Resumen de Teoría – CADP 2025

Cómo se trabaja

En la Informática se siguen los siguientes pasos a la hora de resolver los problemas del mundo real:

- **Poseer un problema.**
- **Modelizar.**
- **Modularizar.**
- **Realizar el programa:** Se emplean algoritmos y datos.
  - **Algoritmo:** Especificación rigurosa de la secuencia de pasos (instrucciones) a realizar sobre un autómata para alcanzar un resultado deseado en un tiempo finito.
  - **Dato:** 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. Puede ser constante o variable.
- **Utilizarlo en una computadora.**

Pascal: Una introducción a la mayoría de los lenguajes.

Tipos de datos

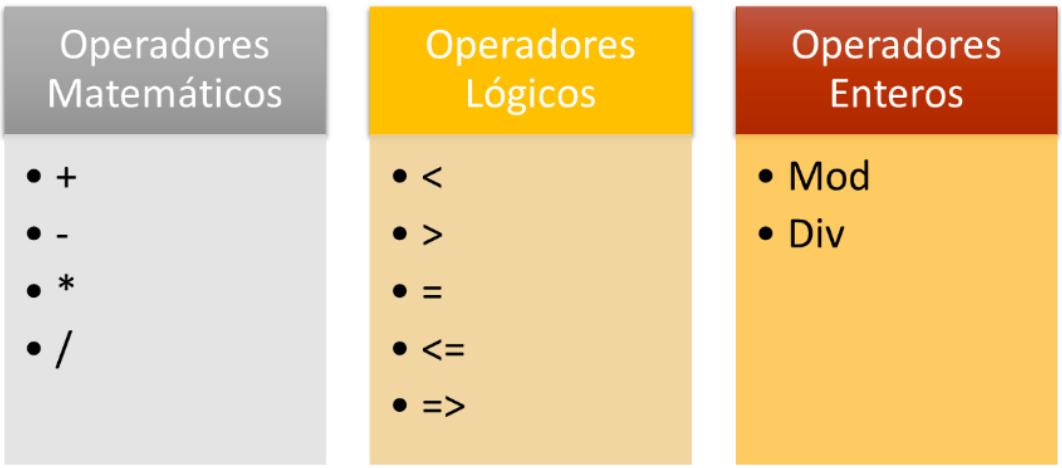
Los tipos de datos que existen se pueden clasificar de dos formas:

- **Simples:** Toman un único valor dentro de los distintos que pueden tomar en un determinado momento. Además, pueden ser ordinales (que se sabe con exactitud el dato anterior o posterior) o no.
- **Compuestos:** Pueden tomar varios valores que guardan una relación lógica entre si al mismo tiempo.

Además, según por quién están definidos se pueden clasificar de las siguiente forma:

- **Definido por el lenguaje.**
- **Definido por el programador:** Tipos que se definen por el programador a partir de los datos ya provistos por el lenguaje. Estos pueden tomar cualquier valor y realizar cualquier operación que se permita en el tipo del que se desprenden, pero no pueden relacionarse con variables que no sean de su mismo tipo.

Con los datos se pueden hacer distintas operaciones:



Cuadro resumen:

Tipos de datos	Cant. de valores	Definidos por...	Operadores permitidos
Integer (Entero)	Simple, ordinal	Lenguaje	Matemáticos, lógicos y enteros.
Real	Simple, no ordinal	Lenguaje	Matemáticos y lógicos.
Boolean (Lógico)	Simple, ordinal	Lenguaje	Lógicos (and, or y not).
Char (Carácter)	Simple, ordinal	Lenguaje	Lógicos (según el orden en la tabla ASCII).
String	Compuesto	Lenguaje	Lógicos.
Subrango	Simple, ordinal	Programador	Los que permita el tipo simple base.
Puntero	Simple	Lenguaje	Asignación con otra variable del mismo tipo; `new` y `dispose`.

Estructuras 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. Se clasifican según distintas cualidades:

- **Por sus elementos:** Homogénea o heterogénea.
- **Según su acceso:** Secuencial o directo.

- **Por su tamaño en memoria:** Dinámica o estática.
- **Según su linealidad:** Lineal o no lineal.

Estructura de Datos	Elementos	Acceso	Tamaño	Linealidad	Caracteristicas
Registro (record)	Heterogéneos	Directo	Estático	Lineal	Permite representar la información en una única estructura; La única operación permitida es la *asignación* con otra variable del mismo tipo; Para saber si dos registros son iguales se debe comparar campo por campo.
Arreglo (array)	Homogéneos	Directo (indexado o a través de un índice)	Estático	Lineal	Permite acceder a cada componente por una variable índice, que da la posición del componente dentro de la estructura de datos; El rango de su índice debe ser de tipo ordinal; Contiene un único tipo de dato, que tiene que ser de tamaño estático; La única operación

					permitida es la asignación con otra variable del mismo tipo.
Lista (`lista = ^nodo`)	Homogéneos	Secuencial	Dinámico	Lineal	Es una colección de nodos. Cada **nodo** contiene un elemento (valor que se quiere almacenar en la lista) y una dirección de memoria dinámica que indica donde se encuentra el siguiente nodo de la lista. Toda lista tiene un nodo inicial; Es una estructura recursiva y el orden de su declaración debe respetarse.

Características particulares

Vectores

En los vectores se hace una distinción entre dos tipos de dimensiones:

- **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 debe superar la dimensión física.

(su aplicación se encuentra detallada en la diapositiva 6-2)

- **Búsqueda en un vector:** Se lo recorre buscando un valor que puede o no estar en el vector. Se debe tener en cuenta que no es lo mismo buscar en un vector ordenado que en uno que no lo esté.
  - **Búsqueda desordenada:**
    1. Inicializar la búsqueda desde la posición 1 (pos).
    2. Mientras ((el elemento buscado no se igual al valor en el arreglo[pos]) y (no se termine el arreglo)) : Avanzo una posición
    3. Determino porque condición se ha terminado el while y devuelvo el resultado.
  - **Búsqueda ordenada:**
    1. Inicializar la búsqueda desde la posición 1 (pos).
    2. Mientras ((el elemento buscado sea menor al valor en el arreglo[pos]) y (no se termine el arreglo)): Avanzo una posición
    3. Determino porque condición se ha terminado el while y devuelvo el resultado.

Corte de control

Para procesar los datos en una estructura podemos aprovecharnos del orden en el que se encuentren ingresados los mismos. Si están ordenados por algún criterio podemos emplear lo que se conoce como **corte de control**. Se implementa en el código utilizando una variable que almacene el dato por el que está ordenada la estructura (por ejemplo una variable *actual*): Mientras el dato que estemos procesando sea igual a la variable previamente inicializada, operamos en la estructura según lo que necesitemos.

Variables y Constantes

Para ambas se reserva un espacio de memoria que va a ser ocupado por alguno de los tipos de datos dichos anteriormente. La dirección inicial de esta zona se asocia con el nombre de la variable. Las primeras **pueden cambiar de valor** en la ejecución del programa, pero las **constantes no**.

Según en donde se declaren varía su alcance. Pueden ser:

- **Globales:** Se pueden utilizar en cualquier lugar del programa. **(EN CADP NO SE UTILIZAN)**
- **Locales al proceso:** Se pueden utilizar nada mas en el proceso en el que sean declaradas.
- **Locales al programa:** Se pueden utilizar nada mas en el programa principal.

Cuando se utiliza una variable la computadora verifica el alcance que le corresponde...

Si es una variable utilizada en un proceso:

1. Se busca si es una variable utilizada en un proceso.
2. Se busca si es una variable utilizada en un proceso.
3. Se busca si es variable global al programa.

Si es una variable usada en un programa

1. Se busca si es variable local al programa.
2. Se busca si es variable global al programa.

Estructuras de Control

Son instrucciones que determinan el control del algoritmo que se esté ejecutando.

Cuadro resumen:

Estructura de Control	Sintaxis	Características
Secuencia	-	El orden de ejecución coincide con el orden físico de aparición de las instrucciones.

Decisión	`if (condición) then`
	`…`
	`else`
	`…`
		Se decide entre dos alternativas a cumplir en base al valor de verdad de la condición.

Selección	`case (variable) of`
	`condicion1: …;`
	`condicion2:…;`
	`end;`
		Permite realizar distintas acciones dependiendo del valor de una variable de tipo ordinal.

Iteración precondicional	`while (condición) do` `…`	Evalúan la condición (conocida previamente) y si es verdadera se ejecuta el bloque de acciones. Dicho bloque se pueda ejecutar 0, 1 ó más veces.

Iteración poscondicional	`repeat`
	`…`
	`until (condición);`
		Ejecutan las acciones luego evalúan la condición y ejecutan las acciones mientras la condición es falsa. Dicho bloque se pueda ejecutar 1 ó más veces.

Repetición	`for i:= vI to/downto vF do` `…`	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 (el índice `i`, que debe ser ordinal).

Modularización

Consiste en dividir un problema en partes funcionalmente independientes, que encapsulen operaciones y datos. Trae muchas ventajas al programa:

- Mayor productividad.
- Reusabilidad.
- Facilidad de mantenimiento.
- Facilidad de crecimiento.
- Mejor legibilidad.

En Pascal existen dos tipos:

- **Procesos:** Conjunto de instrucciones que realizan una tarea especifica y retorna 0, 1 ó más valores.
  - **Sintaxis:** `procedure` *`nombre`* `(par1: tipo; var par2: tipo);`
  - Puede tener variables locales.
  - Puede tener dos tipos de parámetros:
    - ✧ **Por valor:** El módulo recibe un valor, puede realizar operaciones y/o cálculos, pero no producirá ningún cambio ni tampoco tendrá incidencia fuera del módulo.
    - ✧ **Por referencia (`var`):** El módulo recibe una dirección, puede realizar operaciones y/o cálculos, que producirán cambios y tendrán incidencia fuera del módulo.

- **Funcion:** Conjunto de instrucciones que realizan una tarea especifica y retorna 1 valor de tipo simple.
  - **Sintaxis:** `function` *`nombre`* `(parámetros): tipo;`
  - Antes de la finalización de su código se le tiene que asignar si o si un valor.
  - Puede tener variables locales.
  - Solo puede tener parámetros por valor.
  - Se puede invocar usando una *variable*, para evaluar una condición en un *while* o *if*, o para imprimirlo en un *write*.

Alocación estática y dinámica

Todos los tipos de datos que se vieron en CADP se alojan en la memoria estática de la CPU, y permanecen allí ocupando memoria durante toda la ejecución del programa, a pesar de que se sigan utilizando o no. Para solucionar esto, la mayoría de los lenguajes de programación permiten la utilización de tipos de datos que permiten reservar y liberar memoria dinámica durante la ejecución del programa a medida que el programador lo requiera.

Una variable de tipo **puntero** se aloja en la memoria estática, pero puede reservar memoria dinámica para su contenido:

- Cuando quiere cargar contenido reserva memoria dinámica y cuando no necesita más el contenido la libera.
- Siempre ocupa 4 bytes de memoria estática.

Con este tipo de variable se pueden hacer las siguientes operaciones:

- **`new(puntero):`** Se reserva una dirección de memoria dinámica libre para poder asignarle contenidos a la dirección que contiene la variable de tipo puntero.
- **`dispose(puntero):`** Se libera la memoria dinámica que contenía la variable de tipo puntero.
- **`puntero:= nil:`** Se corta el enlace que existe con la memoria dinámica. La misma queda ocupada pero ya no se puede acceder.

dispose(puntero)	**puntero:= nil**
Libera la conexión que existe entre la variable y la posición de memoria.	Libera la conexión que existe entre la variable y la posición de memoria.
Libera la posición de memoria dinámica.	La memoria dinámica sigue ocupada.
La memoria dinámica liberada puede utilizarse en otro momento del programa.	La memoria dinámica **no** se puede referenciar ni utilizar.

Tipo de variable	Bytes que ocupa
Char	1 byte
Boolean	1 byte
Integer	6 bytes
Real	8 bytes
String	Tamaño + 1 (sino se especifica el tamaño es 255 + 1)
Subrango	Depende el tipo
Registro	La suma de sus campos
Vector	Dimensión física * tipo elemento
Puntero	4 bytes

Corrección y Eficiencia

Corrección

Un programa es correcto si se realiza de acuerdo a sus especificaciones. Hay cuatro técnicas para la corrección de programas:

- **Testing:** Se busca proveer evidencias convincentes que el programa hace el trabajo esperado.
  - 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 hechas estas pruebas se analiza el programa con los casos de prueba, y si hay errores se corrigen.
- **Debugging:** Es el proceso de descubrir y reparar la causa del error. Para esto pueden agregarse sentencias adicionales en el programa que permiten monitorear el comportamiento más cercanamente. Los errores que se encuentren pueden ser de tres tipos:
    - **Sintácticos:** Se detectan en la compilación.
    - **Lógicos:** Generalmente se detectan en la ejecución.
    - **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 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:** Es el proceso de controlar que se cumplan las pre y post condiciones del programa.

Eficiencia

La eficiencia de un programa se mide a partir del tiempo de ejecución del algoritmo y la memoria que ocupa el mismo a través de la variables y las constantes. El tiempo de ejecución se puede analizar de dos maneras:

- **Análisis empírico:** Se implementa el programa varias veces y se saca el promedio de duración. Es fácil de realizar pero es completamente dependiente de la maquina en la que se lleve a cabo.
- **Análisis teórico:** Implica buscar un peor caso para expresar el tiempo de nuestro algoritmo sin necesidad de ejecutarlo. A partir de un programa correcto, se obtiene el tiempo teórico del algoritmo y luego el orden de ejecución del mismo.

En el análisis teórico se cuentan solo las instrucciones elementales, es decir la asignación y operaciones aritmético/lógicas **(NO READ NI WRITE)**. Estas mismas suman una unidad de tiempo (1uT).

CADP – EFICIENCIA DE PROGRAMAS T. de EJECUCION

ANALISIS TEORICO

El tiempo de ejecución de un algoritmo que **NO tiene estructuras de control** está dado por:

T (alg) = T(1) + T(2) + T(3) + T(4)

```
Program uno;

var
 aux,temp,x: integer;

Begin
 (1) aux:= 58;
 (2) aux:= aux * 5;
 (3) temp:= aux;
 (4) read (x);
End.
```

T(1)= asignación = 1UT

T(2)= multiplicación + asignación = 2UT

T(3)= asignación = 1UT

T(4)= no se considera

Clase 11-2

El tiempo de ejecución del IF

Tiempo de evaluar la condición + tiempo del cuerpo.

Si hay else, Tiempo de evaluar la condición + max(then,else).

El tiempo de ejecución del FOR

(3N + 2) + N(cuerpo del for).
N representa la cantidad de veces que se ejecuta el for.

El tiempo de ejecución del WHILE

C(N+1) + N(cuerpo del while).
N representa la cantidad de veces que se ejecuta el while.(N>=0)
C cantidad de tiempo en evaluar la condición

El tiempo de ejecución del REPEAT UNTIL

C(N) + N(cuerpo del repeat).
N representa la cantidad de veces que se ejecuta el repeat. (N>0)
C cantidad de tiempo en evaluar la condición