Práctica 3. TAD cola de prioridad

Objetivos.

Se trata de construir el TAD cola de prioridad con una implementación no acotada con cabecera.

Se introduce una forma de simular *genericidad* en los TADs de Modula-2 mediante el uso del TAD ITEM.

Enunciado.

Construir el TAD COLAP según el siguiente módulo de definición:

```
DEFINITION MODULE ColaP;
        FROM TadItem IMPORT ITEM;
        TYPE    COLAP;
                TIPO_ERROR= <<a definir>>;

        PROCEDURE Error():  TIPO_ERROR ;

        PROCEDURE Crear(): COLAP;

        PROCEDURE Longitud(cp: COLAP): CARDINAL;

        PROCEDURE Insertar(VAR cp: COLAP; x: ITEM);

        (* Pre: Longitud(cp)<>0 *)
        PROCEDURE Extraer(VAR cp: COLAP );

        (* Pre: Longitud(cp)<>0 *)
        PROCEDURE Frente(cp: COLAP): ITEM;

        PROCEDURE Copiar(VAR cp1:COLAP; cp2:COLAP);

        PROCEDURE Destruir(VAR cp: COLAP);
END ColaP.
```

donde el papel del tipo ITEM importado del módulo TadItem se explicará más adelante. El TAD COLAP se representará internamente mediante la siguiente estructura:

```
TYPE
        COLAP =  POINTER TO CABECERA;
        CABECERA=   RECORD
                        primero: POINTER TO NODO;
                        longitud: CARDINAL;
                    END;
        NODO  =     RECORD
                        cont: ITEM;
                        sig: POINTER TO NODO;
                    END;
```

Detalles de implementación.

En una cola de prioridad los elementos están ordenados dependiendo de su prioridad, de manera que esté disponible (para las operaciones Frente y Extraer) el elemento de máxima prioridad. En caso de igualdad se sigue la regla FIFO, de dos elementos con igual prioridad sale primero el que primero entró. Esto se puede conseguir bien insertando ordenadamente y extrayendo el primer elemento, bien

insertando por el final y recorriendo la cola para encontrar el máximo a la hora de extraer o de consultar el frente. El alumno puede elegir cualquiera de estas dos políticas.

El procedimiento Copiar debe efectuar la operación de Crear sobre el parámetro en el que se hace la copia (cp1) de forma que se pierde su contenido anterior. Es responsabilidad del usuario liberar adecuadamente los recursos de la cola destino (cp1) antes de efectuar la copia.

Genericidad y el módulo TadItem.

El TAD PILA presentado en la práctica anterior era genérico en el sentido de que podía emplearse para cualquier tipo base sin modificar el código de la implementación. Si el usuario deseaba emplear pilas de un tipo base diferente, bastaba con modificar la definición del tipo ITEM incluida en el módulo de definición. Sin embargo, esta solución debe considerarse insatisfactoria pues:

1. el usuario necesita modificar el módulo de definición del TAD
2. se introducen detalles sobre el tipo base en el módulo de definición del TAD

Además de los defectos anteriormente citados, en este TAD COLAP surge la necesidad de disponer de un mecanismo *más potente* para conseguir que el TAD sea **genérico**; es decir, funcione para cualquier tipo ITEM **sin necesidad de modificar la implementación**. Para entender este problema, considérese la implementación del procedimiento Insertar, en el que es necesario comparar las prioridades de los elementos para determinar la posición de la cola que le corresponde al ITEM x. Si el tipo ITEM es CARDINAL, el procedimiento Insertar deberá incluir comparaciones del tipo <, =, o >:

```
PROCEDURE Insertar(VAR cp: COLAP; x: ITEM);
BEGIN
        ...
        IF x < c^.cont THEN  (* comparación de cardinales *)
        ...
END Insertar;
```

Sin embargo, si el tipo ITEM es una cadena de caracteres, la comparación deberá efectuarse a través del procedimiento Compare de la biblioteca Str:

```
PROCEDURE Insertar(VAR cp: COLAP; x: ITEM);
BEGIN
        ...
        IF Str.Compare(x,c^.cont)=-1 THEN  (* comparación de cadenas *)
        ...
END Insertar;
```

Se plantea entonces la siguiente pregunta: ¿cómo podemos codificar el procedimiento Insertar de forma que no sea necesario modificarlo aunque se altere la definición del tipo ITEM?

La solución que adoptaremos a lo largo del curso consiste en tratar al tipo base de nuestros TADs como si fuera otro TAD; es decir, se trata de construir un TAD ITEM para el cual se definan las operaciones necesarias para soportar la genericidad (pero sólo éstas, de forma que no puede hablarse de un auténtico TAD). En el caso del TAD COLAP, es necesario emplear como tipo base un TAD ITEM que ofrezca una operación de comparación, a la que denominaremos Comparar:

```
DEFINITION MODULE TadItem;
        TYPE    ITEM = <<a definir>>;
        PROCEDURE Comparar(i1,i2: ITEM): INTEGER;
END TadItem.
```

Como puede apreciarse el TAD ITEM no es un auténtico TAD, pues el tipo ITEM es un tipo transparente definido en su totalidad en el propio módulo de definición. De esta manera, simplificaremos la implementación prescindiendo de operaciones tales como Crear y Destruir sobre el tipo ITEM. Sin embargo, el TAD *genérico* (en este caso COLAP) importará el tipo ITEM y lo manipulará **como si fuera un TAD**; es decir, a través de las operaciones exportadas en el módulo de definición TadItem, sin acceder **jamás** a su estructura de datos de manera directa aunque ésta sea visible. El procedimiento Insertar puede codificarse ahora de la siguiente manera:

```
PROCEDURE Insertar(VAR cp: COLAP; x: ITEM);
BEGIN
        ...
        IF Comparar(x,c^.cont)=-1 THEN  (* comparación de ITEM *)
        ...
END Insertar;
```

Finalmente y para completar la práctica, construir un programa de prueba para el TAD COLAP.

Práctica Suplementaria.

Construir el TAD POLINOMIO según el siguiente módulo de definición:

```
DEFINITION MODULE Polinom;

        TYPE    POLINOMIO;
                TIPO_ERROR= <<a definir>>;

        PROCEDURE Error():  TIPO_ERROR ;

        PROCEDURE Crear():POLINOMIO;
        (* Devuelve el polinomio nulo *)

        PROCEDURE Destruir(VAR p:POLINOMIO);

        PROCEDURE Anyadir(VAR p:POLINOMIO; i:CARDINAL; c:INTEGER);
        (* añade, reemplaza o elimina el término de grado i *)

        (* Pre: 0 <= i <= Grado(p) *)
        PROCEDURE Coeficiente(p:POLINOMIO; i:CARDINAL): INTEGER;

        PROCEDURE Grado(p:POLINOMIO):CARDINAL;
        PROCEDURE Igual(p1,p2:POLINOMIO):BOOLEAN;
        PROCEDURE Copiar(VAR p1:POLINOMIO; p2:POLINOMIO);

END Polinom.
```

Detalles de implementación.

Sólo se incluirán en el polinomio los términos cuyo coeficiente sea distinto de cero. Al aplicar Anyadir, si el polinomio p no tiene término de grado i entonces se añadirá ese nuevo término. Por el contrario, si p contiene un término de grado i, entonces se

reemplazará el coeficiente de dicho término. Si el nuevo coeficiente c es cero, se eliminará el término correspondiente al grado i.

Representaremos los polinomios mediante colas de prioridad:

```
IMPLEMENTATION MODULE Polinom;
FROM ColaP IMPORT .....;

    TYPE   POLINOMIO = COLAP;
    ...
    ...

END Polinom;
```

Cada término del polinomio será un elemento de la cola, cuya prioridad vendrá dada por su grado. Las operaciones sobre polinomios se definirán entonces como operaciones sobre el TAD ColaP. Para emplear esta solución será necesario definir un TAD TadItem adecuado.

Haciendo uso de este TAD crear una biblioteca con las siguientes operaciones:

```
DEFINITION MODULE BibPol;
FROM Polinom IMPORT ....;

    PROCEDURE Suma(VAR a:POLINOMIO; b,c:POLINOMIO);
    PROCEDURE Resta(VAR a:POLINOMIO; b,c:POLINOMIO);
    PROCEDURE Producto(VAR a:POLINOMIO; b,c:POLINOMIO);
    (* Pre: c <> 0 *)
    PROCEDURE Cociente(VAR a:POLINOMIO; b,c:POLINOMIO);

    PROCEDURE Derivar(VAR a:POLINOMIO; p:POLINOMIO);
    PROCEDURE Integrar(VAR a:POLINOMIO; p:POLINOMIO);

    PROCEDURE Evaluar(p:POLINOMIO; x:REAL):REAL;

END BibPol.
```

Estas operaciones presuponen que el polinomio resultado (argumento a) fue creado anteriormente.

Por último construir un programa de prueba que opere con polinomios, importando ambos módulos.