INSTITUTO POLITECNICO NACIONAL

Escuela Superior de Computo

ACTIVIDAD AUTONOMA

“Tipos Abstractos de Datos y su Representación”

ALUMNO: González Barrientos Geovanni Daniel

BOLETA: 2020630148

CARRERA: Ingeniería en Sistemas Computacionales

GRUPO: 1CV3

PROF: Flores Mendoza Yaxkin

MATERIA: Estructuras De Datos

Fecha: 27 de Octubre del 2020

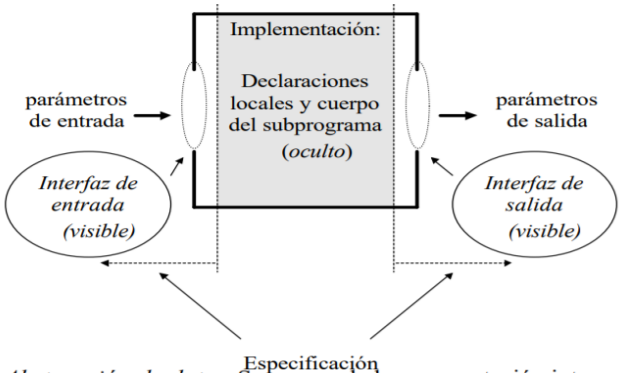
DEFINICION

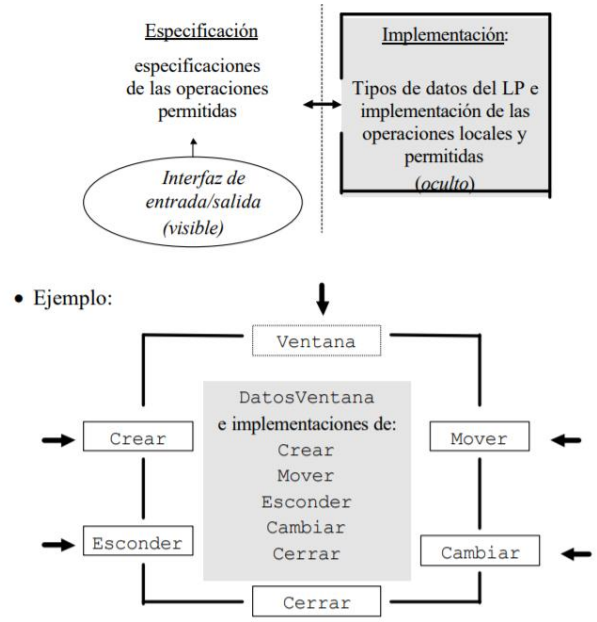
Un Tipo de Dato Abstracto (TDA) es un modelo que define valores y las operaciones que se pueden realizan sobre ellos. Y se denomina abstracto ya que la intención es que quien lo utiliza, no necesita conocer los detalles de la representación interna o bien el cómo están implementadas las operaciones.

Es por esto una práctica que nos provee un grado de abstracción que permite desacoplar al código que usa un TDA de aquel código que lo implementa.

Un ejemplo de un tipo de dato aplicado es un programa que saca el área de un rectángulo de las dimensiones que un usuario decida. Sería muy tedioso definir la multiplicación de 'base' por 'altura' varias veces en el programa, además que limitaría al usuario a sacar un número determinado de áreas. Por ello, se puede crear una función denominada 'Área', la cual va a ser llamada, con los parámetros correspondientes, el número de veces que sean necesitadas por el usuario y así evitar realizar mucho trabajo, por lo que el programa resulta más rápido, más eficiente y de menor longitud. En el método principal solamente se llama a la función Área y el programa hace el resto.

- Abstracción o encapsulamiento:
  - Separación de la especificación de un objeto o algoritmo de su implementación, en base a que su utilización dentro de un programa sólo debe depender de un interfaz explícitamente definido (la especificación) y no de los detalles de su representación física (la implementación), los cuales están ocultos.
- Ventajas de la abstracción:
  - Establece la independencia de QUÉ es el objeto (o QUÉ hace el algoritmo) y de CÓMO está implementado el objeto (o algoritmo), permitiendo la modificación del CÓMO sin afectar al QUÉ, y por lo tanto, sin afectar a los programas que utilizan este objeto o algoritmo.
- Tipos de abstracciones:
  - Abstracción de operaciones. Una serie de operaciones básicas se encapsulan para realizar una operación más compleja. En los lenguajes de programación este tipo de abstracción se logra mediante los subprogramas:





Existen cuatro tipos abstractos de datos (o TADs) muy utilizados en la programación:

a) Listas

b) Colas

c) Pilas

d) Arboles

Cada TAD consta de su especificación o definición, que será independiente de su implementación (un TAD puede implementarse de diversas formas siempre que cumplan su definición). Además, la definición nos proporcionará las operaciones, las cuales, a su vez, nos determinarán la utilización del TAD.

1) TIPO LISTA

Colección de elementos homogéneos (del mismo tipo: TipoElemento) con una relación LINEAL establecida entre ellos. Pueden estar ordenadas o no con respecto a algún valor de los elementos y se puede acceder a cualquier elemento de la lista.

OPERACIONES:

TipoLista Crear()

void Imprimir(TipoLista lista)

int ListaVacía(TipoLista lista)

void Insertar(TipoLista lista, TipoElemento elem)

void Eliminar(TipoLista lista, TipoElemento elem)

Opcionalmente, si la implementación lo requiere, puede definirse:

int ListaLlena(TipoLista lista)

Hay que tener en cuenta que la posición de la inserción no se especifica, por lo que dependerá de la implementación. No obstante, cuando la posición de inserción es la misma que la de eliminación, tenemos una subclase de lista denominada pila, y cuando insertamos siempre por un extremo de la lista y eliminamos por el otro tenemos otra subclase de lista denominada cola.

2) TIPO COLA

Colección de elementos homogéneos (del mismo tipo: TipoElemento) ordenados cronológicamente (por orden de inserción) y en el que sólo se pueden añadir elementos por un extremo (final) y sacarlos sólo por el otro (frente). Es una estructura FIFO (First In First Out)

OPERACIONES

```
TipoCola Crear();
/* Crea y devuelve una cola vacía */
int ColaVacía(Tipocola Cola);
/* Devuelve 1 sólo si “cola” está vacía */
int ColaLlena(TipoCola Cola);
/* Devuelve 1 sólo si “cola” está llena */
int Sacar(Tipocola Cola, TipoElemento *elem);
/* Saca el siguiente elemento del frente y lo pone en “elem” */
int Meter(TipoCola Cola, TipoElemento elem);
/* Mete “elem” al final de la cola */
```

Hay que tener cuidado con las precondiciones de las operaciones Meter y Sacar, debido a:

a) No se puede meter más elementos en una cola llena.

b) No se puede sacar elementos de una cola vacía.

Pueden tenerse en cuenta ANTES de cada llamada a Meter y Sacar, o en los propios procedimientos Meter y Sacar.

3) TIPO PILA

Colección de elementos homogéneos (del mismo tipo: TipoElemento) ordenados cronológicamente (por orden de inserción) y en el que sólo se pueden añadir y extraer elementos por el mismo extremo, la cabeza. Es una estructura LIFO (Last In First Out)

OPERACIONES:

```
TipoPila Crear(void); /* vacía */
/* Crea una pila vacía */
int PilaVacia(TipoPila pila);
/* Comprueba si la pila está vacía */
int PilaLlena(TipoPila pila);
/* Comprueba si la pila está llena */
void Sacar(TipoPila *pila, TipoElemento *elem);
/* Saca un elemento. No comprueba antes si la pila está vacía*/
void Meter(TipoPila pila, TipoElemento elem);
/* Mete un elemento en la pila. No comprueba si está llena. */
```

4) TIPO ARBOL BINARIO

Estructura de elementos homogéneos (del mismo tipo: TipoElemento) con una relación JERARQUICA establecida entre ellos a modo de árbol binario.

OPERACIONES:

```
CrearVacío(): TipoABin
(* Crea un árbol vacío *)
CrearRaíz(elem: TipoElemento): TipoABin
(* Crea un árbol de un único elemento *)
ArbolVacío(árbol: TipoABin): B
(* Comprueba si “árbol” está vacío *)
AsignarIzq(VAR árbol: TipoABin; izq: TipoABin)
```

(* Establece “izq” como subárbol izq. de “árbol” *)

AsignarDer(VAR árbol: TipoABin; der: TipoABin)

(* Establece “izq” como subárbol izq. de “árbol” *)

Info(árbol: TipoABin): TipoElemento

(* Devuelve el nodo raíz de “árbol” *)

Izq(árbol: TipoABin): TipoABin

(* Devuelve el subárbol izquierdo de “árbol” *)

Der(árbol: TipoABin): TipoABin

(* Devuelve el subárbol derecho de “árbol” *)

BorrarRaíz(VAR árbol, izq, der: TipoABin)

(* Devuelve en “izq” y “der” los subárboles de “árbol” *)

Imprimir(árbol: TipoABin)

(* Imprime en pantalla el contenido completo de “árbol” *)

REPRESENTACION FORMAL E INFORMAL

Hay dos formas de realizar la especificación de un TAD: informal y formal.

FORMAL

Descripción: Esta se basa en una especificación algebraica y se compone de la siguiente forma:

- TAD Nombre del tipo de datos (VALORES: valores que toman los datos del tipo; OPERACIONES: nombre de las operaciones que los manipulan)
- Sintaxis: {Forma de las operaciones}
- Nombre de operación (tipo de argumento) → tipo de resultado
- Semántica: {Significado de las operaciones. - Expresión que refleja, para unos determinados argumentos, el resultado que se puede obtener}
- Nombre de operación (valores particulares) ⇒Expresión resultado

En la especificación formal intervienen tres componentes:

1. Un conjunto de objetos.
2. Un conjunto de descripciones sintácticas de operaciones: declaración de encabezamientos de operaciones con sus tipos de argumentos de entrada y de salida.

3. Una descripción semántica, esto es, un conjunto suficientemente completo de relaciones que especifiquen el funcionamiento de las operaciones para describir su comportamiento sin ningún tipo de ambigüedad.

Ejemplo:

espec secuenciaDeNaturales

usa naturales_3

género secN

operaciones

[ ] : % secN {secuencia vacía}

[_] : natural % secN {secuencia unitaria}

_ ++ _ : secN secN % secN {concatenar secuencias}

_ ⊕ _ : natural secN % secN

{añadir un natural a una secuencia}

ecuaciones x: natural; s, s1, s2, s3: secN

s ++ [ ] = s

[ ] ++ s = s

(s1 ++ s2) ++ s3 = s1 ++ (s2 ++ s3)

x ⊕ s = [x] ++ s

fespec

INFORMAL

Descripcion: Este tipo de representación se basa mas en el lenguaje natural, contando con la siguiente estructura:

- TAD Nombre del tipo de datos (VALORES: valores que toman los datos del tipo; OPERACIONES: nombre de las operaciones que los manipulan)
- Nombre de operación (tipo de argumento) → tipo de resultado
- Efecto: Descripción de la operación
- Excepción: Posibles excepciones

En la especificación informal intervienen cuatro componentes:

1. Un conjunto de objetos.
2. La declaración de los encabezamientos de las operaciones.
3. La descripción de cada operación bajo el epígrafe de efectos.
4. Las excepciones que pueden producirse en cada operación (si se producen).

Ejemplo:

constructor Crear (n: cadena; e, h: byte);

{

Parámetros:

n: cadena de caracteres donde se almacena el nombre de la persona

e: número natural que indica la edad de la persona

h: número natural que indica el número de hijos que tiene la persona

Devuelve:

Precondiciones:

0 < e ≤ 110

0 ≤ h ≤ 15

Efecto:

Crea una ficha con los valores indicados en los argumentos

Excepciones:

Si (e < 16 y h > 0) no se crea la ficha

}

BIBLIOGRAFIA

- Es.wikipedia.org. 2020. Tipo De Dato Abstracto. [online] Available at: <https://es.wikipedia.org/wiki/Tipo_de_dato_abstracto> [Accessed 28 October 2020].
- [2]"Estructuras de Datos II", Informatica.utem.cl, 2020. [Online]. Available: http://informatica.utem.cl/~mcast/ESDATOS/TADS/Ttema1_0506.pdf. [Accessed: 28- Oct- 2020].
- [3]"TIPOS ABSTRACTOS DE DATOS", Infor.uva.es, 2020. [Online]. Available: https://www.infor.uva.es/~mserrano/EDI/cap2.pdf. [Accessed: 28- Oct- 2020].