Práctica 5. TAD conjunto

Objetivos.

Se trata de construir el TAD CONJUNTO con una implementación no acotada con cabecera y una biblioteca que incluya operaciones básicas sobre conjuntos.

Se introduce el concepto de *iterador activo*, y se compara éste con el iterador pasivo estudiado en la práctica 2 (el TAD PILA).

Enunciado.

Construir el TAD CONJUNTO según el siguiente módulo de definición:

```
DEFINITION MODULE Conjunto;
        FROM TadItem IMPORT ITEM;
        TYPE
                CONJUNTO;
                TIPO_ERROR=  << a definir>>;

        PROCEDURE Error (): TIPO_ERROR;

        PROCEDURE Crear (): CONJUNTO;
        PROCEDURE Destruir (VAR c: CONJUNTO);

        PROCEDURE EsVacio (c: CONJUNTO): BOOLEAN;
        PROCEDURE Esta(c: CONJUNTO; x: ITEM): BOOLEAN;
        PROCEDURE Cardinal (c: CONJUNTO): CARDINAL;

        PROCEDURE Incluir (VAR c: CONJUNTO; x: ITEM);

        (* Pre: Esta(c, x) *)
        PROCEDURE Excluir (VAR c: CONJUNTO; x: ITEM);

        PROCEDURE Copiar (VAR c1: CONJUNTO; c2: CONJUNTO);

        PROCEDURE Inicializar(c: CONJUNTO);
        PROCEDURE Elemento (c: CONJUNTO):ITEM;
END Conjunto.
```

donde el papel de los procedimientos Inicializar y Elemento se explicará más adelante. El TAD CONJUNTO se representará mediante la siguiente estructura:

```
TYPE    PNODO = POINTER TO NODO;

        NODO = RECORD
                        Dat : ITEM;
                        Sig : PNODO;
                END;

        CONJUNTO = POINTER TO CABECERA;

        CABECERA =  RECORD
                                Cont : PNODO;
                                Card: CARDINAL;
                                Iterador : PNODO;
                        END;
```

Implementar una biblioteca para el TAD CONJUNTO con las siguientes operaciones:

```
DEFINITION MODULE BibConj;
        FROM Conjunto IMPORT CONJUNTO;

        PROCEDURE Union(VAR c: CONJUNTO; c1,c2: CONJUNTO);

        PROCEDURE Inter(VAR c: CONJUNTO; c1,c2: CONJUNTO);

        PROCEDURE Difer(VAR c: CONJUNTO; c1,c2: CONJUNTO);

        PROCEDURE DifSim(VAR c: CONJUNTO; c1,c2: CONJUNTO);

        PROCEDURE EsIgual (c1, c2: CONJUNTO: BOOLEAN;
END BibConj.
```

donde se supone que el conjunto c está creado, está vacío y es distinto de los conjuntos c1 y c2.

Finalmente construir un programa que, importando el TAD CONJUNTO y su biblioteca, compruebe el correcto funcionamiento de ambos.

Iteradores pasivos vs. Iteradores activos.

En la práctica 2 (el TAD PILA) se introdujo el concepto de iterador *pasivo*. Recordemos que un iterador pasivo visita *todos* los elementos de un TAD y aplica la *misma* operación a todos ellos, sin modificar por ello el TAD visitado. Para implementar los iteradores pasivos en Modula-2, definimos un tipo procedimiento:

```
TYPE TIPO_OPERACION = PROCEDURE(ITEM);
```

y un procedimiento genérico (un procedimiento que recibe otro procedimiento como parámetro):

```
PROCEDURE Aplicar(P: PILA; op: TIPO_OPERACION);
```

Si bien los iteradores pasivos son muy útiles, a veces surgen situaciones en las que su estructura monolítica (aplicar la *misma* operación a *todos* los elementos del TAD) resulta demasiado restrictiva. Considérense, por ejemplo, situaciones en las que no deseemos visitar *todos* los nodos del TAD, o bien situaciones en las que no deseemos aplicar la *misma* operación a todos los elementos del TAD.

Otra complicación diferente surge cuando queremos computar un resultado que depende del valor de todos los nodos del TAD. Supongamos, por ejemplo, que empleamos un iterador pasivo para calcular la suma de todos los nodos de una Pila. La operación que debemos pasar al iterador Aplicar como parámetro debe acumular el valor de todos los nodos visitados, por lo que tendrá el siguiente aspecto:

```
PROCEDURE SumaNodo(X:ITEM);
BEGIN
        Total:= Total + X;
END SumaNodo;
```

donde Total es una variable global que empleamos para acumular la suma (la variable Total no puede ser local a SumaNodo, pues de lo contrario no conservaría su valor entre llamada y llamada). Además, antes de invocar al iterador Aplicar nos hará falta inicializar la variable Total a 0. Podemos asegurarnos de que tal inicialización se efectúa siempre incluyéndola en un procedimiento SumarPila, que inicialice al acumulador Total e invoque seguidamente al iterador Aplicar, obteniendo el siguiente código:

```
VAR Total : ITEM;

PROCEDURE SumaNodo(X:ITEM);
BEGIN
        Total:= Total + X;
END SumaNodo;

PROCEDURE SumarPila(P: PILA): ITEM;
BEGIN
        Total := 0;
        Aplicar(P,SumaNodo);
        RETURN Total;
END SumarPila;
```

Como vemos, es imprescindible emplear un objeto global (Total) para que las diferentes invocaciones a SumaNodo se comuniquen entre sí y acumulen la suma de los nodos de la pila. Claramente, el iterador pasivo no está pensado para resolver este tipo de problemas.

Una solución alternativa consiste en emplear un un iterador *activo*, que ofrece un esquema de iteración mucho más flexible y potente. Mientras que los iteradores pasivos recorren completamente el TAD sin intervención alguna por parte del programador que los emplea, los iteradores activos recorren los elementos del TAD uno a uno, devolviendo en cada llamada un elemento *no visitado* del TAD. Esto significa que el iterador activo debe recordar entre llamada y llamada por dónde se quedó el recorrido, para devolver el *siguiente* elemento no visitado. Es decir, un iterador activo tiene asociado cierto estado que le permite saber por dónde va el recorrido actualmente. Para implementar este recorrido *secuencial*, los iteradores activos ofrecen dos operaciones: Inicializar y Elemento.

- Inicializar – Inicializa el estado del iterador activo
- Elemento - devuelve el elemento referido por el estado actual y actualiza el estado del iterador para que se refiera al siguiente elemento. Si el estado actual no refiere a ningún elemento (es decir, el recorrido había sido completado con anterioridad), el iterador activo debe señalar un error de IteracionCompleta, y devolver ValorPorDefecto().

Para el TAD PILA, el iterador activo se implementará a través de los procedimientos:

```
PROCEDURE Inicializar(P:PILA);
PROCEDURE Elemento(P:PILA): ITEM;
```

El problema de sumar los nodos de una pila puede entonces resolverse aplicando el iterador activo de la siguiente manera:

```
PROCEDURE SumarPila(P:PILA): ITEM;
VAR Total: ITEM;
BEGIN
        Total:= 0;
        Inicializar(P);
        IF Error() = SinError THEN
                FOR I:= 1 to Profundidad(P) DO
                        Total:= Total + Elemento(P);
                END;
        END;
        RETURN Total;
END SumarPila;
```

Para el TAD CONJUNTO, estos procedimientos tienen las siguientes definiciones:

```
Inicializar(C: CONJUNTO);
Elemento(C: CONJUNTO): ITEM;
```

El estado del iterador se incluye en la propia estructura del TAD CONJUNTO, y viene dado por el campo Iterador de la estructura CABECERA.

Práctica Suplementaria

Se trata de implementar la simulación del funcionamiento de un centro de distribución.

El centro se compone de un almacén donde se guardan los artículos y un número de andenes de carga donde los camiones esperan los artículos que componen el pedido que hay que enviar en ese camión. Para servir los pedidos, los artículos son colocados desde el almacén en una cinta transportadora que los lleva hasta el andén de carga. Cuando un artículo sale de la cinta, se mira a qué pedido pertenece. Si no pertenece a ninguno, ha sido una equivocación y se devuelve al almacén. Cuando un pedido ha sido completado, el camión puede partir y deja libre el andén.

Los pedidos se representarán mediante conjuntos con los números de serie de los artículos. La cinta transportadora será una cola donde se insertan los elementos desde el almacén y se extraen en los andenes de carga. Se supondrá un número de andenes fijo, que pueden estar ocupados o no en un momento dado.