Ontiveros Lara Claudia Sarahi. 17212167

Ejemplos de tipos de datos abstractos

TipoLista
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, 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.
· En el procedimiento Eliminar el argumento elem podría ser una clave, un campo de un registro TipoElemento.

TipoCola
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 */

¡Ojo con las precondiciones de las operaciones Meter y Sacar!:
a) No se puede meter más elementos en una cola llena.
b) No se puede sacar elementos de un cola vacía.
Pueden tenerse en cuenta ANTES de cada llamada a Meter y Sacar, o en los propios procedimientos Meter y Sacar.

TipoPila
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. */
```

TipoAbin

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” *)
```

· Obsérvese que no se define una operación de inserción de elementos ya que el punto de inserción puede variar de una aplicación a otra, razón por la cual se deja al usuario del TAD que defina su propio Insertar().
· En Imprimir() habría que definir de alguna manera el orden de impresión.

TipoABB

operaciones

Las ya definidas para árboles binarios (cambiando TipoABin por TipoABB) y las siguientes propias:

```
Buscar(abb: TipoABB; clave: TipoClave;
VAR elem: TipoElemento; VAR está: B)
(* Pone en “elem” el nodo “clave”. “está” indica si está
“clave”*)
```

Insertar(VAR abb: TipoABB; elem: TipoElemento)
(* Insertar en “abb” el nodo “elem” *)
Eliminar(VAR abb: TipoABB; clave: TipoClave)
(* Elimina el nodo “clave” de “abb” *)
Imprimir(abb: TipoABB; rec: TipoRecorrido)
(* Saca en pantalla “abb” completo, recorriéndolo según “rec”*)

· Normalmente, aunque depende de cómo se hayan realizado las inserciones, el acceso a los nodos es más eficiente que en las listas, ya que sólo se recorre desde la raíz la rama que lo contiene en base a la ordenación del árbol.
· Supondremos que no existen nodos con claves duplicadas.
· rec indica la forma de recorrer el árbol según se procese la raíz antes (PREORDEN), después (POSTORDEN) o entre (ENORDEN) el procesamiento de sus dos subárboles.

Referencias:

Joyanes Aguilar, L., & Zahonero Martínez, I. (1998a). Estructura de datos. Algoritmos, abstracción y objetos. (Ed. rev.). Madrid, España: McGraw-Hill.
Martínez, I., & Aguilar, L. (2011). Estructuras de datos en Java. España: McGraw-Hill España.