* Analizar problemas, poniendo énfasis en la **modelización**, **abstracción** y en la **modularización** de los mismos.
* Obtener una expresión sintética, precisa y **documentada** de los problemas y su solución.

(conseguir una descripción breve, precisa y respaldada por evidencias de los problemas existentes)

* Analizar y expresar correctamente algoritmos, orientando los mismos a la **resolución de las partes** (módulos) en que se descomponen los problemas.
* Introducir las nociones de **estructuras de datos**, **tipos de datos** y **abstracción de datos**.

Informática: Es la **ciencia** que estudia el análisis y **resolución de problemas** utilizando **computadoras**.

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

Resolución: Se puede utilizar las herramientas informáticas en aplicaciones de áreas muy diferentes.

Modelizar: **El modelo define los mecanismos de interacción y sus condiciones.**

permite simular diferentes escenarios y analizar los resultados para determinar la mejor solución

**Pensar que acciones se van a permitir y que implica cada acción permitida**

Modularizar: **A partir del modelo es necesario encontrar la forma de descomponer en partes (módulos) para obtener una solución.**

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 reutilizar los módulos.

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

Programa = Algoritmo + Datos

**Algoritmo: Secuencia de pasos (instrucciones) a realizar sobre un autómata para alcanzar un resultado deseado en un tiempo finito.**

**Dato: Es una representación de un objeto del mundo real mediante la cual podemos modelizar aspectos del problema que se quiere resolver. Puede ser constante o variable**.

**Es una clase de objetos de datos ligados a un conjunto de operaciones para crearlos y manipularlos.**

**SIMPLE**: aquellos que toman un único valor, en un momento determinado, de todos los permitidos para ese tipo.

**COMPUESTO**: pueden tomar varios valores a la vez que guardan alguna relación lógica entre ellos, bajo un único nombre.

DEFINIDO POR EL LENGUAJE: Son provistos por el lenguaje y tanto la representación como sus operaciones y valores son reservadas al mismo.

DEFINIDO POR EL PROGRAMADOR: Permiten definir nuevos tipos de datos a partir de los tipos simples.

**DATO NUMERICO**

**Representa el conjunto de números que se pueden necesitar. Estos números pueden ser enteros o reales.**

Es un tipo de dato simple, ordinal

Operadores Enteros: Mod/Div

**DATO REAL:**

**Representa el conjunto de números que se pueden necesitar. Estos números pueden ser enteros o reales.**

Es un tipo de dato simple, permiten representar números con decimales

**DATO LÓGICO:**

**Permite representar datos que pueden tomar dos valores verdadero o falso.**

Es un tipo de dato simple, ordinal

**DATO CARÁCTER:**

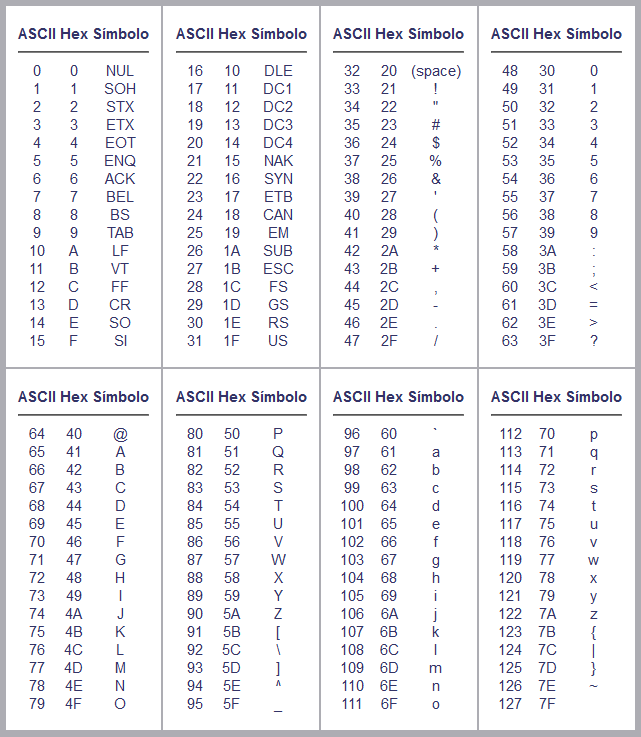
**Representa un conjunto finito y ordenado de caracteres que la computadora reconoce. Un dato de tipo carácter contiene solo un carácter.**

Es un tipo de dato simple, ordinal

Operadores Lógicos: <, <=, >, =>, =, <>

Precondición: **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.**



**READ:**

**Se usa para tomar datos desde un dispositivo de entrada (por defecto desde teclado) y asignarlos a las variables correspondientes.**

**WRITE:**

**Se usa para mostrar el contenido de una variable, por defecto en pantalla.**

**Secuencia:**

**Está representada por una sucesión de operaciones (por ej. asignaciones), en la que el orden de ejecución coincide con el orden físico de aparición de las instrucciones.**

**Decisión:**

**En un algoritmo representativo de un problema real 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:**

**ITERACION – PRECONDICIONAL (WHILE):**

**Evalúan la condición y si es verdadera se ejecuta el bloque de acciones. Dicho bloque se pueda ejecutar 0, 1 o más veces.**

**Importante: el valor inicial de la condición debe ser conocido o evaluable antes de la evaluación de la condición.**

**ITERACION – POSTCONDICIONAL (REPEAT-UNTIL):**

**Ejecutan las acciones luego evalúan la condición y ejecutan las acciones mientras la condición es falsa. Dicho bloque se pueda ejecutar 1 o más veces.**

**REPETICION (FOR):**

**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**

**SELECCIÓN:**

**Permite realizar distintas acciones dependiendo del valor de una variable de tipo ordinal.**

case (variable) of

condición (variable = 3): accion1;

condición: begin accion1; accion2; end;

else accion1;

end;

**Tipos de Datos definidos por el programador:**

**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.**

Ventajas:

**Flexibilidad: en el caso de ser necesario modificar la forma en que se representa el dato, sólo se debe modificar una declaración en lugar de un conjunto de declaraciones de variables.**

**Documentación: se pueden usar como identificador de los tipos, nombres auto explicativos, facilitando de esta manera el entendimiento y lectura del programa.**

**Seguridad: se reducen los errores por uso de operaciones inadecuadas del dato a manejar, y se pueden obtener programas más confiables.**

**SUBRANGO:**

**Es un tipo ordinal que consiste de una sucesión de valores de un tipo ordinal (predefinido o definido por el usuario) tomado como base.**

Es simple y ordinal.

**MODULARIZAR:**

**Significa dividir un problema en partes funcionalmente independientes, que encapsulen operaciones y datos.**

**MODULO:**

**Tarea específica bien definida se comunican entre sí adecuadamente y cooperan para conseguir un objetivo común.**

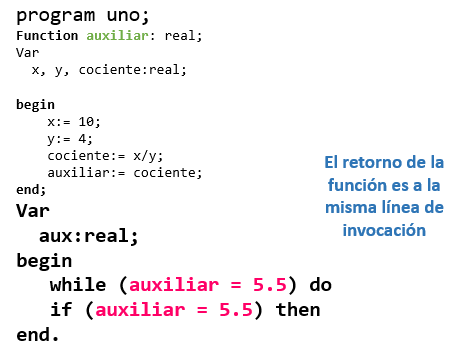
**Encapsula acciones tareas o funciones.**

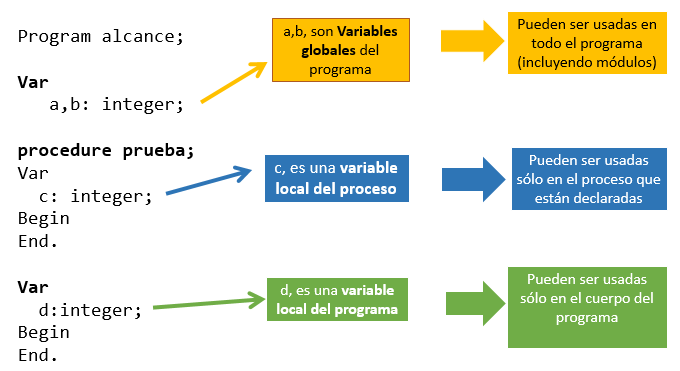
**En ellos se pueden representar los objetivos relevantes del problema a resolver.**

**PROCEDIMIENTO:**

**Conjunto de instrucciones que realizan una tarea específica y retorna 0, 1 o más valores.**

**FUNCION:**

**Conjunto de instrucciones que realizan una tarea específica y retorna un único valor de tipo simple.**



Los módulos se comunican entre ellos por medio de parámetros y variables globales (esto no es aconsejable, ya que no se especifica la comunicación entre módulos, por lo que cada módulo no sabe si una variable global fue modifica por otro módulo o no).

**ESTRUCTURA DE DATOS:**

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

Elementos: Pueden ser **homogéneos** (mismo tipo) o **heterogéneos** (diferente tipo).

Tamaño: Si la estructura puede variar su tamaño durante la ejecución del programa. **Estático** o **dinámico**.

Acceso: Como se accede a los elementos que la componen.

-**Secuencial**: Para acceder a un elemento particular se debe respetar un orden predeterminado.

-**Directo**: Se puede acceder a un elemento particular, directamente

Linealidad: Como se encuentran almacenados los elementos que la componen.

-**Lineal**: Está formada por ninguno, uno o varios elementos que guardan una relación de adyacencia ordenada donde a cada elemento le sigue uno y le precede uno, solamente.

-**No lineal**: Para un elemento dado pueden existir 0, 1 o más elementos que le suceden y 0, 1 o más elementos que le preceden.

**REGISTRO:**

**Es uno de los tipos de datos estructurados, que permiten agrupar diferentes clases de datos en una estructura única bajo un sólo nombre**

**Heterogéneo, estático y su acceso es por campos (representan cada uno de los datos que forman el registro)**

**ARREGLO:**

**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.**

Vector: Es una colección de elementos que se guardan consecutivamente en la memoria y se pueden referenciar a través de un índice.

Es homogénea (mismo tipo), estática (no cambia el tamaño durante la ejecución del programa) e indexada (para acceder a cada elemento de la estructura se debe utilizar una variable índice de tipo ordinal (integer, real, char, boolean)).

Por ser **estática**, permite el acceso rápido a sus componentes a través de la variable **índice.**

Vector = array [rango] of tipo;

El rango debe ser de tipo ordinal (integer, char, boolean, subrango);

El tipo debe ser estático (Integer, char, real, boolean, subrango, registro, vector);

Type

numeros = array [1..10] of real;

frecuen = array [char] of real;

otros = array [‘h’..’m’] of integer;

var num:numeros; **num reserva memoria para 10 números reales.**

Nuevo:frecuen; **nuevo reserva memoria para 256 números reales**

Otro:otros; **otro reserva memoria para 6 números enteros**

El vector se puede recorrer de forma total (analiza todos los elementos de la estructura) o parcial (analiza los elementos hasta encontrar aquel que cumpla con lo pedido).

Dimensión física: **Se especifica en el momento de la declaración y determina su ocupación máxima de memoria.**

**La cantidad de memoria total reservada no variará durante la ejecución del programa.**

Dimensión lógica: **Se determina cuando se cargan contenidos a los elementos del arreglo.**

**Indica la cantidad de posiciones de memoria ocupadas con contenido real. Nunca puede superar la dimensión física.**

Agregar: **Significa poner al final de los elementos que tiene el vector un nuevo elemento. Puede pasar que esta operación no se pueda realizar si el vector está lleno.**

Insertar: **Significa agregar en el vector un elemento en una posición determinada. Puede pasar que esta operación no se pueda realizar si el vector está lleno o si la posición no es válida**

Eliminar: **Significa borrar (lógicamente) en el vector un elemento en una posición determinada, o un valor determinado. Puede pasar que esta operación no se pueda realizar si la posición no es válida, o en el caso de eliminar un elemento si el mismo no está**

Buscar: Recorrer el vector buscando un valor que puede o no estar en el vector.

Si el vector está desordenado se debe recorrer todo el vector (en el peor de los casos), y detener la búsqueda cuando se encuentra el dato o se terminó de recorrer.

Si está ordenado se debe aprovechar el orden, existe dos formas búsqueda mejorada y búsqueda dicotómica.

**MEMORIA DINÁMICA Y ESTÁTICA:**

Variables:

Char, (1 byte)

Boolean, (1 byte)

Integer, (4 bytes)

Real, (8 bytes)

String, (tamaño + 1 byte)

subrango, (depende el tipo)

registro, (suma de sus campos)

arreglos (dimFísica\* tipo elemento)

puntero (4 bytes)

VARIABLES ESTATICAS: Las variables y tipos reservan memoria en su declaración y se mantienen durante todo el programa.

**PUNTERO:**

**Es un tipo de variable usada para almacenar una dirección en memoria. Se almacena cualquier tipo de dato.**

**Es un tipo simple.**

Pueden apuntar solamente a direcciones almacenadas en memoria dinámica.

Cada variable de tipo puntero puede apuntar a un único tipo de dato.

Una variable de tipo puntero puede reservar y liberar memoria durante la ejecución de un programa.

**En su creación, reserva una dirección memoria dinámica libre para poder asignarle contenidos a la dirección que contiene la variable de tipo puntero.**

**If(p = nil) compara si el puntero p no tiene dirección asignada.**

if **(p = q)** then, compara si los punteros p y q apuntan a la misma dirección de memoria.

if **(p^ = q^)** then, compara si los punteros p y q tienen el mismo contenido.

**No** se puede comparar por mayor o menor direcciones de punteros **(p>q)**;

**LISTA:**

**Colección de nodos, donde cada nodo contiene un elemento y en qué dirección de memoria se encuentra el siguiente nodo.**

**Cada nodo de la lista se representa con un puntero, que apunta a un dato (elemento de la lista) y a una dirección (donde se ubica el siguiente elemento de la lista).**

**Toda lista tiene un nodo inicial.**

Los **nodos** que la componen pueden no ocupar posiciones contiguas de memoria. Es decir, pueden aparecer dispersos en la memoria, pero mantienen un orden lógico interno.

Es una estructura de datos compuesta que es **homogénea**, **dinámica** (la cantidad de nodos puede variar durante la ejecución), **lineal** (cada nodo tiene un único antecesor y sucesor), y el acceso a cada uno de sus elementos es **secuencial**.

|  |  |  |  |
| --- | --- | --- | --- |
| **TIPOS DE DATOS** | | | |
| **SIMPLE** | | **COMPUESTO** | |
| **LENGUAJE** | **PROGRAMADOR** | **LENGUAJE** | **PROGRAMADOR** |
| **CHAR** | **SUBRANGO** | **STRING** | **REGISTROS** |
| **BOOLEAN** |  |  | **ARREGLOS** |
| **INTEGER** |  |  | **LISTA** |
| **PUNTERO** |  |  |  |
| **REAL** |  |  |  |

**TESTING**

El propósito del Testing es proveer evidencias convincentes que el programa hace el trabajo esperado.

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. Lo mejor es hacerlo antes de escribir el programa.

una vez que el programa ha sido implementado y se tiene el plan de pruebas

Se analizan los casos de prueba.

Si hay errores se corrigen.

Estos dos pasos se repiten hasta que no haya errores.

**DEBUGGING**

Es el proceso de descubrir y reparar la causa del error.

El diseño y aplicación de pruebas adicionales para ubicar y conocer la naturaleza del error.

Es agregar sentencias adicionales en el programa para poder monitorear su comportamiento más cercanamente.

Los errores pueden provenir de dos fuentes:

**El diseño del programa no es el adecuado.**

**El programa no está escrito correctamente.**

**WALKTHROUGH**

Es recorrer un programa frente a 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.

**VERIFICACION**

Verificar un programa significa controlar que se cumplan las pre y post condiciones del mismo.

**EFICIENCIA**

El análisis de la eficiencia de un algoritmo estudia el **tiempo** que tarda un algoritmo en ejecutarse y la **memoria** que requiere.

|  |
| --- |
| // C = evaluar condicion/es en UTs |
|  |

|  |
| --- |
| // N = veces que se ejecuta |
|  |

|  |
| --- |
| // Cuerpo = ejecutar cuerpo en UTs |
|  |

|  |
| --- |
| // |
|  |

|  |
| --- |
| // IF = C + Cuerpo |
|  |

|  |
| --- |
| // IF-ELSE = C + MAX(Cuerpo1, Cuerpo2) |
|  |

|  |
| --- |
| // FOR = 3N + 2 + N(Cuerpo) |
|  |

// WHILE = C(N + 1) + N(Cuerpo)

Vector:

**Remover**

if((indice >= 1) and (indice <= dimL)) then

begin

dimL := dimL - 1;

for i:=indice to dimL do

v[i] := v[i+1];

end;

**Existe valor ordenado**

while(pos <= dimL) and (v[pos] < num)do

pos := pos + 1;

existeValorOrdenado := v[pos] = num;

**Insertar**

if((dimL < N) and (indice >= 1) and (indice <= dimL)) then

begin

dimL := dimL + 1;

for i:=dimL downto indice+1 do

v[i] := v[i-1];

v[indice] := valor;

end;

**Agregar**

if(dimL < N) then

begin

dimL := dimL + 1;

v[dimL] := valor;

end;

Listas

**Agregar adelante**

new(nuevo);

nuevo^.valor := valor;

nuevo^.sig := L;

L := nuevo;

**Agregar atrás**

new(nuevo);

nuevo^.valor := valor;

nuevo^.sig := nil;

if(L = nil) then

L := nuevo

else

ultimo^.sig := nuevo;

ultimo := nuevo;

**Busqueda ordenada**

while((L <> nil) and (X < L^.valor)) do

L := L^.sig;

if((L = nil) or (L^.valor <> X)) then

busquedaOrdenada := false

else

busquedaOrdenada := true;

**Insertar ordenado**

new(nuevo);

nuevo^.valor := X;

actual := L;

anterior := L;

// Recorro mientras no se termine la lista y no encuentro la posición correcta

while((actual <> nil) and (actual^.valor < X)) do // De menor a mayor

begin

anterior := actual;

actual := actual^.sig;

end;

if (anterior = actual) then

L := nuevo // el dato va al principio

else

anterior^.sig := nuevo; // va entre otros dos o al final

nuevo^.sig := actual;

**Eliminar ordenado**

actual:= L;

anterior := L;

while(actual <> nil) and (actual.elem <> value) do

begin

anterior:= actual;

actual:= actual ^.sig;

end;

if(actual <> nil) then

begin

if(actual = L) then

L := L^.sig;

else

anterior ^.sig := actual ^.sig;

dispose(actual);

end;

Testing: proveer evidencias convincentes de que el programa hace el trabajo esperado.

Para eso se diseña un plan de pruebas.

En este, se deciden qué aspectos del programa van a ser testeados, se encuentran datos de prueba para cada aspecto.

Se determina el resultado esperado que producirá el programa para cada caso.

Se pone atención a los casos límite.

Se diseñan casos de prueba en base a lo que se sabe del programa y no sobre lo que se escribió. Esto se recomienda hacer antes de escribir el programa.

Luego, con el programa implementado y el plan de pruebas diseñado, se analizan los casos de prueba, se buscan errores y se los corrige, esto se repite hasta que no haya más errores.

Debugging: Es buscar y reparar las causas del error.

Se diseña y aplican casos adicionales para saber y encontrar la naturaleza del error.

Se agregan sentencias adicionales al programa para poder monitorearlo mas cerca.

Testing: El propósito del testing es proveer evidencias convincentes de que el programa realiza el trabajo esperado. Esto se hace mediante el diseño de un plan de pruebas.

Se seleccionan que aspectos del programa van a ser testeados, y se encuentran los datos de pruebas para cada uno de los aspectos.

Se determina el resultado esperado al finalizar el caso por el programa.

Se pone atención a los casos límite.

Se diseñan casos de prueba sobre lo que hace el programa y no lo que está escrito. Se recomienda hacer esto antes de escribir el programa.

Una vez implementado el programa y diseñado el plan de pruebas, se analizan los casos de prueba, se buscan errores y se arreglan, esto se repite hasta que no haya más errores.

Debugging: Es el proceso de descubrir y arreglar la causa del error.

El diseño y aplicación de pruebas adicionales para ubicar y conocer la naturaleza del error.

Se agregan sentencias adicionales al programa para monitorear el comportamiento más de cerca.

Las causas del error pueden ser porque el diseño del programa no es el adecuado o el programa no está escrito correctamente.

Debugging: Consiste en descubrir y arreglar la causa del error.

El diseño y aplicación de pruebas adicionales para ubicar y entender la naturaleza del error.

Agregar sentencias adicionales para monitorear el comportamiento del error más de cerca.

Los errores pueden venir de dos fuentes: El diseño del programa no es el adecuado y El programa no está escrito correctamente.

Verificación: Consiste en controlar que se cumplan las pre y post condiciones del programa.

Walkthrought: Consiste en leer y explicar el problema a una audiencia u otra persona, esta al no saber puede aportar una visión diferente o tú mismo al leer el problema puedes descubrir la forma de resolverlo o nuevos problemas.

Testing: El testing consiste en proveer evidencia convincente de que el programa hace lo esperado. Esto se realiza con un plan de pruebas.

Decidir qué aspectos del programa van a ser testeados, encontrar los datos de prueba para cada uno de los aspectos.

Determinar los resultados que se espera que produzca el programa en cada uno de los aspectos.

Poner atención en los casos límite.

Diseñar casos de prueba en base a lo que hace el programa y no sobre lo que está escrito.

Esto se recomienda hacer antes de escribir el programa.

Una vez implementado el programa y diseñado el plan de pruebas.

Se analizan los casos.

Si hay errores, se arreglan.

Esto se repite hasta que no haya más errores.

Debugging: Consiste en descubrir y arreglar la causa del error.

El diseño y aplicación de pruebas adicionales para buscar y entender la naturaleza del error.

Agregar sentencias adicionales al programa para monitorear más de cerca el comportamiento.

Los errores pueden venir de dos fuentes:

El diseño del programa no es el adecuado.

El programa no está escrito correctamente.

Verificación:

Consiste en controlar que se cumplan las pre y post condiciones del programa.

Vector: Es una estructura de datos compuesta en la que sus elementos se guardan en memoria de forma secuencial, lo que permite acceder a ellos mediante una variable ‘índice’ (El acceso es indexado). El tamaño del arreglo no varía por lo que es estático. Todos sus elementos son del mismo tipo, por lo que es un homogénea.

Al ser una colección de elementos, en la que todos los elementos son del mismo tipo (homogénea) y el acceso a cada uno es por medio de una variable índice (que tiene que ser de tipo ordinal), ya que al ser estática el acceso