Listas

Una lista es una colección finita de elementos de un mismo tipo, ordenados de acuerdo a su posición en la lista. Una lista lista L de n elementos se denota como L=(e1, e2, ..., en), y una lista vacía como L=(). Las listas pueden crecer o decrecer, en tiempo de ejecución y además subdividirse en sub-listas. Generalmente, las listas se acceden de manera secuencial. Así, mediante una posición es posible moverse dentro de la lista y efectuar operaciones como insertar, eliminar, etc. Cada elemento de una lista se puede estructurar como un nodo, que representa la información que se desea almacenar en la estructura.

Las listas se pueden clasificar como simplemente enlazadas (se acceden en un solo sentido) o doblemente enlazadas (se acceden en ambos sentidos)

Simplemente enlazadas

Cada nodo de una lista simplemente enlazada contiene un apuntador al siguiente elemento de la lista. La especificación de las operaciones de una lista simplemente enlazada general se puede definir como una clase de la siguiente forma:

```
class List <T>        // definicion de un template de listas, donde T indica el tipo de dato
public:
    Type <...> tPosition      // tipo de dato posicion para desplazarse (sin definirse aun)

    Constructor List()        // construye la lista vacia ().
    Destructor List()         //destructor de la clase. Libera la lista de memoria

    function IsEmpty() : Boolean  // retorna true si la lista es vacia, y false en caso contrario
    function First() : tPosition  // retorna la posicion del 1er elemento de la lista
    function Last() : tPosition   // retorna la posicion final de la lista (posicion despues del ultimo↩
        )
    void Next(ref tPosition pValue) // mueve la posicion hacia la posicion siguiente de la lista
    function Get (tPosition pValue): ref T// retorna la referencia a la informacion contenida en pValue
    void Insert (ref T x, tPosition pValue)// inserta x antes de la posicion pValue
    void Delete (tPosition pValue)  // elimina el elemento de pValue. La posicion pValue queda
                                    // referenciando a un elemento inexistente
    function Size() : Integer   // retorna el numero de elementos en la lista
end
```

Doblemente enlazadas

La idea central de una lista doblemente enlazada es poder recorrerla en ambos sentidos (desde el inicio al final, y viceversa). Para ello se requiere de una marca de finalización para cada recorrido. Entonces, es posible utilizar una función End () como parada cuando se recorre desde el inicio al final, y Start() cuando se recorre en sentido inverso. Del mismo modo, se permiten las operaciones de PreInsert y PostInsert debido a que será posible insertar antes y después de una posición dada. La especificación de las operaciones de una lista doblemente enlazada general se puede definir como una clase de la siguiente forma:

```
class ListDouble <T>
public:
    Type <...> tPosition      // tipo de dato posicion para desplazarse

    Constructor ListDouble ()   // construye la lista vacia ()
    Constructor ListDouble (ref List <T> lSource)  // copia lSource en la lista a construir
    Destructor ListDouble ()    // destructor de la clase. Libera la lista de memoria

    function IsEmpty() : Boolean    // retorna True si la lista es vacia, y false en caso contrario
    function Start() : tPosition    // retorna la posicion del inicio de la lista (previo al primero)
    function First() : tPosition    // retorna la posicion del 1er elemento de la lista
    function End() : tPosition    // retorna la posicion final de la lista (despues del ultimo)
    function Last() : tPosition   // retorna la posicion del ultimo elemento de la lista
    void Next(ref tPosition pValue) // mueve la posicion hacia la posicion siguiente de la lista
    void Prev(ref tPosition pValue) // mueve la posicion hacia la posicion previa de la lista
    function Get (tPosition pValue): ref T// Retorna la referencia a la informacion contenida en pValue
    void PreInsert (ref T x, tPosition pValue)// inserta x antes de la posicion pValue.
    void PostInsert (ref T x, tPosition pValue)// inserta x despues de la posicion pValue
    void Delete (tPosition pValue)  // Pre-condicion: pValue es una posicion valida.
                                    // Elimina el elemento de posicion pValue
    function Size() : Integer   // retorna el numero de elementos en la lista
end
```

Existen diversas implementaciones de la clase lista. Adicionalmente, es posible particularizar la estructura de datos para resolver problemas específicos. Por ejemplo, es posible representar los lugares de una mesa redonda como una lista circular (caso particular de lista doblemente enlazada); o emplear múltiples apuntadores dentro de una lista para crear sub-listas (por ejemplo, una lista de enteros donde se ordenen los números pares e impares como sub-listas de ésta).

esmitt ramírez/2014