Resumen para final CADP

▼ Tipos de datos

Dato

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

Tienen

- Rango de valores posibles.
- Conjunto de operaciones permitidas.
- Representación interna.

Pueden ser **simples** o **compuestos**

▼ Simples

Son aquellos que toman un único valor, en un momento determinado.

▼ Compuestos

Pueden tomar varios valores a la vez que guardan alguna relación lógica entre ellos, bajo un único nombre.

1. Tipos de datos definidos por el lenguaje: son provistos por el lenguaje y tanto la representación como sus operaciones y valores son reservadas al mismo.
2. tipos de datos definidos por el programador: permiten definir nuevos tipos de datos a partir de los tipos simples.

▼ Dato numérico

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

▼ Enteros

1. Dato simple
2. Ordinal
3. Forma: -10/200/-30/500..
4. Al tener una representación interna, tienen:
    a. Máximo.
    b. Mínimo

Operaciones

Operaciones matemáticas	Operaciones lógicas	Operaciones entre enteros
+	<	**DIV** ⇒ cociente
-	>	**MOD** ⇒ resto
*	=	
/	≤ ó ≥	

▼ Reales

1. Dato simple
2. forma: -2 / 200 / 3000 / 11,5 / 22,89
3. Representación interna:
    a. Máximo.
    b. Mínimo.

Operaciones

Operaciones matemáticas	Operaciones lógicas
+	<
-	>

*	=
/	≤ ó ≥

▼ Paréntesis

Las expresiones que tienen dos o más operandos requieren reglas matemáticas que permitan determinar el orden de las operaciones.

En caso de que el orden de la procedencia natural deba ser alterado, es posible la utilización de paréntesis dentro de la expresión.

▼ Dato lógico

Permite representaciones de datos que puedan tomar 2 valores

1. Simple.
2. Ordinal.
3. Forma:
    a. True = verdadero.
    b. False = falso.

Operaciones

AND (conjunción)

V	V	V
V	F	F
F	V	F
F	F	F

OR (Disyunción)

V	V	V
V	F	V
F	V	V
F	F	F

NOT (negación)

V	F
F	V

▼ Dato carácter

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

1. Simple.
2. Ordinal.

3. Forma: “a” / “B” / “ ! “ / “%” / “4”

Operaciones

Operaciones lógicas:

< ó >

≤ ó ≥

=

<>

▼ Dato string

Representa un conjunto finito de caracteres. Como máximo representa 256 caracteres. En general se utilizan para representar nombres.

1. Compuesto
2. Forma: “casa” / “231BER”

Operaciones

Operaciones lógicas:

< ó >

≤ ó ≥

=

<>

▼ Variables y constantes

Variables

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

Lo mismo que una variable pero no puede cambiar su valor durante el programa

Especificación

Los diferentes tipos de datos deben ser especificados. Esta especificación dentro de un programa se la conoce como **DECLARACIÓN.**

▼ Clasificación de lenguajes

Strongly types

Verifican que el tipo de los datos asignados a esa variable se correspondan con su definición.

Self types

Verifican el tipo de las variables según su nombre.

Dinamically types

Permiten que una variable tome valores de distinto tipo durante la ejecución de un programa.

▼ Programas

Pre condición

Es la información que se conoce como verdadera antes de iniciar el programa (o módulo).

Post condición

Es la información que debería ser verdadera al concluir el programa (o módulo), si se cumplen adecuadamente los pasos especificados.

▼ Lectura y escritura

Read

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

```
Program UNO;
var
    cant: integer;
begin
    Read(cant);
End.
```

Write

Se usa para mostrar el contenido de una variable.

```
Program UNO;
var
    cant: integer;
begin
    Read(cant);
    write(cant);
End.
```

▼ Estructuras de control

Todos los lenguajes de programación tienen un conjunto mínimo de instrucciones que permiten especificar el control del algoritmo que se quiere implementar. Como mínimo deben contener: secuencia, decisión e iteración.

1. **Secuencia:** La estructura de control más simple, está representada por una sucesión de operaciones en la que el orden de ejecución coincide con el orden físico de aparición de las instrucciones.
2. **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:
3. **Selección:** Permite realizar distintas acciones dependiendo del valor de una variable de tipo ordinal.
4. **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.
    a. Pre condicionales: Evalúan la la condición y si es verdadera se ejecuta el bloque de acciones. Dicho bloque se puede ejecutar 0, 1 o **n** veces.
    b. Post condicionales: Ejecutan las acciones, luego evalúan la condición y ejecutan las acciones mientras la condición es falsa. Dicho bloque se ejecuta 1 o **n** veces.
5. **Repetición:** Es una extensión natural de la secuencia. Consiste en repetir **n** veces un bloque de acciones. Este número de veces es fijo y conocido de antemano.

▼ Tipos de datos definidos por el usuario

TDU

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, solo de debe modificar una declaración en lugar de un conjunto de declaraciones de variables.
- **Documentación**: Se puede usar como identificador de los tipos, nombres auto explicativos, facilitando el entendimiento y lectura del programa.
- **Seguridad:** se reducen los errores por uno de operaciones inadecuadas del dato a manejar, y se pueden obtener programas más confiables.

TDU - Subrango

Es un tipo ordinal que consiste en una sucesión de valores de un tipo ordinal tomado como base

▼ Modularización

Modularizar

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

Subproblemas

- Puede resolverse de forma independiente.
- Está en un mismo nivel de detalle.
- Las soluciones de los subproblemas pueden combinarse para resolver el problema original.

Módulo

- Encapsulan acciones, tareas o funciones.
- Tarea específica bien definida.

- Se comunican entre sí de forma adecuada y cooperan para conseguir un objetivo común.
- En los módulos podemos representar los objetivos relevantes del problema a resolver.

  En la materia usamos la metodología TOP-DOWN

Procedimiento

Es un conjunto de instrucciones que realizan una tarea especifica y retorna 1 o más valores.

```
Procedure nombre(var parametros: indice; parametro: indic
var
    . . . {variables locales}
begin
    . . . {codigo del procedimiento}
End;
```

Funciones

Es un conjunto de instrucciones que realizan una tarea especifica y retorna 1 valor de tipo simple.

```
Function nombre(parametro: indice) : tipo;
var
    . . . {variables locales}
begin
    . . . {codigo del procedimiento}
End;
```

▼ Comunicación entre módulos

La solución a los problemas ocasionados por el uso de variables globales es una combinación de ocultamiento de datos (data hiding) y el uso de parámetros.

Data hiding: Los datos exclusivos de un módulo NO deben ser visibles o utilizables por los demás módulos.

Parámetros: Los datos compartidos se deben especificar como parámetros que se transmiten entre módulos.

Por valor

El modulo recibe un valor, puede realizar operaciones y/o cálculos pero no producirá ningún cambio ni tampoco tendrá incidencia fuera del módulo.

Por referencia

El módulo recibe una dirección, puede realizar operaciones y/o cálculos que producirán cambios y tendrán incidencia fuera del módulo.

Ventajas

1. Cada módulo indica que necesita recibir.
2. Cada módulo indica que devuelve.

▼ 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	Acceso	Tamaño	Linealidad
Homogénea	Secuencial	Dinámica	Lineal
Heterogénea	Directo	Estática	No lineal

1. **Elementos:** Depende de que si los elementos son del mismo tipo o no.
   a. Homogénea: son del mismo tipo.
   b. Heterogénea: son de distinto tipo.
2. **Acceso:** hace referencia a como se pueden acceder a los elementos que lo componen.

   a. Secuencial: para acceder a un elemento particular se debe respetar un orden predeterminado. Por ejemplo, pasando por todos los elementos que le preceden.
   b. Directo: Se puede acceder a un elemento particular, directamente, sin necesidad de pasar por los anteriores a él. Por ejemplo, referenciando una posición.
3. **Tamaño:** hace referencia a si la estructura puede variar su tamaño durante la ejecución del programa.
   a. Estática: El tamaño de la estructura no varía durante la ejecución del programa.
   b. Dinámica: El tamaño de la estructura puede variar durante la ejecución del programa.
4. **Linealidad:** Hace referencia a como se encuentran almacenados los elementos que la componen.
   a. Lineal: está formada por ninguno, uno o varios elementos que guardan una relación de adyacencia ordenada donde a cada elemento le sigo uno y le precede otro.
   b. No lineal: Para un elemento dado pueden existir 0, 1 o más elementos que le suceden o preceden.

Registro

Es un tipo de dato estructurado, que permite agrupar diferentes clases de datos en una única estructura bajo un solo nombre.

Su estructura es **heterogenea** ya que los elementos pueden ser de distinto tipo. Su tamaño es **estático** ya que no cambia durante la ejecución del programa y poseen campos los cuales representa cada uno de los datos que forman el registro.

DEFINICIÓN

```
type
    nombre = record
        dato1: integer;
```

```
        dato2: char;
        dato3: real;
end;:
```

Arreglos

Un arreglo (array) es una estructura de datos compuesta que permite acceder a cada componente por una variable índice, que da la posición del componente dentro de la estructura de datos.

Un vector es un arreglo de una dimensión.

Es **homogenea** ya que los elmentos son del mismo tipo, **estatica** porque el tamaño no cambia durante la ejecución e **indexada** porque para acceder a cada elemento de la estructura, se debe utilizar una variable que es de tipo ordinal.

DEFINICION

```
Type
    Vector = array[rango] of tipo;
```

Puntero

Es un tipo de variable usada para almacenar una dirección en memoria dinámica. En esa dirección de memoria se encuentra el valor real que almacena. El valor puede ser de cualquyier tipo.

Un puntero es un tipo de dato simple.

Una variable de tipo puntero ocupa una cantidad de memoria fija, independientemente del tipo de dato al que apunta.

Puede reservar y liberar memoria durante la ejecución de un programa para almacenar su contenido.

Un dato referenciado o apuntado no tiene memoria asignada, o sea, no existe inicialmente espacio reservado en memoria para este dato.

Listas

Es una colección de **nodos** a base de punteros. Cada nodo contiene un elmento (valor que se quiere almacenar) y una dirección de memoria dinámica que indica donde se encuentra el siguiente nodo de la lista.

Toda lista tiene un nodo inicial

Los nodos que la componen pueden aparecer dispersos en la memoria pero mantienen un orden lógico interno.

Caracteristicas

- Homogenea ⇒ los elmentos son del mismo tipo.
- Dinámica ⇒ El tamaño puede cambiar durante la ejecución de un programa.
- Lineal ⇒ Cada nodo de la lista tiene un nodo que le sigue (salvo el último) y uno que le antecede (salvo el primero).
- Secuencial ⇒ El acceso a cada elmento es de manera secuencial, es decir, para acceder al 5° elemento, primero se debe pasar por los 4 anteriores.

DEFINICION

```
Type
    lista = ^nodo;
    nodo = record
        dato: integer;
        siguiente: lista;
    end;
    var
        L: lista;
    begin
        L := NIL; {Corta enlace con la memoria dinamica.
            Queda ocupada pero no se puede acceder}
        new(L); {Reservar una dirección en memoria dinámi
            para poder asignarle contenido.}
        L^.dato := 8; {Accede al dato apuntado por la lis
        Dispose(Dato); {Libera la memoria que contenia el
```

```
    End.
```

▼ Memoria de un programa

Memoria estática

Se considera solo las variables locales, variables globales del programa y constantes.

Memoria dinámica

Se considera solo cuando en la ejecución de un programa se reserva o libera memoria.

Ejemplo de tabla de ocupación

Char	1 byte
Boolean	1 byte
Integer	6 bytes
Real	8 bytes
String	tamaño + 1 byte
Subrango	depende el tipo
Registro	suma de sus campos
Arreglos	Dimensión física * tipo elemento
Punteros	4 bytes

▼ Corrección de programas

Un programa es correcto si se realizo de acuerdo a sus especificaciones.

Técnicas para corrección de programas

Testing

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

Diseñar un plan de pruebas:

1. Decidir cuales aspectos del programa deben ser testeados y encontrar datos de prueba para cada uno de esos aspectos
2. Determinar el resultado que se espera que el programa produzca para cada caso de prueba.
3. Poner atención a los casos límite.
4. Diseñar casos de prueba sobre la base de la que hace el programa y no de lo escribió el programa. Lo mejor es hacerlo antes de escribir el programa.

Una vez el programa ha sido implementado y se tiene el plan de pruebas:

a. Se analiza el programa con los casos de prueba.
b. Se corrigen los errores hasta solucionar todos.

Debugging

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

Para esto pueden agregarse sentencias adicionales en el programa que permiten monitorear el comportamiento más cercanamente.

Los errores encontrados pueden ser de tres tipos

1. Sintácticos: Se detectan en la compilación.
2. Lógicos: generalmente se detecta en la ejecución.
3. De sistema: son muy raros los casos en los que ocurren.

Walkthrough:

Es el proceso de 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 dispuesta a descubrir errores u omisiones.

- Cuando no se puede detectar un error, el programador trata de probar que no existes, pero mientras lo hace, puede detectar el error o bien puede que otro lo encuentre.

Verificación

Es el proceso de controlar que se cumplen las pre y post condiciones del mismo.

▼ Eficiencia de programas

Una vez que se obtiene un algoritmo y se verifica que es correcto, es importante determinar la eficiencia del mismo.

El análisis de la eficiencia de un algoritmo estudia el tiempo de ejecución de un algoritmo y la memoria que requiere para su ejecución.

Memoria

Se calcula teniendo en cuenta la cantidad de bytes que ocupa la declaración en el programa de

a. Constantes
b. Variables globales
c. Variables locales

Tiempo de ejecución

Puede calcularse haciendo un análisis empírico o un análisis teórico del programa.

El tiempo de un algoritmo puede definirse como una función de entrada.

En algunos algoritmos el tiempo de ejecución no depende de las características de los datos de entrada sino de la cantidad o su tamaño.

Medir el tiempo

Análisis empírico

Requiere la implementación del programa, luego ejecutar el programa en la maquina y medir el tiempo consumido para su ejecución.

Ventajas

- Fácil de realizar.
- Obtiene valores exactos para una máquina determinada y unos datos determinados.

Desventajas

- Completamente dependiente de la máquina donde se ejecuta.
- Requiere implementar el algoritmo y ejecutarlo repetidas veces (para luego calcular un promedio).

Análisis técnico

Implica encontrar una cota máxima (”peor caso”) para expresar el tiempo de nuestro algoritmo, sin necesidad de ejecutarlo.

Solo de consideran las instrucciones elementales del algoritmo: asignaciones y operaciones aritmético/lógicas.

Cada instrucción elemental toma 1 unidad de tiempo (**UT**).