***Resumen de CADP***

**Programa: algoritmo + datos**

**Algoritmo:** Es un conjunto de instrucciones que representan las operaciones que se ejecutaran.

**Dato**: Los datos son los valores de información de los que se necesita disponer y en ocasiones transformar para ejecutar la función del programa.

**Tipos de Datos**

Un tipo de dato es una clase de objetos ligados a un conjunto de operaciones para crearlos y manipularlos. Los tipos de datos se caracterizan por: un rango de valores posibles, un conjunto de operaciones realizables sobre ese tipo, su representación interno.

Existen 2 tipos de datos:

* **Simple**: toman un único valor en un momento determinado.
  + Definidos por el lenguaje(Provistos por el lenguaje y la representación

como sus operaciones y valores las pone el lenguaje)

* + - Numerico: números que pueden ser Enteros(Ordinal) o Reales, al tener una representación interna tienen un número mínimo y máximo.
    - Carácter: representa un conjunto finito y ordenado de caracteres que la computadora reconoce
    - Logico/Booleano: puede representar dos valores - verdadero o falso
  + Definidos por el programador: es aquel dato que no está definido por el lenguaje y el programador es el encargado de su especificación.
    - Subrangos: sucesión de valores de tipo ordinal, predefinidos o definido por el usuario.
* **Compuestos**: Pueden tomar varios valores a la vez y guardan una relación lógica entre ellos.
  + String: sucesión de caracteres de un largo determinado.
  + Definidos por el programador:
    - **Registros:** permiten agrupar diferentes tipos de datos en una estructura única. Es heterogénea porque los datos pueden ser de distintos tipos, es estática porque al declararla se sabe cuánto ocupa, y de acceso directo.
    - **Listas:** Es una colección de elementos homogéneos (todos del mismo tipo) con una relación lineal que los vincula, es decir, que cada elemento tiene un único predecesor (excepto el primer elemento), y un único sucesor (excepto el ultimo). Es dinámica ya que su tamaño se modifica durante la ejecución. Los elementos no tienen un orden secuencial en la memoria, pero tienen un orden interno lógico.

Compuesta por Nodos, se conectan por medio de enlaces o punteros. Cada nodo tiene un dato (cualquier dato) y siguiente (tipo puntero)

* + - **Arreglos:** Es una colección de elementos que se guardan consecutivamente en la memoria y se pueden referenciar a través de una variable “índice”. Es homogénea porque los elementos son todos del mismo tipo, es estática porque el tamaño no cambia durante la ejecución e indexada porque para ingresar a una posición se debe llevar una variable “índice”. Se utilizan dos valores para la dimensión de un arreglo.   
      Dimensión Física: se especifica en el momento de la declaración y determina la ocupación máxima de memoria.

Dimensión Lógica: Se determina cuando se cargan contenidos a los elementos del arreglo. Indica la cantidad de posiciones ocupadas con contenido.

**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.

**Clasificación:**

-**Elementos:**

Homogénea: Los elementos son del mismo tipo

Heterogénea: Los elementos son de distintos tipos

-**Tamaño:**

Estática: El tamaño de la estructura no varia durante la ejecución

Dinámica: El tamaño de la estructura varia durante la ejecución.

-**Acceso**

Secuencial: Para acceder a un elemento se debe respetar un orden, pasando por

todos los elementos que le preceden.

Directo: se puede acceder a un elemento particular, directamente, sin necesidad de

pasar por los anteriores a él, por ejemplo, referenciando una posición.

-**Linealidad**

Lineal: ninguno, uno o varios elementos que tienen una relación de adyacencia

ordenada donde a cada elemento le sigue uno y le precede uno.

No lineal: Para un elemento dado puede existir 0 ,1 o más elementos que le

suceden y 0,1 o más elementos que le precedes (arboles)

**Estructuras de control**

Todos los lenguajes de programación tienen un conjunto minimo de instrucciones que permiten especificar el control del algoritmo que se quiere implementar. Como minimo debe contener:

**- Secuencia**: estructura de control mas simple esta representada por una sucesión de operación (por ejemplo asignación), 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 prácticamente imposible que las instrucciones sean secuenciales puras. Es necesario tomar decisiones en función de los datos del problema. Por ejemplo: If.

**- Iteracion**: puede ocurrir que se desee ejecutar un bloque de instrucciones desconociendo el numero exacto de veces que se ejecutan. Para estos casos existen en la mayoría de los lenguajes de programación estructurada

Existen 2 tipos:

* **Precondicionales**: evalúan la condición y si es verdadera se ejecuta el bloque de acciones. Se puede ejecutar varias veces. El valor de la condición debe ser evaluable antes de realizar la evaluación de la condición.

Por ejemplo: While.

* **Postcondicionales**: ejecutan las acciones, luego evalúan la condición y ejecutan las acciones mientras la condición sea falsa.

Por ejemplo: Repeat Until

**-Repeticion**: consiste en repetir N (deben conocerse de antemano) veces un bloque de acciones. Por ejemplo: for.

**-Selección**: permiten realizar distintas acciones dependiendo del valor de una variable de tipo ordinal. Por ejemplo: case.

**Definiciones Generales**

**-Variable:** Es una zona de memoria cuyo contenido va a ser alguno de los tipos mencionados anteriormente. La dirección inicial de esta zona se asocia con el nombre de la variable. Puede cambiar su valor durante el programa.

**-Constantes**: Es una variable cuyo contenido no puede cambiar durante el programa.

**-Variable Global:** Son variables declaradas al comienzo del programa y que pueden ser modificadas o utilizadas por todos los módulos. **NO ES UTIL UTILIZARLAS**

Ventaja: amplia utilización, debido a lo anterior mencionado.

**Desventajas**: posibilidad de modificar involuntariamente en un módulo a la misma, cuando debe ser utilizada también en otro modulo, y no se especifica la comunicación entre los módulos.

Alcance de una variable:

**Clasificación de los lenguajes de acuerdo a la declaración de sus variables**

Algunos lenguajes 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. Esta clase de lenguajes se denomina fuertemente tipados (strongly typed).

Otra clase de lenguajes, que verifica el tipo de las variables según su nombre, se denomina auto tipados (self typed).

Existe una tercera clase de lenguajes que permiten que una variable tome valores de distinto tipo durante la ejecución de un programa. Esta se denomina dinámicamente tipados (dinamically typed).

**-DIV-MOD**: Se usan solo para números enteros. Div da el numero del cociente y Mod da el resto. Es lo ultimo que se hace cuando se hace una operación matemática en un programa, Ejemplo:  
b = 24 DIV 2 + 5 → 24 DIV 7 → 3

**Tipos de datos definidos por el programador-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 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.

**Modularización:**Modular es dividir el programa en partes funcionalmente independientes, que encapsulan operaciones y datos. Las ventajas de hacer esto son:

* Mayor productividad: Al dividir un sistema de software en módulos funcionalmente independientes, un equipo de desarrollo puede trabajar simultáneamente en varios módulos, incrementando la productividad (es decir reduciendo el tiempo de desarrollo global del sistema).
* 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.
* Facilidad de mantenimiento: 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 disminuye los costos de mantenimiento de los sistemas.
* 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.

**Parámetros**

Los parámetros son el mecanismo de comunicación entre el modulo y el resto del sistema.

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.

El alcance de los parámetros son: las variables globales (pueden ser usadas en todo el programa (incluyendo módulos)), las variables locales al programa (Pueden ser usadas sólo en el cuerpo del programa), y las variables locales del proceso (Pueden ser usadas sólo en el proceso que están declaradas).

* Referencia: Se copia la dirección de la variable original y con el dato se puede operar pero las modificaciones se verán afectadas en dónde provino la variable original.
* Valor: Recibe un valor proveniente de otro modulo o programa principal y que se le puede realizar operaciones pero no producirá un cambio ni incidencia fuera del módulo. Se le hace una copia de la variable/dato la cual si se modifica no modificara la variable o dato de dónde provino.

**Función**: Conjunto de instrucciones que realizan una tarea específica y retorna un único valor de tipo simple. Se invocan una variable o en write o en condiciones de IF o While.

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

**Diferencia entre función y procedure**:

-*Distintas formas de llamado*: La función se invoca en una variable o en un Write o en una condición de un IF o While. El procedure se lo llama por separado a través del nombre y sus respectivos parámetros.

-*Distinto lugar de retorno*: La función vuelve a la misma línea, el procedure vuelve a la línea siguiente

-*Distinta forma de retorno de datos*: Función devuelve 1 solo dato de tipo simple, y procedure a través de los parámetros de referencia puede devolver 0,1 o más datos de cualquier tipo.

**Diferencia entre Arreglos y Listas**

|  |  |  |
| --- | --- | --- |
| -------------------------- | ARREGLOS | LISTAS |
| Características  Almacenamiento  Agregar  Insertar  Acceso  Espacio  Pasados por parametros | Estática  Homogénea  Indexada  Los elementos se encuentran alojado en posiciones de memoria consecutiva.  El arreglo mantiene la dimensión logica, para agregar se utiliza ese valor (acceso directo) que luego debe incrementarse.  Puede no poder hacerse la operación.  Debe buscarse el elemento,  una vez encontrado debe hacerse el corrimiento para hacer lugar.  Se carga el elemento, se incrementa la dimensión lógica.  Puede no hacerse  Para acceder a la posición i(si es valida), se puede utilizar el acceso directo.  Siempre ocupa la misma cantidad de memoria  Por Valor: se copia todo el arreglo  Por referencia se pasa la dirección | Dinámica  Homogénea  Lineal  Al ser una estructura dinámica, cada vez que se necesita agregar un elemento se solicita memoria, por lo tanto no existe relación entre lo posiciones de memoria de los elementos.  Es lineal, es decir que para agregar hay que recorrerla completa o usar un puntero auxiliar.  Siempre se puede hacer la operación.  Debe buscarse el elemento, una vez el elemento encontrado debe reorganizarse los punteros.  Se pide espacio para el elemento.  Siempre se puede hacer la operación  Para acceder a la posición i, se debe pasar por los i-1 elementos anteriores.  La cantidad de memoria depende de la cantidad de elementos de la lista  Por valor: se copia la dirección inicial de la lista.  Por referencia: se pasa la dirección inicial de la lista. |

Pasos para agregar en un vector:

If(dimL+1 <= dimF) then

dimL:=dimL+1;

A[dimL]:= valor

1-Verificar si hay espacio

2-Agregar al final el elemento (Con la dimensión lógica + 1)

3-Incrementar la dimensión Logica.

Pasos para insertar en un vector

If(dimL+1 <= dimF) and (pos<= dimL)and (pos>=1) then

For i:=dimL down to pos do

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

A[pos]:=valor;

dimL:=dimL+1;

1-Verificar si hay espacio

2-Verificar que la posición sea valida

3-Hacer lugar para poder insertar en la posición

4-Incrementar la dimL

Diferencias entre :=nil y Dispose

|  |  |
| --- | --- |
| := nil | Dispone |
| Libera la conexión que existe entre la variable y la posición de memoria.  La memoria sigue ocupada    La memoria no se puede referenciar ni utilizar. | Libera la conexión que existe entre la variable y la posición de memoria  Libera la posición de memoria  La memoria liberada puede ser utilizada en otro momento en el programa |

**Eficiencia:**

Técnicas de corrección: forma de probar que el programa es correcto

* Testing
* Debbugin
* Walkthrougs
* Verificacion

Testing: busca proveer evidencias convincentes de que el programa hace lo esperado. Debe realizar un plan de pruebas (generar distintos casos y analizar si hay errores o no).

Debbugin: proceso de descubrir y reparar la causa del error. Por ejemplo agregar sentencias en medio del código para poder ir siguiéndolo a medida que se ejecuta. Los errores pueden provenir de 2 fuentes:

* Diseño de programa
* Error en la escritura del programa

Errores Sintácticos: Se detectan en la compilación

Errores Lógicos: Se detectan en la ejecución  
Errores de Sistema: Muy raros que ocurran

Walkthroughs: recorrer el programa frente a una audiencia, así de esta forma una persona ajena al código la cual no tenga la carga emocional de haberlo escrito, pueda reconocer el error más fácilmente

Verificación: significa controlar que el programa cumple con el pre-condiciones y post-condiciones del mismo.

**Eficiencia**

**Definición:**

**La eficiencia es la métrica de calidad de los algoritmos asociados 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**

**Tiempo de ejecución**

El análisis de la eficiencia de un algoritmo estudia el tiempo que tarda un algoritmo en ejecutarse y la memoria que requiere.  
El tiempo se puede definir como una función de entrada.

Se puede calcular de dos maneras:

* Análisis empírico: realizar el programa y medir los recursos consumidos.

(Problema: datos únicos para una maquina)

* Análisis teórico: se consideran las instrucciones elementales de un algoritmo. Cada operación elemental (asignación, comparación y una operación aritmética simple) ocupa una unidad de tiempo, la cual es constante independientemente del tipo de dato con el que trabaje.

**Memoria**

Para calcular la memoria se debe hacer la suma de las variables que utiliza el programa multiplicando la cantidad de cada variable por la memoria que esta ocupa:

Tamaño de las variables:

Char = 1byte

Integer = 2 bytes

Real = 6 bytes

Boolean = 1 byte

String = cantidad de caracteres + 1

Registro = la suma de lo que ocupa c/ campo

Puntero = 4 bytes

Lista= ( Dato + Sig ) \* Cant de Elementos

La eficiencia se relaciona con:

Datos de entrada por el tamaño de los datos y la cantidad de datos.

Calidad del código generado por el compilador

**Tiempo de ejecución relación:**

**-Las estructuras de control:** Si afectan el tiempo de ejecución.

**-Cantidad de líneas del programa**: **No** se puede establecer una relación directa

**-Modularización**: Una solución modularizada **no** es necesariamente más eficiente que una sin modularizar. Sin embargo facilita el análisis de eficiencia, dado que puede calcularse el tiempo de ejecución de los distintos procesos uno a la vez.

**-Las variables locales**: tienden a mejorar la legibilidad pero **no** hacen al programa más eficiente.

**-Un algoritmo bien documentado** puede ser menos eficiente que otro

**-Datos de entrada** afecta al tiempo por la cantidad de datos de entrada y el tamaño de los datos.

**Tiempo vs Orden**

Tiempo: suma de la cantidad de unidades de tiempo de cada instrucción.

For i:= 1 to n

X:= 1

If (1>z) then

Writeln(y)

F:= 1+2

Else

F:=x

***T(n): N(cantidad veces For) \* ( 1 (asig.) + 1 (evaluación if) + 2 (suma + asig.)***

***Orden: el caso que mas tiempo toma***.

(entre las instrucciones en este caso 1 y 2)

**T(n):(1)+(2) = [ 100\*(1+3) ] + [ 3 ]**

1) For i:= 1 to 100 do begin ***O(n):100 \*(1+3) = 400 Unidades de Tiempo***

X:= 1

If (1>z) then

Writeln(y)

F:= 1+2 ***WRITE Y READ NO SUMAN COMO UNIDAD DE***

Else ***TIEMPO***

F:=x

End;

2)J:=2+3+2

**Diferencia de espacio entre un arreglo y una lista**

Type  
A= array [1..100] of string;

L=^nodo;

Nodo= record

Dato:string;

Sig:L; end;

**Si ambas están vacías:**

A = 100 \* 256 = 25.600 Byte

L= 4 Bytes

**Si ambas tienen 40 elementos cargados:**

Lista, calcular espacio:  
L= ( Dato + Sig ) \* Cant de Elementos

A= 100 \* 256 = 25.600 Bytes

L= (256+4) \* 40 = 10.400Bytes

**Si ambas tienen 100 elementos cargados:**

A= 100 \* 256 = 25.600 Bytes

L= (256 + 4) \* 100 = 26.000 Bytes

**Es lo mismo hacer esto? Cumplen la misma función?**

Type

A=array [1..10]of integer;

Procedure  
end;

Var  
L1: A;  
begin

{Se usa L1 entre varios procedures}

end;

Type

Procedure  
end;

Var  
L2: array [1..10] of integer;  
begin

{Se usa L2 entre varios procedures}  
end;

Las variables si cumplen la misma función, pero solo la que esta el arreglo declarado en el type es el que se puede usar y pasar entre parámetros.