
* New (puntero)
* Dispose (puntero)

Operaciones frecuentes:

* 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
* **Eliminación de un objeto apuntado que no se necesita:**

Liberación de memoria 🡺Dispose( p )

* **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

**Clase 10: Listas**

* 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). Homogénea, Dinámica, De acceso secuencial, Lineal.
* 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.
* 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.

**Características de las listas:**

* 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

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

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

Arreglos: son más económicos.

Listas: requieren espacio extra para los enlaces.

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

**a. Recuperar Datos**: se tienen dos estrategias básicas

* Acceso directo
* Acceso secuencial

Listas: acceso secuencial, el tiempo para acceder a un ítem es proporcional a su posición en la lista. Cada nodo contiene la dirección del próximo.

**b. Almacenar Datos**: se deben tener en cuenta tres factores

1. Determinar en qué lugar de la estructura el dato será almacenado

2. Crear el espacio para el dato en la posición apropiada

3. Copiar el dato a su ubicación de memoria

Listas 🡺 agregar adelante, agregar atrás, insertar…

**Listas con doble enlace:**

Son listas enlazadas que tienen dos enlaces en lugar de uno. Sus NODOS se encuentran enlazados a otros 2 NODOS por medio de criterios diferentes. Podemos recorrerla en 2 sentidos (órdenes) diferentes 🡺 tenemos 2 “vistas” diferentes de los mismos datos

**VENTAJA**

* Pueden accederse siguiendo cualquiera de los dos órdenes, sin utilizar espacio extra...

**INCONVENIENTE**

* Ocupan más memoria por nodo que una lista simple.

**Listas doblemente enlazadas**: Cada nodo tiene un “siguiente” y un “anterior” los cuales consideran el mismo criterio.

Por ejemplo: Una lista de números donde:

Orden 1: Orden Ascendente

Orden 2: Orden Descendente

Los enlaces permitirán recorrer la lista en dos sentidos de manera tal que un sentido da la inversa del otro.

**Listas con múltiples enlaces:** Son listas cuyos nodos están enlazados por diferentes criterios.

Podemos establecer similitudes con las listas con doble enlace.

**VENTAJA**

* Pueden recorrerse siguiendo diferentes enlaces, lo que permite vistas diferentes de la misma información.

**INCONVENIENTE**

* Ocupan más memoria por nodo …

**Listas circulares:** Característica principal: El último nodo de la lista contiene la dirección del comienzo de la lista.

Cuando se crea la lista se tendrá que tener en cuenta que el último nodo se enlace con el primero. Se pueden utilizar todas las operaciones de lista ya definidas teniendo en cuenta la característica del último nodo de la lista.

**Ventaja**:

Cada nodo de la lista es accesible desde cualquier otro nodo de ella.

**Inconvenientes**

Se pueden producir lazos infinitos.

**Merge de Listas.**

**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.

**Clase 11: Corrección**

Una vez escrita una posible solución es necesario comprobar que cumple con el objetivo propuesto.Esta tarea se conoce como: Corrección de programa

Una vez que se ha escrito un programa se debe probar que el mismo es correcto.Un programa es correcto si se realiza de acuerdo a sus especificaciones.Por esta razón es muy importante una completa especificación.

**Técnicas para verificar corrección:**

* + - * + **TEST**
        + **VERIFICACION**
        + **WALKTHROUGH**
        + **DEBUGGING**

**Testing:** El proceso de proveer evidencias convincentes respecto de si el programa hace el trabajo esperado

¿Como se proveen evidencias?

* Diseñar 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 en los casos límite
* 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. (Esto asegura que las pruebas no están pensadas a favor del que escribió el programa)
* Cuando se tiene el plan de pruebas y el programa, el plan debe aplicarse sistemáticamente.
* 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**: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 poder monitorear su comportamiento más cercana

**Los errores pueden provenir de varios caminos, por ejemplo:**

* El diseño del programa puede ser defectuoso.
* El programa puede usar un algoritmo defectuoso.
* Algunas veces el error es tan evidente que se reconoce rápidamente en que lugar del programa está la falla y se reescribe la parte que era fuente del error.
* Otras veces se puede necesitar agregar sentencias de salida adicionales que sirven como punto de control o para señalar cambios en ciertas variables claves.

**Walkthroughs**: Elproceso de recorrer el programa ante una audiencia.

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

**Verificación**: El proceso de analizar las post-condiciones en función de las precondiciones establecidas.

* 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 deben considerarse antes de que el programa pueda comenzar a ejecutarse, por ejemplo: entradas de datos disponibles. Son consideradas SIEMPRE verdaderas.
* Las **postcondiciones** describen los aspectos que deben cumplirse cuando el programa terminó.

**Clase 12: Eficiencia.**

El análisis de la eficiencia de un algoritmo estudia el **tiempo** que tarda un algoritmo en ejecutarse y la **memoria** requerida, entre otros aspectos.

Una vez que se obtiene un algoritmo y se comprueba que es correcto, es importante determinar cuántos recursos, como tiempo de ejecución o espacio en memoria, se requiere para la solución del problema.

Medición de la Memoria utilizada en un programa

Se puede calcular únicamente la cantidad de memoria estática que utiliza el programa.

Se deben analizar 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, Problemas de Ordenación de vectores

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

**Análisis Empírico:** Para realizar un análisis empírico, es necesario ejecutar el programa y medir los recursos utilizados.

**Inconveniente:** este análisis tiene varias limitaciones: puede dar una información pobre de los recursos consumidos. Por ejemplo:

* 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.

**Análisis Teórico:**Para realizar un análisis teórico, es necesario establecer una medida intrínseca de la cantidad de trabajo realizado por el algoritmo; esta medida nos permite comparar algoritmos y seleccionar la mejor implementación.

* + - Obtiene valores aproximados
    - Es aplicable en la etapa de diseño de los algoritmos, uno de los aspectos fundamentales a tener en cuenta
    - Independiente de la máquina donde se ejecute
    - Permite analizar el comportamiento
    - Se puede aplicar sin necesidad de implementar el algoritmo.
    - La predicción del costo puede evitar una implementación posiblemente compleja.
    - Se tendrá en cuenta el número de operaciones elementales que emplea el algoritmo.
    - Una operación elemental utilizará un tiempo constante para su ejecución, independientemente del tipo de dato con el que trabaje.
    - Cada operación elemental se ejecutará en una unidad de tiempo.
    - Una operación elementales: una asignación, una comparación o una operación aritmética simple.

**Reglas Generales para el cálculo del tiempo de ejecución**

* **Regla 1**: For
* **Regla 2**: For anidados
* **Regla 3**: sentencias consecutivas
* **Regla 4**: If / else
* Los comentarios y declaraciones no se consideran para el cálculo

**Cálculo del tiempo de ejecución en una solución modularizada**

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

Desde el punto de vista del tiempo, una solución no recursiva puede ser más eficiente que otra recursiva, esto se debe a:

* El excesivo uso de recursos (overhead) asociado con la llamada a subprogramas
* La ineficiencia inherente de algunos algoritmos recursivos

Algunas condiciones en las cuales el tiempo de ejecución de un programa se puede ignorar en favor de otros factores:

* Si un programa se va a utilizar algunas veces, el costo de su escritura y depuración es el dominante.
* Si un programa se va a ejecutar sólo con entradas “pequeñas”, la velocidad de crecimiento puede ser menos importante.
* Un algoritmo eficiente pero complicado puede no ser apropiado para el mantenimiento por parte de un tercero.

**Conclusiones**

* El término *eficiencia* no debería relacionarse únicamente con el tiempo de ejecución de un proceso, sino con la buena administración de TODOS los recursos disponibles, entre los que se halla el tiempo de la CPU y la memoria.
* La recursividad es una técnica de programación muy útil para resolver problemas que no tienen una solución iterativa sencilla. No siempre resultan ser soluciones eficientes.

**Clase 13: Arboles**

Un **árbol** es una estructura de datos que satisface tres propiedades:

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

Arbol:

1. Múltiples sucesores

2. Un solo predecesor

3. Relación uno a muchos 🡪 organización jerárquica de la información

* Homogénea: todos los elementos son del mismo tipo.
* Dinámica: puede aumentar o disminuir su tamaño durante la ejecución del programa
* No lineal: cada elemento puede tener 0, 1, o más sucesores
* Acceso Secuencial

Árbol: Estructura Enlazada donde cada nodo tiene múltiples links a otros nodos

**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 mas largo de la raíz a una hoja

Un **árbol** es una estructura donde cada nodo puede tener múltiples links a otros nodos:

-Si cada nodo tiene como máximo dos links entonces se denomina árbol binario.

-Si cada nodo tiene como máximo tres links entonces se denomina árbol ternario.

Cada nodo del árbol es un registro con un campo de datos y campos de enlace a otros nodos del mismo tipo.

**Árboles Binarios de Búsqueda**: 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.

1. Cada nodo tiene un valor que

(a) es más grande que el valor de todos los nodos del subárbol izquierdo

(b) es menor que el valor de todos los nodos del subárbol derecho

2. **Utilidad más importante**🡺**Búsquedas**

(el tiempo medio es O(log n))

* Recorrido de un árbol: permite desplazarse a través de un árbol en forma tal que, cada nodo sea visitado una y solo una vez.

Existen varios métodos

* **Recorrido En – Orden**
* **Recorrido Pre – Orden**
* **Recorrido Post – Orden**
* Operación Buscar: retorna un valor booleano que indica si el dato está o no en el árbol
* Borrado de un nodo del árbol

**Se deben considerar diferentes situaciones:**

1. Si el nodo **es una hoja**
2. Si el nodo tiene un hijo, el nodo puede ser borrado después que su padre actualice el puntero al hijo del nodo que se quiere borrar.
3. Si el nodo tiene dos hijos

**Clase 14: TAD. Tipo Abstracto de Dato.**

* Los ***tiposde datos*** son necesarios para identificar valores y operaciones posibles para variables y expresiones.
* Las nociones de ***estructuras de datos****,* ***variables*** *y* ***constantes*** que se han estudiado son abstracciones para tratar de acercar al mundo real la especificación de los datos de problemas.
* Los lenguajes de programación y los sistemas operativos se encargan de manejar las conversiones entre las definiciones abstractas y la representación interna en memoria.
* Hasta ahora se han considerado tipos de datos simples, estructurados y definidos por el usuario.
* La mayoría de los lenguajes de programación modernos *impone la obligación de declarar* todos los nombres de variables, *asociándoles un tipo.*
* Los lenguajes también verifican que las operaciones se hagan entre datos *del mismo tipo.* La mezcla de tipos en general es detectada y considerada un error por los lenguajes de programación.

También hemos visto la ventaja de descomponer (modularizar) un problema (sistema de software) en procedimientos y funciones.

De este modo se abstraen las operaciones del sistema, de modo de descomponerlo en funcionalidades (procedimientos/ funciones) relativamente independientes y más simples que el problema global.

El proceso de abstracción se entiende como la identificación de los conceptos esenciales, ignorando los detalles.

Hasta ahora se ha puesto el foco en:

* + Abstracción de procedimientos 🡪**módulos**

Ahora vamos a trabajar sobre los datos:

* + Abstracción de datos 🡪**TAD**

Caracterizar los problemas del mundo real significa *reconocer* los objetos del mundo real y abstraer sus aspectos fundamentales (especificación) y su comportamiento (operaciones), de modo de representarlos sobre una computadora.

La *utilidad* fundamental de *abstraer y modelizar* objetos del mundo real es **re-usar** soluciones en diferentes problemas.

En el desarrollo de sistemas de software *es* importante trabajar con especificaciones abstractas, de modo de mejorar la calidad de los programas.

Los tipos abstractos de datos permiten realizar encapsulamiento o empaquetamiento de los datos ya que se define un nuevo tipo y las operaciones que se pueden hacer con él.

El concepto de***tipo abstracto de dato*** es un tipo de dato definido por el programador que incluye:

* 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.

**Cuáles son los objetivos que se persiguen?**

* Ocultamiento de datos (data hiding): para que el usuario solo conozca el nombre y las operaciones del dato sin permitirle el acceso a la representación del dato y la implementación de sus operaciones
* Independencia de la representación: cualquier cambio en la especificación o implementación de las operaciones no afecta lo que el usuario conoce del dato.

En un TAD existen dos partes diferenciadas:

* La Interface de utilización : **Interface** (visible)

- Definición

- Operaciones

* La Implementación: **Implementation**(oculto)

-Representación tipo

-Implementación de las operaciones

A la hora de utilizar el TAD, la representación del tipo y la implementación de las operaciones debe permanecer oculta. Solo podremos utilizar la definición y operaciones permitidas en la interface.

**Características del TAD:**

* + Es un tipo de datos para representar objetos del mundo real que no existen en el lenguaje.
  + Presentan una parte pública (INTERFACE) donde se declara el nombre del tipo y las operaciones permitidas; y una parte privada (IMPLEMENTACION) donde se representa el tipo declarado y todas las operaciones descriptas en la interface.
  + El usuario sólo conoce la parte pública, es decir, no conoce cual es la representación del tipo utilizada ni cual es la implementación de las operaciones.

En el TAD hay dos 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.

**Ventajas del TAD:**

* 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.

Consideraciones:

* Un TAD puede tener un único tipo exportado.
* Los parámetros de los procedimientos y funciones de un TAD pueden ser del tipo integer, boolean, char, string, el propio tipo exportado o bien el tipo exportado de otro TAD.
* Uno de los parámetros necesariamente debe ser el del tipo exportado
* En la implementación de las operaciones del TAD no se deberían utilizar las operaciones de lectura y escritura
* El programa que utiliza el TAD sólo puede aplicar a las variables del tipo exportado del TAD las operaciones definidas en la interface, esto debe incluir a las operaciones de asignación y comparación en caso de ser necesarias.

**Tipos Abstractos de Datos: PILA**

El tipo de Dato PILA (stack) 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:

* **Homogénea**: ya que almacena elementos del mismo tipo.
* **Dinámica**: ya que permite agregar y sacar elementos durante la ejecución del programa.
* **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).
* **Lineal**: cada elemento de la estructura posee un elemento que le precede y uno que le sigue, excepto el primero y el último.

Son muy pocos los lenguajes de programación que cuentan con una estructura de este tipo como tipo estándar. En particular, Pascal no dispone del tipo de datos pila.

**OPERACIONES:**

* Crear una pila: st\_create
* Agregar un elemento: st\_push
* Sacar un elemento: st\_pop
* Conocer cual 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

**Tipos Abstractos de Datos: COLA**

El tipo de Dato COLA (queue) 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:

* **Homogénea**: ya que almacena elementos del mismo tipo.
* **Dinámica**: ya que permite agregar y sacar elementos durante la ejecución del programa.
* **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).
* **Lineal**: cada elemento de la estructura posee un elemento que le precede y uno que le sigue, excepto el primero y el último.

Son muy pocos los lenguajes de programación que cuentan con una estructura de este tipo como tipo estándar. En particular, Pascal no dispone del tipo de datos cola.

**OPERACIONES:**

* Crear una cola: q\_create
* Agregar un elemento a la estructura: q\_push
* Sacar un elemento de la estructura: q\_pop
* Conocer cual es el elemento que está en el frente de la cola: q\_top
* Conocer cual 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

Function potenciaIterativa (x,n:integer) : integer;

Var

tot:integer;

Begin

Tot:=1;

While (n>=0) do begin

tot:=tot\*x;

n:=n-1;

end;

potenciaiterativa:=tot;

END;

//agrega el elemento elem a la cola

Procedure q\_push (var c:colaAlumnos; elem:alumno);

var

aux:cola;

Begin

new (aux);

asignar (aux^.elemento, elem);

aux^.sig:= nil;

if (c.pri= nil) then c.pri:= aux

else c.ult^.sig:= aux;

c.ult:= aux;

c.cant:= c.cant+1;