Tecnología de la programación
Sesión 14

Objetivos de la sesión

1. Ejercicios de listas enlazadas de nodos
2. Ejercicios de recursividad

Guion

Consejo general para los ejercicios de punteros: dibuja lo que quieres hacer.

Ejercicios de listas enlazadas de nodos

Añadir el 10 al principio de una lista enlazada de nodos (transparencia 41)

acción añadir(E/S puntero a Nodo p)

```
variables
      puntero a Nodo nuevo
principio
      nuevo = reservar(Nodo)
      dest(nuevo).dato = 10
      dest(nuevo).sig = p
      p=nuevo
fin
```

Ejercicio de leer enteros del teclado y mostrar en orden inverso (trasparencia 42).

acción leerYmostrarInverso()

```
variables
      puntero a Nodo nuevo, aux, p
      entero n
principio
      p=NULL //Creo la lista
      leer (n)
      mientras que n!=0 hacer
```

```
            nuevo = reservar(Nodo)
            dest(nuevo).dato = n
            dest(nuevo).sig = p
            p=nuevo
            leer(n)
      fmq
      aux=p //Para recorrer la lista enlazada
      mientras que aux != NULL hacer
            escribir(dest(aux).dato))
            aux = dest(aux).sig
      fmq
      mientras que p != NULL hacer //Liberar memoria
            aux = p
            p = dest(p).sig
            liberar(aux)
      fmq
fin
```

Añadir al final de la lista

```
accion añadirAlFinal(E/S Puntero a Nodo p, entero n)
variables
      puntero a Nodo aux
principio
      nuevo = reservar(Nodo)
      dest(nuevo).dato = n
      dest(nuevo).sig = NULL
      si p == NULL
            entonces p = nuevo
      si_no
            aux = p //Recorrido hasta llegar al último nodo
            mientras que dest(aux).sig != NULL hacer
                  aux = dest(aux).sig
```

```
            fmq
            dest(aux).sig = nuevo
      fsi
fin
```

Ver si un elemento está en una lista

```
funcion esta(Puntero a Nodo p, entero n) devuelve booleano
variables
      booleano encontrado
      puntero a Nodo aux
principio
      encontrado = falso
      aux = p
      mientras que aux != NULL and NOT encontrado hacer
            si dest(aux).dato == n entonces
                  encontrado = verdad
            fsi
            aux = dest(aux).sig
      fmq
      dev (encontrado)
fin
```

Ver si un elemento está en una lista (recursivo)

```
funcion estaR(Puntero a Nodo p, entero n) devuelve booleano
principio
      si p == NULL
            dev FALSO
      si_no
            si dest(p).dato==n
                  dev VERDAD
            si_no
                  dev (estaR(dest(p).sig, n))
fin
```

Ejercicio 8E de recursividad

Averiguar si los elementos de un vector v entre las componentes i, j están ordenadas.

```
función ordenado_ij (tVector v, entero i, entero j) dev booleano
{PRE: 0<=u<=j<100, j menor que el número de elementos que tiene el vector}
{POST: …}
Principio
     si i == j
          devuelve VERDAD
     si_no
          si v[i] <= v[i+1] entonces
               devuelve (ordenado_ij (v,i+1,j))
          si_no
               devuelve FALSO
          fsi
     fsi
fin
```

Ejercicio 8F de recursividad

Decidir si alguna de las componentes del vector es suma de las de su izquierda

```
función suma (tVector v, entero n) dev booleano
{PRE: v vector de tamaño n, 0<=n<100}
{POST: devuelve la suma de las n componentes del vector}
Principio
     si n==0
          devuelve 0
     si_no
          devuelve (v[n-1] + suma (v,n-1))
fin
```

```
función sumaAnteriores (tVector v, entero n) dev booleano
{PRE: v vector de tamaño n, 1<=n<100}
{POST: devuelve verdad si existe i tal que 1<=i<n tal que
```
$\sum_{j=0}^{j=i-1} v[j] = v[i]$
```
y falso en caso contrario}
Principio
     si n==1
          devuelve FALSO
     si_no
          si suma(v,n-1) == v[n-1]
               dev VERDAD
          si_no
               dev (sumaAnteriores(v,n-1))
          fsi
     fsi
fin
```

Siendo precisos, en el caso n==1 habría que hacer otra vez dos casos: si v[0] == 0 entonces devuelves VERDAD y si no devuelves FALSO.

Esto se debe a que matemáticamente, la suma de un conjunto vacío es 0.

Es ineficiente, ¡¡Date cuenta de que estamos recalculando la suma todo el rato!!

Ejercicio 8F de recursividad (alternativa eficiente con inmersión)

```
función sumaAnterioresInmersion (tVector v, entero n, entero
suma) dev booleano
principio
     Si n==1
          dev FALSO
     si_no
          si suma - v[n-1] == v[n-1]
               dev VERDAD
          si_no
               dev sumaAnterioresInmersion(v,n-1,suma-v[n-1])
          fsi
     fsi
fin
```

```
función sumaAnteriores (tVector v, entero n) dev booleano
principio
     entero s = suma(v,n)
     dev sumaAnterioresInmersion(v,n-1,s-v[n-1])
fin
```

Nota: también se puede resolver haciendo la suma de todos los elementos menos el último (es decir, que la primera línea sea `entero s = suma(v,n-1)`), adaptando el resto del código adecuadamente.