# Tipos de Datos Abstractos (TDA)

Estructura de Datos

# Datos, tipos de datos y TAD

- **Datos:** son los valores que manejamos en la resolución de un problema, tanto los valores de entrada, como los de proceso y los de salida.

- **Tipo de dato:** un conjunto de valores y un conjunto de operaciones definidas por esos valores.

- **Tipos de datos abstractos:** extienden la función de un tipo de dato ocultando la implementación de las operaciones definidas.

  Esta capacidad de ocultamiento nos permite desarrollar software *reutilizable y extensible*.

# Principales ventajas

- Las principales ventajas que nos aportan los TAD son las siguientes:
  1. Mejoran la conceptualización y hacen más claro y comprensible el código.
  2. Hacen que el sistema sea más robusto.
  3. Reducen el tiempo de compilación.
  4. Permiten modificar la implementación sin que afecte al interfaz público.
  5. Facilitan la extensibilidad.

# Modularidad

- La programación modular *descompone* un programa en un pequeño número de abstracciones independientes unas de otras pero fáciles de conectar entre sí.

- Refleja la independencia de la *especificación y la implementación*. Un TAD puede funcionar con diferentes implementaciones.

- Ventaja que permite distribuir de mejor manera el trabajo y facilita la detección de fallos.

# Abstracción

- **La abstracción como proceso:** consiste en separar las propiedades esenciales de un objeto, sistema, fenómeno o problema y/u omitir las propiedades no esenciales.

- **La abstracción como producto:** es una descripción o especificación de un sistema en el que se enfatiza algunos detalles o propiedades y se suprimen otros.

# Propiedades de un TDA

- **Encapsulación:** un TDA encapsula (protege) ciertos tipos de datos y operaciones con el objetivo de localizar en un punto determinado de su programa la especificación del TDA.

- **Generalización:** se pueden definir sus propios tipos de datos y sus propias operaciones con el objeto de aplicarlos a operandos que no necesariamente tiene que se de un tipo fundamental.

# Tipos de TDA

**TAD’s Estáticos.**

- La creación y mantenimiento de un TAD estático requiere de memoria no dinámica, es decir, el espacio en memoria para almacenar los datos es reservado en tiempo de compilación. (Arreglos).

**TAD’s Dinámicos (Asignación dinámica de memoria).**

- La creación y mantenimiento de estructuras dinámicas de datos (TAD’s dinámicos), requiere de obtener más espacio de memoria (reservar memoria) en tiempo de ejecución para almacenar datos o para almacenar el tipo de clase “Nodo”

# TAD más comunes

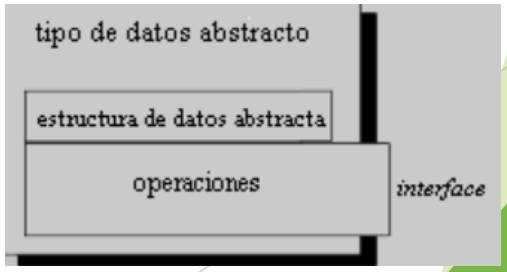
- Los tipos abstractos de datos básicos se clasifican habitualmente, atendiendo a su estructura, en:
  - Lineales
    - Listas
    - Colas
    - Pilas
  - No lineales
    - Árboles
    - Árboles binarios de búsqueda
    - Grafos

# Definición

- Un TAD se define como una estructura algebraica compuesta por un conjunto de objetos abstractos que modelan elementos del mundo real, y un conjunto de operaciones para su manipulación.

- Las partes que forman un TAD son:
  - **Atributos:** tipos de datos, identificadores, etc.
  - **Funciones:** (rutinas) que definen las operaciones válidas para manipular los datos (atributos).

- Un TDA al momento de la implantación, debe de cumplir con las especificaciones algebraicas de sus operaciones:
  - **Sintaxis:** Forma de las operaciones
  - **Semántica:** Significado de las operaciones



# Especificaciones

La **especificación informal** utiliza el lenguaje natural describiendo los valores que pueden tomar los datos de ese tipo prescindiendo de cómo se codifiquen y describiendo las operaciones mediante un esquema
La **especificación formal** nos permite definir el TDA con precisión, y de forma más rigurosa. Normalmente se utiliza el siguiente esquema:

**Tipo:** Nombre del tipo de dato.

**Sintaxis:** Forma de las operaciones. Se suministra una lista de las funciones de la abstracción, indicando el tipo de los argumentos y del resultado.

**Semántica:** Significado de las operaciones. Se indica el comportamiento de las funciones definidas sobre la abstracción.

# Especificaciones - Ejemplo

| | |
|---|---|
| Tipo: | Natural |
| Sintaxis: | Cero->Natural |
| | Sucesor(Natural)->Natural |
| | EsCero(Natural)->Booleano |
| | Igual(Natural, Natural)->Booleano |
| | Suma(Natural, Natural)->Natural |
| Semántica: | Para todo n, m perteneciente Natural |
| | EsCero(Cero)=>Verdadero |
| | EsCero(Sucesor(n))=>Falso |
| | Igual(Cero, n)=>EsCero(n) |
| | Igual(Sucesor(n), Cero)=>Falso |

# Especificación sintáctica

- Se tratan las funciones u operaciones que actúan sobre las instancias del TDA, se definen los nombres, dominios y rangos de dichas funciones.

- Estas operaciones se pueden clasificar de la siguiente manera:
  - **Operación constructor:** esta operación produce una nueva instancia para el tipo de dato abstracto, proveyendo al usuario de una capacidad para generar dinámicamente instancias de nuevos objetos y asignarles valores por defecto a las propiedades (atributos) del TDA. Aparta memoria principal

# Especificación sintáctica

- **Operación destructor:** elimina aquellas instancias del tipo de dato que el usuario tiene en desuso. Libera memoria principal.

- **Operación de acceso:** permite al usuario obtener elementos que sólo son propiedades del tipo de dato del sistema.

- **Operación de transformación:** producen nuevos elementos del tipo de dato abstracto, partiendo del elemento ya existente y posiblemente de otros argumentos.

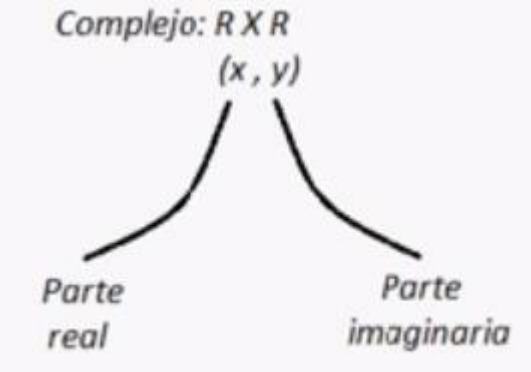
# Ejemplo: número enteros

- Consideremos el tipo de datos de los enteros que ofrece el lenguaje C++, la definition del TDA correspondiente consiste en determinar:

  - **Valores:** los números enteros dentro del intervalo [minint, maxint]
  - **Operaciones:** la suma, la resta, el producto, el cociente y el resto de la división, y
  - **Propiedades de las operaciones**, por ejemplo la propiedad conmutativa: a+b=b+a, Orden de operaciones, primero * /, luego se + o -, etc

# Ejemplo: número complejos

- Los TDA permite diseñar nuevos tipos de datos, por lo que se requiere los valores O + Oi (parte real y la parte imaginaria)




```
struct base_complejo{
        double x;
        double y;
};

typedef base_complejo* complejo;
```

```
struct base_A{
        A1 x1;
        A2 x2;
        .
        .
        .
        An xn;
}
typedef base_A A;
```

# Ejemplo: número complejos

- **Constructor**
- Permite construir una variable.

*crear_complejo: R x R -> C*

*(x, y) -> (x, y)*

```
complejo crear_complejo(double x, double y){
        complejo x=new base_complejo;
        z->x=x;
        z->y=y;
        return z;
}
```

# Ejemplo: número complejos

- **Destructor**
- Devuelve espacio al sistema operativo, los tipos de datos primitivos no lo requieren. En tipos de datos no primitivos no está definido.

*liberar_complejo: C -> $\phi$*

*z*

```
void liberar_complejo(complejo z){
     delete z;
}
```

# Ejemplo: TDA en arreglo

- **Definición del TDA**

```
struct E_Libro{
        char* titulo;
        char* autor;
        char* editorial;
        int precio;
};

typedef E_Libro* libro;
```

# Ejemplo: TDA en arreglo

- **Definición función crear libro**

```
libro crear_libro(){
      libro x=new E_Libro;
      x->titulo=new char[50];
      x->autor=new char[50];
      x->editorial=new char[30];
      return x;
}
```

- **Definición función liberar libro**

```
void liberar_libro(libro x){
      delete[] x->titulo;
      delete[] x->autor;
      delete[] x->editorial;
      delete[] x;
}
```

# Ejemplo: TDA en arreglo

- **Definición función leer libro**

```
libro leer_libro(libro x){
      cout<<"Digite el titulo del libro: ";
      cin.getline(x->titulo,50);
          while(strlen(x->titulo)==0){
                cin.getline(x->titulo,50);
          }
      cout<<"Digite el autor del libro: ";
      cin.getline(x->autor,50);
          while(strlen(x->autor)==0){
                cin.getline(x->autor,50);
          }
      cout<<"Digite la editorial del libro: ";
      cin.getline(x->editorial,30);
          while(strlen(x->editorial)==0){
                cin.getline(x->editorial,50);
          }
     cout<<"Digite el precio del libro: ";
     cin>>x->precio;
         while(x->precio<0){
               cout<<"No valido, digite el precio del libro";
               cin>>x->precio;
         }
     return x;
}
```

# Ejemplo: TDA en arreglo

- **Definición función imprimir libro**

```
void imprimir_libro(libro x){
     cout<<"\nInformacion del libro\n";
     cout<<"-------------------------\n";
     cout<<"Titulo: \t"<<x->titulo;
     cout<<"\nAutor: \t\t"<<x->autor;
     cout<<"\nEditorial: \t"<<x->editorial;
     cout<<"\nPrecio: \t"<<x->precio<<"\n\n";
}
```

# Ejemplo: TDA en arreglo

- **Definición función leer conjunto de libros**

```
libro* leer_conjunto_libros(libro* c, int n){
      for(int i=0;i<n;i++){
                cout<<"\n\nLibro "<<i+1<<":\n\n";
                libro a=crear_libro();
                a=leer_libro(a);
                c[i]=a;
     }
     return c;
}
```

# Ejemplo: TDA en arreglo

- **Definición función imprimir conjunto de libros**

```
void imprimir_conjunto_libros(libro* c, int n){
     cout<<"TITULO \t\tAUTOR \t\tEDITORIAL \t\tPRECIO \n";
     for(int i=0;i<n;i++){
             cout<<c[i]->titulo<<"\t\t"<<c[i]->autor<<"\t\t";
             cout<<c[i]->editorial<<"\t\t\t"<<c[i]->precio<<"\n";
     }
}
```

# Ejemplo: TDA en arreglo

- **Definición función ordenar libros por precio**

```
libro* ordenar_por_precio(libro* c, int n){
        libro aux;
        for(int i=0;i<n;i++){
                for(int j=i+1;j<n;j++){
                        if(c[j]->precio<c[i]->precio){
                                aux=c[i];
                                c[i]=c[j];
                                c[j]=aux;
                        }
                }
        }
        return c;
}
```

# Ejemplo: TDA en arreglo

- **Definición función main**

```
int main(){
    int n=0;
    cout<<"Cuantos libros desea ingresar? \n";
    cin>>n;
    libro* conjunto=new libro[n];

    conjunto=leer_conjunto_libros(conjunto,n);
    cout<<"\n_____\n";
    cout<<"\nLIBROS INGRESADOS POR EL USUARIO\n";
    cout<<"_____\n\n";
    imprimir_conjunto_libros(conjunto,n);
    conjunto=ordenar_por_precio(conjunto,n);

    cout<<"\n_____\n";
    cout<<"\nLIBROS ORDENADOS POR PRECIO\n";
    cout<<"_____\n\n";
    imprimir_conjunto_libros(conjunto,n);


    system("pause");
}
```

# GRACIAS