ALGORITMOS Y ESTRUCTURAS DE DATOS
Programador Universitario
ALGORITMOS Y ESTRUCTURAS DE DATOS I
Licenciatura en Informática – Ingeniería en Informática

Mg. Griselda María Luccioni
Lic. María Cristina Werenitzky Curia
Ing. Gabriela Odstrcil

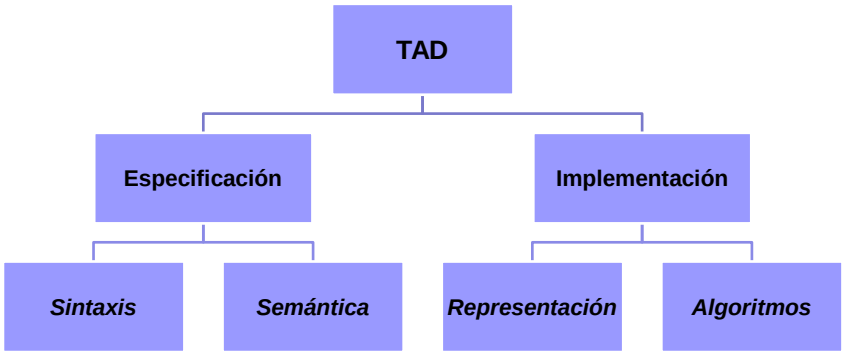
TPN°4: Especificaciones algebraicas. Tipos abstractos de datos

Algoritmos y Estructuras de Datos

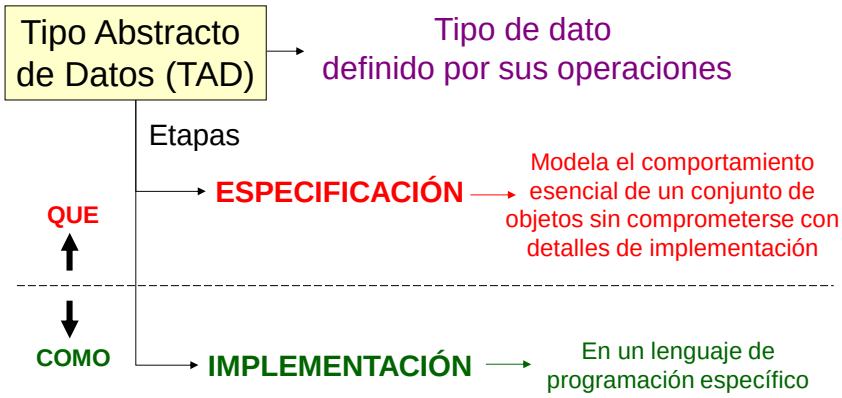
TIPOS ABSTRACTOS DE DATOS

Tipos abstractos de datos

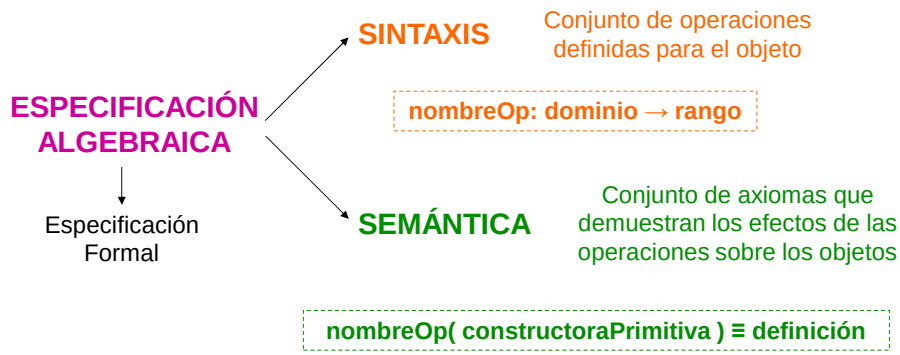
Colección de valores y de operaciones definidas sobre ellos mediante una especificación independiente de cualquier representación



Tipos Abstractos de Datos



Tipos Abstractos de Datos



Tipos Abstractos de Datos

SINTAXIS → Establece el nombre, dominio y rango de cada operación del TAD

nombreOp: dominio → rango

TIPOS DE OPERACIONES

- Constructoras Primitivas (construyen objetos del tipo)
- Constructoras/modificadoras
- Selectoras (separan distintas partes del objeto)
- Test (deciden sobre alguna propiedad de objetos del tipo)
- Otras operaciones

Tipos Abstractos de Datos

SEMÁNTICA → Establece los axiomas que determinan el comportamiento de cada operación del TAD

nombreOp(constructoraPrimitiva) ≡ definición

CONSTRUCTORAS PRIMITIVAS

Operaciones para las cuales no se dan axiomas

Se utilizan para definir los axiomas de las otras operaciones del TAD

Definición

- ✓Valores
- ✓Constructoras
- ✓Operaciones
- ✓Invocación a la misma operación
- ✓Condicionales

TAD CADENA
ESPECIFICACIÓN

ESPECIFICACIÓN
TAD CADENA

SINTAXIS

Constructoras primitivas

NULA: → CADENA

ESNULA: CADENA → BOOL

LARGO: CADENA → ENTERO ≥ 0

AGREGAR: CADENA X CHAR → CADENA

CONCAT : CADENA X CADENA → CADENA

ESPECIFICACIÓN TAD CADENA

¿Cómo construimos la cadena "HOLA" ?

CONSTRUCTORAS

NULA: → CADENA

AGREGAR: CADENA X CHAR → CADENA

AGREGAR(AGREGAR(AGREGAR(AGREGAR(NULA , **H**) , **O**) , **L**) , **A**)

ESPECIFICACIÓN TAD CADENA

Planteamos axiomas para cada operación utilizando las 2 constructoras primitivas
NULA: → CADENA
AGREGAR: CADENA X CHAR → CADENA

SEMÁNTICA ∀ C ∈ CADENA, ∀ x ∈ CHAR

ESNULA: CADENA → BOOL

ESNULA(NULA) ≡ true

ESNULA(AGREGAR(C, x)) ≡ false

LARGO: CADENA → ENTERO ≥ 0

LARGO(NULA) ≡ 0

LARGO(AGREGAR(C, x)) ≡ 1 + LARGO(C)

ESPECIFICACIÓN TAD CADENA

LARGO(NULA) ≡ 0

LARGO(AGREGAR(**C**, **x**)) ≡ 1+LARGO(**C**)

¿Cuál es el largo de la cadena "HOLA" ?

LARGO(AGREGAR(AGREGAR(AGREGAR(AGREGAR(NULA , H) , O) , L) , A) **)**

1 + **LARGO(** AGREGAR(AGREGAR(AGREGAR(NULA , H) , O) , L) **)** **= 4**

1 + **LARGO(** AGREGAR(AGREGAR(NULA , H) , O) **)** **= 3**

1 + **LARGO(** AGREGAR(NULA , H) **)** **= 2**

1 + **LARGO(** NULA **)** **= 1**

0

ESPECIFICACIÓN TAD CADENA

Planteamos axiomas para cada operación utilizando las 2 constructoras primitivas
NULA: → CADENA
AGREGAR: CADENA X CHAR → CADENA

SEMÁNTICA ∀ C, T ∈ CADENA, ∀ d ∈ CHAR

CONCAT: CADENA X CADENA → CADENA

CONCAT (C , NULA) ≡ C

CONCAT(C , AGREGAR(T,d)) ≡ AGREGAR(CONCAT(C,T),d)

ESPECIFICACIÓN TAD CADENA

¿Cuál es el resultado de concatenar la cadena "AB"CON LA CADENA "XYZ" ?

CONCAT (C, NULA) ≡ C

CONCAT (**C**, AGREGAR(**T**, d)) ≡ AGREGAR(CONCAT(**C**,**T**), d)

AGREGAR(AGREGAR(NULA, A), B) ← **CADENA C**

AGREGAR(AGREGAR(AGREGAR(NULA, X),Y) , Z)

CONCAT(**C** , AGREGAR(AGREGAR(AGREGAR(NULA, X),Y) , Z) **)**

ESPECIFICACIÓN TAD CADENA

¿Cuál es el resultado de concatenar la cadena "AB"CON LA CADENA "XYZ" ?

CONCAT (C, NULA) ≡ C

CONCAT (**C**, AGREGAR(**T**, d)) ≡ AGREGAR(CONCAT(**C**,**T**), d)

CONCAT(**C** , AGREGAR(AGREGAR(AGREGAR(NULA, X),Y), Z) **)**

AGREGAR(**CONCAT(** **C**,AGREGAR(AGREGAR(NULA,X),Y),Z)**)**

AGREGAR(AGREGAR(**CONCAT(** **C** , AGREGAR(NULA, X)**)** ,Y) ,Z)

AGREGAR(AGREGAR(AGREGAR(**CONCAT(** **C** , NULA**)** ,X),Y),Z)

AGREGAR(AGREGAR(AGREGAR(**C** ,X),Y),Z)

ESPECIFICACIÓN TAD CADENA

Planteamos axiomas para cada operación utilizando las 2 constructoras primitivas
NULA: → CADENA
AGREGAR: CADENA X CHAR → CADENA

Dado el TAD CADENA, agregue a la especificación la operación BORRAR que, dada una cadena, elimine el último carácter agregado a la misma

SINTAXIS BORRAR: CADENA → CADENA

SEMÁNTICA ∀ C ∈ CADENA , ∀ d ∈ CHAR

BORRAR(NULA) ≡ NULA

BORRAR(AGREGAR(C, d)) ≡ C

ESPECIFICACIÓN TAD CADENA

Dado el TAD CADENA, agregue a la especificación la operación PRIMERO, que dada una cadena devuelva el 1° carácter agregado a la misma

AGREGAR(AGREGAR(AGREGAR(AGREGAR(NULA, H), O), L) , A)

CADENA VACIA

ESPECIFICACIÓN TAD CADENA

Planteamos axiomas para cada operación utilizando las 2 constructoras primitivas
NULA: → CADENA
AGREGAR: CADENA X CHAR → CADENA

Dado el TAD CADENA, agregue a la especificación la operación PRIMERO, que dada una cadena devuelva el 1°carácter agregado a la misma

SINTAXIS PRIMERO: CADENA → CHAR U {indefinido}

SEMÁNTICA ∀ C ∈ CADENA , ∀ d ∈ CHAR

PRIMERO(NULA) ≡ indefinido

PRIMERO(AGREGAR(C, d)) ≡ SI ESNULA(C) ENTONCES
d
SINO
PRIMERO(C)

COMO USUARIO TAD CADENA

TAD CADENA

Como usuario del TAD CADENA diseñe una función que cuente la cantidad de veces que aparece un determinado caracter en la cadena

NULA, AGREGAR, LARGO, ESNULA, CONCAT, PRIMERO, BORRAR, ULTIMO, AGREGARP

```
FUNCION contarCar(C, x): CADENA x CHAR → entero ≥ 0
  SI ESNULA( C ) ENTONCES
        RETORNA 0
  SINO
        SI ULTIMO( C ) = x ENTONCES
              RETORNA 1 + contarCar( BORRAR(C) , x)
        SINO
              RETORNA contarCar( BORRAR(C) , x)
```

TAD COMISION

ESPECIFICACIÓN
TAD COMISION

SINTAXIS

COMISIONVACIA: → COMISION

Constructoras primitivas

ALTAALUMNO: COMISION x ALUMNO → COMISION

CANTIDAD: COMISION → ENTERO >= 0

SEMÁNTICA ∀ C ∈ COMISION, ∀ x ∈ ALUMNO

CANTIDAD(COMISIONVACIA) ≡ 0

CANTIDAD(ALTAALUMNO(C, x)) ≡ 1 + CANTIDAD(C)

TAD CUENTA

ESPECIFICACIÓN
TAD CUENTA

Sintaxis:

CUENTAVACIA :		→ CUENTA
DEPOSITAR:	CUENTA x DINERO	→ CUENTA
ESTAVACIA:	CUENTA	→ BOOL
SALDO:	CUENTA	→ DINERO
EXTRAER:	CUENTA x DINERO	→ CUENTA
TIENESALDO:	CUENTA	→ BOOL
ULTIMA:	CUENTA	→ DINERO U {indefinido}
DESHACER:	CUENTA	→ CUENTA
CONTIENE:	CUENTA x DINERO	→ BOOL
CANTIDAD:	CUENTA	→ ENTERO≥0

ESPECIFICACIÓN
TAD CUENTA

Semántica: para todo C ∈ CUENTA, para todo u, v ∈ DINERO.

ESTAVACIA (CUENTAVACIA) ≡ TRUE
ESTAVACIA (DEPOSITAR(C,u)) ≡ FALSE

SALDO(CUENTAVACIA) ≡ 0
SALDO(DEPOSITAR(C,u)) ≡ u + SALDO(C)

CANTIDAD (CUENTAVACIA) ≡ 0
CANTIDAD (DEPOSITAR(C,u)) ≡ 1 + CANTIDAD (C)

TIENESALDO (CUENTAVACIA) ≡ FALSE
TIENESALDO (DEPOSITAR(C,u)) ≡ SALDO(DEPOSITAR(C,u)) > 0

ESPECIFICACIÓN
TAD CUENTA

Semántica: para todo C ∈ CUENTA, para todo u, v ∈ DINERO.

EXTRAER (CUENTAVACIA, v) ≡ CUENTAVACIA
EXTRAER (DEPOSITAR(C,u), v) ≡ SI SALDO(DEPOSITAR(C,u)) ≥ v ENTONCES
DEPOSITAR(DEPOSITAR(C,u), -v)
SINO
DEPOSITAR(C,u)

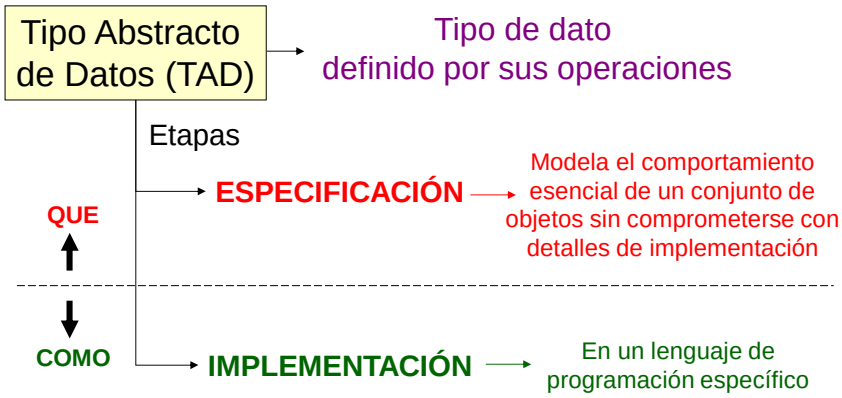
ULTIMA (CUENTAVACIA) ≡ indefinido
ULTIMA (DEPOSITAR(C,u)) ≡ u

DESHACER (CUENTAVACIA) ≡ CUENTAVACIA
DESHACER (DEPOSITAR(C,u)) ≡ C

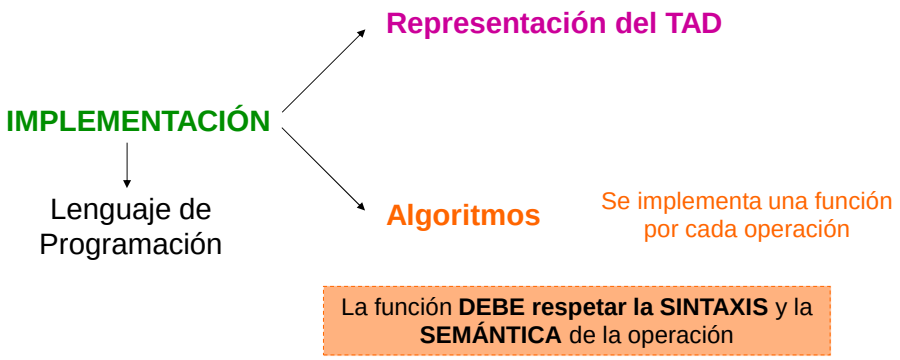
CONTIENE (CUENTAVACIA, v) ≡ FALSE
CONTIENE (DEPOSITAR(C, u),v) ≡ u=v OR CONTIENE (C,v)

TAD CUENTA
IMPLEMENTACIÓN

Tipos Abstractos de Datos



Tipos Abstractos de Datos



IMPLEMENTACIÓN TAD CUENTA

CON LISTA ENLAZADA
DINERO DE TIPO REAL>0

Lenguaje C++

REPRESENTACIÓN DEL TAD

```
typedef float Dinero;

struct Nodo{
            Dinero dato;
            Nodo* siguiente;
};
```

```
struct Cuenta{
              Nodo* depositos;
              ...
              ...
};
```

TAD CUENTA(DINERO)

Sintaxis:

CUENTAVACIA: → CUENTA
DEPOSITAR: CUENTA x DINERO → CUENTA
ESTAVACIA: CUENTA → BOOL
SALDO: CUENTA → DINERO
EXTRAER: CUENTA x DINERO → CUENTA
TIENESALDO: CUENTA → BOOL
ULTIMA: CUENTA → DINERO U {indefinido}
DESHACER: CUENTA → CUENTA
CONTIENE: CUENTA x DINERO → BOOL
CANTIDAD: CUENTA → ENTERO≥0

Implementar el ADT CUENTA de manera que TODAS las operaciones (salvo la operación CONTIENE) sean de costo O(1)

IMPLEMENTACIÓN TAD CUENTA

CON LISTA ENLAZADA
DINERO DE TIPO REAL>0

Lenguaje C++

REPRESENTACIÓN DEL TAD

```
typedef float Dinero;
struct Nodo{
            Dinero dato;
            Nodo* siguiente;
};
```

```
struct Cuenta{
              Nodo* depositos;
              ...
              ...
};
```

OPERACIONES/ALGORITMOS

CUENTAVACIA: → CUENTA

```
Cuenta cuentaVacia(){
      Cuenta nueva;
      nueva.depositos = NULL;
      ...
      ...
      return nueva;
};
```

Instrucciones adicionales para que las operaciones Saldo y Cantidad sean de costo O(1)

IMPLEMENTACIÓN TAD CUENTA

CON LISTA ENLAZADA
DINERO DE TIPO REAL>0

Lenguaje C++

REPRESENTACIÓN DEL TAD

```
typedef float Dinero;
struct Nodo{
                Dinero dato;
                Nodo* siguiente;
};
```

```
struct Cuenta{
                Nodo* depositos;
                ...
                ...
};
```

OPERACIONES/ALGORITMOS

DEPOSITAR: CUENTA x DINERO → CUENTA

```
Cuenta depositar(Cuenta C, Dinero X){
        Nodo *nuevo = new Nodo;
        nuevo->dato = X;
        nuevo->siguiente = C.depositos;
        C.depositos = nuevo;
        . . .
        . . .
        return C;
};
```

Instrucciones adicionales para que las operaciones Saldo y Cantidad sean de costo O(1)

Preguntas…
…y a practicar…