**Tema 1 - Tipos de datos abstractos**

**¿qué es una estructura de datos?**

Es la organización de información que va más allá que la definición misma de un tipo de dato
Pueden contener diferentes tipos o ser del mismo tipo y en forma abstracta representar otra cosa.

**Ventajas de las estructuras**

1. Ahorran tiempo
2. Son eficientes y robustas
3. Se separan en dos capas de código: algoritmo programador y las rutinas de acceso
4. Estimaciones bastantes uniformes de tiempo
5. Las funciones asociadas a la estructura son independientes del lenguaje
6. Una vez que se implementa el código es fácil implementarlo en otros lenguajes

**¿Qué es la abstracción/encapsulamiento?**

Se refiere a dividir el código en varias funciones que hagan ciertas tareas por separadas y esas funciones solo sean llamadas en algún menú o interfaz

**¿Qué es un tipo de dato abstracto?**

Descripción matemática de un objeto abstracto que se define por las operaciones que actúan sobre el

**Tipos de datos abstractos**

**¿Qué hacen?**

Encapsulan datos y funciones que trabajan con este tipo de datos

Es especificado por lo métodos

Manejan memoria dinámica
Esto permite al usuario crear tipos de datos y estructuras de cualquier tamaño de acuerdo a lo que ocupe para su programa

No son visibles y solo pueden ser llamadas por medio de funciones (métodos)

**¿qué son las listas?**

Colección de elementos del mismo tipo con relación LINEAL entre sus elementos, pueden estar en cualquier orden y se puede acceder a cualquier elemento de la lista

Representación enlazada: el orden real no es necesariamente al orden lógico, este se determina por un campo de enlace

Representación secuencial: el orden real coincide con el lógico

Ventaja: se evitan movimiento al insertar y eliminar elementos

Ejemplo: arrays

**¿qué son las colas?**

Colección de elementos del mismo tipo ORDENADOS CRONOLOGICAMENTE**, solo se pueden añadir elementos por un extremo (FINAL) y sacarlos por el otro lado (ENFRENTE)**, también llamada estructura FIFO

**¿Qué son las pilas?**

Colección de elementos del mismo tipo ORDENADOS CRONOLOGICAMENTE, **solo se pueden añadir y extraer elementos por el mismo extremo,** también llamada estructura LIFO

**¿Qué son los árboles?**
Estructura de elementos del mismo tipo con relación JERARQUICA establecida entre ellos.

**1.2 – modularidad**
**¿Qué es modularidad?**
Básicamente es descomponer un problema grande en sub problemas, para así tener menor margen de error y poder hacer el programa más fácil

**Modulo independiente**
**¿Qué nos permite hacer?**
Nos permite concentrarnos en una sola parte del problema, olvidando el resto

Nos permite reutilizar el código, ya sea dentro del mismo problema o dentro de otro problema

Cada modulo se codifica dentro del programa como un subprograma

**1.3 – uso de TDA**
**Uso de TDA (Tipos de Datos Abstractos) en estructuras**
Ayudan al programador a tener mejor visualización de la información, también otorgan un beneficio a ala hora e desarrollar la idea de la vinculación de los datos

**Uso de TDA (Tipos de Datos Abstractos) en herencia**
Se relacionan principalmente con las clases, la herencia siempre está presente siempre y cuando una estructura “struct”, posea otra estructura

En c++ se presentan dos tipos:
- Herencia por agregación o composición
- Herencia por extensión

Los miembros de una estructura pueden ser:
- Ellos mismos
- Otra estructura ya creada
- O una estructura anónima

**Uso de TDA (Tipos de Datos Abstractos) en uniones**
Los datos conviven en el mismo espacio, esto significa que, si afectas una variable, afectara a las demás.

**Uso de TDA (Tipos de Datos Abstractos) en clases**
las clases son como SUPER estructuras capaces de aguantas no solo atributos/datos, sino también métodos/funciones.

Los miembros de una clase son privados

**1.4 manejo de memoria estática**
Un bit representa un dato, este solo puede contener dos estados de información el 0 y 1.

Durante la ejecución del programa el tamaño de la estructura no cambia

Los vectores son quienes manejan la memoria estática

Componentes: los componentes hacen referencia a los elementos que forman el arreglo
Índices: permiten referirse a los componentes del arreglo

Arreglos usa memoria estática

**1.5 manejo de memoria dinámica**
Esta memoria durante la ejecución del programa si puede cambiar el tamaño de la estructura

Es el espacio de almacenamiento que solicita una clase o método cuando se ejecuta el programa

Listas y arboles usan memoria dinámica

Se requiere un elemento llamado nodo para usar la memoria dinámica

**Tema 2 - listas**
**¿Qué es una lista?**
Son dinámicas, pues pueden cambiar su valor

Sus elementos se llaman nodos

Usan listas **enlazadas, doble enlazadas y circulares**

Es una estructura flexible, pueden crecer y decrecer cuando se requiera, se insertan y se eliminan elementos en los extremos como en cualquier otra posición de la lista

El orden de los nodos se establece por medio de los punteros o apuntadores

Una lista es una colección de elementos llamados nodos

Constan de dos partes
1. Campo información
   Es la información que se necesita para el proceso
2. Campo liga
   Es de tipo puntero para establecer el enlace con otros nodos

**2.2 características de las listas**

1. Optimizan el uso de los recursos del sistema
2. Se puede obtener posiciones de memoria a medida que se trabaja con el sistema
3. Son una colección de datos ordenados
4. Liberan espacios de memoria cuando ya no se usan
5. Se pueden repetir elementos
6. Son una estructura lineal
7. Cada elemento tiene un índice que lo ubica
8. Con esta estructura se puede maneja la memoria de manera flexible

**2.3 descripción matemáticamente de las listas**
Matemáticamente se puede llamar elem_t

**2.4 operaciones con listas**
El nodo especial es el primero

**Insertar elementos**
- **Insertar un elemento en una lista vacía**

  **Nodo->siguiente apunte a null**
  **Lista apunte a nodo**

- **Insertar un elemento en la primera posición de una lista**
  Hacemos que nodo->siguiente apunte a lista

- **Insertar un elemento en la última posición de una lista**
  + necesitamos que un puntero señale al ultimo elemento de la lista. La manera de conseguirlo es empezar por el primero y avanzar hasta que el nodo que tenga como siguiente el valor null

  + hacer que nodo->siguiente sea NULL
  + hacer que ultimo->siguiente sea nodo

Para recorrer una lista siempre partiremos del mismo modo, usar un puntero auxiliar como índice
1.- asignamos al puntero índice el valor de la lista
2.- abriremos un bucle que al menos debe tener una condición, que el índice no sea null
3.- dentro del bucle asignaremos al valor índice el valor del nodo siguiente al índice actual

**Listar todos los elementos de una lista**

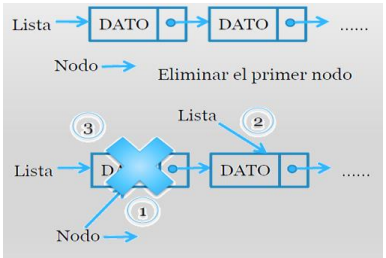
```
typedef struct _nodo {
      int dato;
      struct _nodo *siguiente;
} tipoNodo;

typedef tipoNodo *pNodo;
typedef tipoNodo *Lista;
...
pNodo indice;
...
indice = Lista;
while(indice) {
      printf("%d\n", indice->dato);
      indice = indice->siguiente;
}
```

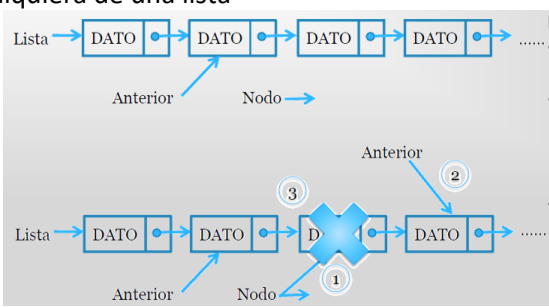
- **TipoNodo** es el tipo de declarar nodos, evidentemente
- **pNodo** es el tipo para declarar punteros a un nodo
- **Lista** es el tipo de declarar listas abiertas

**Eliminar elementos de una lista**
Eliminar el primer nodo de una lista

Es el caso más simple. Se parte de una lista con uno o más nodos y usaremos un puntero auxiliar
- Hacemos que nodo apunte al primer elemento de la lista, es decir a lista
- Asignamos a lista la dirección del segundo nodo de la lista: Lista->siguiente
- Liberamos la memoria asignada al primer nodo, el que queremos eliminar




Eliminar un nodo cualquiera de una lista




**El proceso es parecido al del caso anterior:**
- **Hacemos que nodo apunte al nodo que queremos borrar.**
- **Ahora, asignamos como nodo siguiente del nodo anterior, el siguiente al que queremos eliminar: anterior->siguiente = nodo->siguiente.**
- **Eliminamos la memoria asociada al nodo que queremos eliminar.**

En una lista abierta solo se puede mover hacia adelante

Primer elemento
En una lista abierta el primer elemento siempre es el más accesible ya que nuestro puntero apunta a ese elemento

Elemento siguiente a uno cualquiera
Suponiendo que tenemos un puntero nodo que señala a un elemento cualquiera en la lista, para obtener un puntero al siguiente solo hay que asignarle al campo “siguiente” del nodo

Nodo->siguiente

Elemento anterior a uno cualquiera
Como no se puede retroceder en una lista, partiremos del primer elemento hasta llegar al elemento que esta antes del elemento requerido

Ultimo elemento
Para obtener un puntero al ultimo elemento podemos partir de cualquier nodo hasta llegar al final

Borrar una lista
Consiste en borrar el primer elemento sucesivamente mientras la lista no este vacía

Saber si una lista está vacía
Basta con comparar el puntero lista con NULL

**2.1 listas enlazadas**
Se puede definir como una colección de nodos o elementos
El orden entre estos se establece por medio de punteros

El apuntador al inicio de la lista es importante porque permite posicionarnos en el primer nodo
Si por alguna razón se pierde ese apuntador, perderemos toda la información que hay en esa lista

**Vectores**
Cuando sabes el número máximo de elementos que va a tener tu estructura de datos o si crece muy lento

Cuando necesitas ver de forma aleatoria los elementos

Cuando necesitas insertar y extraer elementos al final del vector

**Listas enlazadas**
Cuando no sabes el número de elementos que vas a insertar

Cuando necesito extraer elementos del principio o del final

Cuando uses estructuras como pilas o colas para insertar o extraer elementos

Para implementar estructuras de datos complejas como arboles

Cuando importe más el espacio de memoria que la velocidad en que se accede a los elementos

**Desventajas de usar listas enlazadas**
Los nodos se deben de leer en orden desde el principio

Los nodos se almacenan de forma no contigua

Son incomodas para navegar hacia atrás

Utilizan más memoria que los arreglos

**2.2 listas doblemente enlazadas**
es una lista lineal en la que cada nodo tiene dos enlaces, uno al siguiente y otro al anterior

no necesitan un nodo especial para acceder a ellas, se puede recorrer en ambos sentidos a partir de cualquier nodo

se comporta como dos listas abiertas que comparten los datos

**2.3 listas circulares**
una lista circular es en donde el ultimo nodo apunta al primero

**tema 3 – pilas**
su nombre proviene a la metáfora de una pila de platos

el ultimo elemento que entre es siempre el único accesible

**¿Qué es una pila?**
se insertan y se extraen elementos de la pila por el principio (también llamado parte superior), estructura LIFO

forma de construir una pila
```
struct nodo {
int dato;
struct nodo
*siguiente;
}
```

Es importante que nunca se pierda el valor del puntero al primer elemento

El primer elemento siempre es en realidad el ultimo elemento de la pila

**3.1 características de las pilas**
Las operaciones se conocer como “push” (empujar) “pop” (tirar)

Son estructuras LIFO, el ultimo en entrar es el primero en salir

Ejemplos de usos de pilas
- navegador web: donde se almacenan los sitios visitados recientemente
- editores de texto: donde se almacenan en una pila, el usuario puede deshacer los cambios mediante la operación undo

**operaciones con pilas**
- insertar un elemento: push (empujar)
- leer y retirar un elemento: pop (tirar)
- verificar si la pila esta vacia: stackempty
- saber cual es el elemento en la cima de la pila: stacktop

**push**
esta operación sirve para insertar elementos
insertaremos un elemento E en la pila S, lo vamos a escribir como

**push(S,e)**

después de esta operación sucede que:
el elemento en la cima de la pila S ahora es e