**CADP: resumen**

Paradigmas:

Funcional, Lógico, Imperativo, Orientado a Objetos

Algoritmo: Secuencia de pasos/instrucciones a realizar para obtener un resultado en cierto tiempo

Ordinal: Se usan los números naturales, con el fin de expresar categorías ordenadas.

**Tipos de Datos**

**Datos:** están particionados en 2 tipos

SIMPLES: Almacenan un único valor

Definidos por el lenguaje: (Integer, Real, Bolean, Char)

Tipos de datos básicos.

Definidos por Programador: (Subrango, Puntero)

Nuevo tipo de dato creado a partir de un tipo de dato simple.

Tiene la ventaja de tener flexibilidad, documentación, seguridad

COMPUESTOS: Almacena múltiples valores con cierta relación lógica.

Definidos por el lenguaje: (String)

Definidos por el programador: (Registros, Arrays, Listas)

Declaración:

Fuertemente-Tipados (Strongly typed)

Requiere la previa especificación del tipo de dato de cada variable.

Auto-Tipados (Self Typed)

Define el tipo dela variable según su nombre.

Dinámicamente-Tipados (Dinamically typed):

La variable toma el tipo durante la ejecución del programa.

**Estructuras de Datos**

Se clasifican en:

Elementos **Homogéneos / Heterogéneos**

Es decir si los elementos que la componen son del mismo tipo o no.

Acceso  **Secuencial / Directo Indexado**

El acceso a los elementos es si o si en línea, o se puede acceder al medio.

Tamaño **Dinámico / Estático**

Si el tamaño de la estructura puede cambiar durante la ejecución del programa

Linealidad **Lineal / No lineal**

Respecto al almacenamiento de datos, a cada elemento le sigue y precede únicamente un elemento (lineal) o puede precederle 1 o más elementos.

**Registros / record:** (Heterogéneo, Estático)

Permite guardar distintos tipos de datos bajo un mismo nombre

**Vectores / Arrays:** (Homogéneo, Estático, Indexado)

Un grupo de elementos del mismo tipo que se almacenan consecutivamente en memoria, se necesita un índice (ordinal) para acceder al valor.

Los vectores utilizan un Dimensión Física y una Lógica para saber cuánto espacio total tiene y cuánto va usando.

**Listas:** (Homogénea, Dinámica, Lineal, Secuencial)

Las listas son un método de almacenamiento dinámico el cual está compuesto por una fila de **NODOS** cada nodo almacena un dato y la dirección de memoria al siguiente nodo. Por lo que se debe recorrer todos los nodos desde el nodo inicial hasta encontrar el deseado.

**Estructuras de Control**

Pre-condición: Dicha información se considera Verdadera antes de iniciar el modulo

Post-condición: Es la instrucción que debe ser verdadera al terminar el programa o modulo

Secuencia:

El básico, las instrucciones se ejecutan en el orden físico de aparición.

Decisión: (IF)

Estructura de **condición** la cual devuelve un valor V o F

Iteración:

Son estructuras de repetición condicionales. Se dividen en Pre-Condicionales y Post-Condicionales.

Es decir: -Evalúa la condición y ejecuta (While).

-Ejecuta y después evalúa la condición (Repeat-until).

Repetición: (For / Repeat)

Repite N veces la secuencia deseada. Dicho N debe ser fijo y conocido de antemano

Selección: (Case)

Ejecuta determinada acción dependiendo según el valor de X variable ordinal.

**Modularización**

Mayor productividad, reusabilidad, facilidad del mantenimiento facilidad del crecimiento, Legibilidad

El paso de variables puede ser:

Variables por valor: Recibe un valor directo el cual es almacenado en una variable local

Variables pro referencia: Recibe la dirección de memoria de una variable. Es decir puede modificar dicha variable.

**CORRECCIÓN**

El proceso de corrección se separa en 4 etapas / formas.

**Testing:**

Se basa en obtener evidencias que el programa hace lo esperado. Para esto se realiza un Plan de pruebas, en este se eligen los conceptos a ser testeados (como los resultados esperados para cada caso, haciendo énfasis en los casos limites), estos casos es recomendable elegirlos antes de hacer el programa.

**Debugging:**

Es el proceso de descubrir y reparar la causa del error. Se hace agregando sentencias adicionales para poder ir controlando su comportamiento más de cerca.

**Walkthroughs:**

Se le denomina de esta forma al presentar el programa a alguna persona anexa al proyecto, de esta forma ellos te pueden ayudar a detectar el problema o incluso darte cuenta vos mismo.

**Verificación:**

Corroborar que se cumplan las pre y post condiciones del programa

Los errores se organizan en 3 tipos:

-Errores sintácticos, son los que saltan durante la compilación

-Errores Lógicos, se suelen detectar durante la ejecución del programa

-Errores de sistema, no suelen ocurrir pero se pueden generar por alguna incompatibilidad o error.

**EFICIENCIA**

La eficiencia es el **tiempo** que tarda un algoritmo en ejecutarse y la **memoria** que requiere.

Para la eficiencia se tiene en cuenta los datos de entrada (tamaño y calidad), calidad del código generado por el compilador naturaleza y rapidez de ejecución y tiempo del algoritmo base

Entonces las 2 ramas fundamentales son:

TIEMPO:

Para el análisis del tiempo de ejecución se emplean 2 métodos

El análisis empírico, que es fácil ya que se realiza ejecutando el algoritmo reiteradas veces, pero tiene como desventaja que dicho tiempo de ejecución varía según las características del dispositivo donde fue ejecutado

**El análisis teórico**, se realiza calculando el tiempo de ejecución (UT) de cada una de las instrucciones. Para dicho cálculo se toma que cada operación elemental (**asignación y operaciones aritmético / lógicas**) tarda **1 UT**

For i:= 1 to 10 (n = 10) == 3n + 2 + (int \* n)

While == n + 1 + (int \* n)

Repeat until == n + (int \* n)

If == 1

MEMORIA estática / dinámica

**Variables Estáticas:** (Variables locales y globales)

Las variables estáticas reservan determinado espacio de memoria al comienzo del programa y dicho espacio no puede ser modificado durante la ejecución del mismo

**Variables Dinámicas:** (new / dispose)

Su dimensión en memoria puede ir variando permanentemente durante la ejecución del programa

Los accesos a estas direcciones de memoria se guardan en **PUNTEROS**, dichos punteros ocupan únicamente 4 bytes.

La creación de un nuevo espacio de memoria dinámica se hace con un **new (** puntero **)**

y con **dispose (** puntero **)** se borra.

Puntero ( ^ ): Un puntero es una variable usada para almacenar una dirección de memoria dinámica.

El cálculo de Memoria Dinámica y Memoria Estática se realiza para:

Memoria Estática: Sumando todas las variables declaradas en el programa principal y las globales.

Memoria Dinámica: Sumando / restando los New / Dispose, por cada new se suma lo que ocupa el tipo de dato de ese new y por cada dispose se resta.

**Memoria:**

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)