Tecnología de la programación
Sesión 15

Objetivos de la sesión

1. Estudiar las transparencias de subalgoritmos con parámetros de tipo puntero (de la 43 a la 47)
2. Hacer el ejercicio leer enteros de teclado y mostrarlo en orden inverso utilizando subalgoritmos (transparencia 48)
3. Repasar los ejercicios de esta un entero en una lista y la de añadir al final. Fueron resueltos en la sesión anterior.
4. Hacer el ejercicio de añadir en una lista ordenada de nodos.
5. Deberes: subalgoritmo para eliminar todas las ocurrencias de un entero n de una lista de nodos (no ordenada).

Guion

Subalgoritmos con parámetros de tipo puntero

Es muy importante darnos cuenta de que si usas punteros en subalgoritmos puedes estar modificando cosas en memoria que afecten al programa principal, aunque hayas pasado el puntero como parámetro de entrada.

Lo primero, contaros que os han "engañado" clasificando los parámetros de un subalgoritmo dependiendo de si son de entrada, salida o entrada salida. Lo que existe en programación es paso por valor y paso por referencia.

Paso por valor significa que el subalgoritmo trabaja con una COPIA de ese parámetro. Es lo que solemos asociar con parámetros de entrada. En el caso de, por ejemplo, variables de enteros tendríamos:

```
Programa Principal
Variables
      entero x
Principio
      x = 5
      subalgoritmo1(x)
      escribir(x)
Fin

acción subalgoritmo1(Ent/ entero x)
Principio
      x = x+15
Fin
```

El resultado de ejecutar lo anterior será simplemente mostrar un 5 por pantalla. El subalgoritmo tiene el parámetro x de entrada, por tanto, el

“nuevo” valor de x no se pasa al principal. Esto ocurre porque se ha hecho paso por valor: el subalgoritmo realmente está trabajando con una copia de la variable x, y el programa principal no modifica su x en ningún momento.

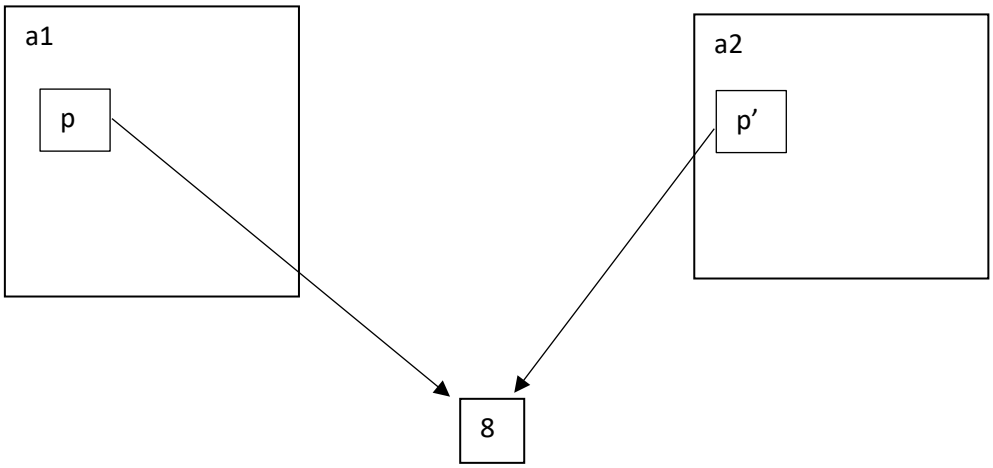
En el siguiente dibujo se ve que son dos variables distintas. Llamo x a la variable en el programa principal (que tiene el valor 5), y x’ a la copia de x en el subalgoritmo (que empieza valiendo 5 también). Cuando se ejecuta el subalgoritmo, lo que sucede es que x’ pasa a valer 20, el subalgoritmo acaba y el valor de x’ se pierde. El valor de x queda intacto, y por eso se mostrará por pantalla un 5.

Paso por referencia significa que el subalgoritmo no recibe una copia de la variable, sino directamente la dirección de memoria del dato. Es decir, que accedes al mismo trozo de memoria desde dos sitios distintos (el principal y el subalgoritmo), y por tanto, las modificaciones que hagas en el subalgoritmo afectan al programa principal. Se suele relacionar con los parámetros de salida y de entrada/salida.

¿Pero qué pasa con los punteros? Que aunque el puntero lo pongas de entrada (es decir, paso por valor, es decir, copias el puntero), realmente el principal podría ver los cambios porque tienes dos cosas distintas que apuntan al mismo trozo de memoria. Es como tener dos mandos a distancia para la misma tele: si la apagas desde uno, el otro se ha enterado. Veámoslo con el ejemplo de las transparencias.

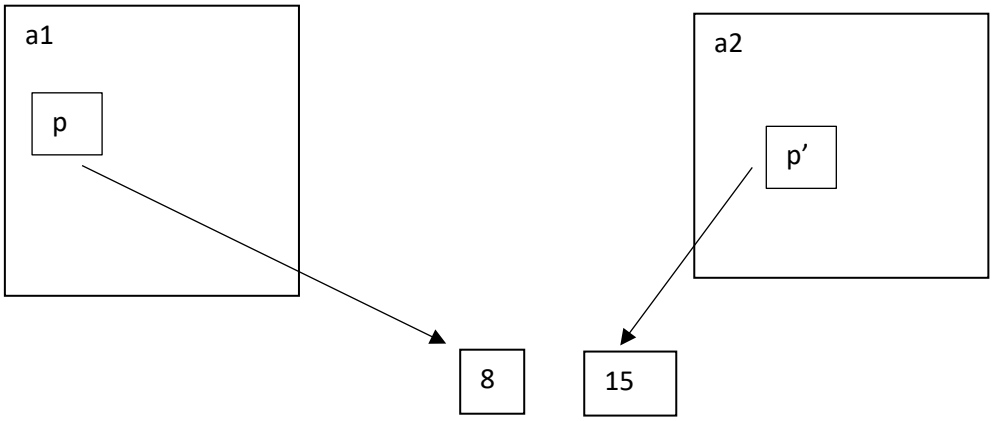
```
acción a1(...)
variables
      puntero a entero p
principio
      p = reservar(entero)
      dest(p) = 8
      a2(p)
      ...
fin
```

Si a2 recibe el puntero p como parámetro de entrada, entonces tenemos esta situación (llamo p' a la copia de p en el subalgoritmo a2):



Entonces, si modificamos desde a2 el destino del puntero, el a1 se enterará, a pesar de que p era parámetro de entrada. Y lo mismo si se libera memoria.

Pero ojo, que si el puntero p es un parámetro de entrada, y en el subalgoritmo lo apuntamos a otro sitio, por ejemplo, a otra variable que tenga el valor 15, entonces el subalgoritmo a1 mantiene el mismo valor (porque no se ha cambiado a lo que apunta su puntero, sino que lo que se ha cambiado es el lugar a donde apunta la copia del puntero p).



Por estas razones, solemos trabajar siempre poniendo que los punteros son de entrada/salida, para saber que los cambios pueden afectar. Pero cuidado, que NO es lo mismo poner que un puntero es de entrada que poner que es de entrada/salida (en el primer caso el subalgoritmo trabaja con una copia, en el segundo caso el subalgoritmo trabaja directamente con el mismo puntero). De hecho, si en el último dibujo pasamos el puntero p a a2 como entrada/salida, entonces si el subalgoritmo a2 cambia el puntero de sitio, el

a1 se entera. En cualquier caso, si estamos seguros de que no afecta, lo podemos poner de entrada.

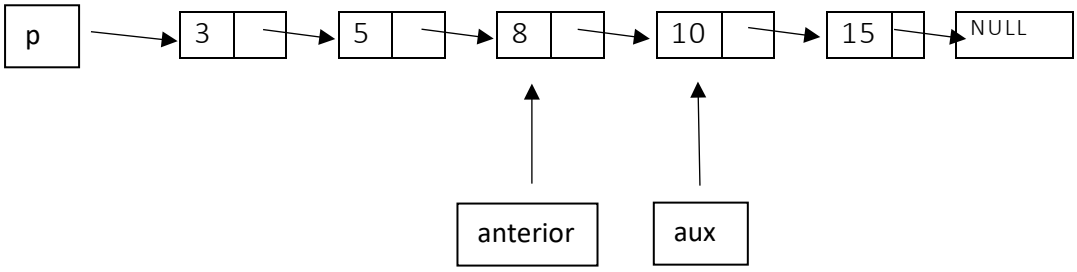
Ejercicio de leer enteros de teclado y mostrarlo en orden inverso utilizando subalgoritmos (transparencia 48)

```
accion crearLista(Ent/Sal puntero a Nodo p)
Variables
     Entero n
Principio
     p=NULL
     leer(n)
     mientras que n!=0 hacer
          añadir(p,n)
          leer(n)
     fmq
fin

acción añadir(Ent/Sal puntero a Nodo q, Ent/ entero num)
variables
     puntero a Nodo nuevo
principio
     nuevo = reservar(Nodo)
     dest(nuevo).dato=num
     dest(nuevo).sig = q
     q=nuevo
fin

acción mostrar (Ent/ puntero a Nodo q)
variables
     puntero a Nodo aux
principio
     aux = q
     mientras que aux != NULL hacer
          escribir (dest(aux).dato)
          aux = dest(aux).sig
     fmq
fin

acción liberar (Ent/Sal puntero a Nodo q)
variables
     puntero a Nodo aux
principio
     mientras que q != NULL hacer
          aux = q
          q=dest(q).sig
          liberar(aux)
     fmq
fin
```

```
Algoritmo Principal
variables
      puntero a Nodo p
principio
      crearLista(p)
      mostrar(p)
      liberar(p)
fin
```

Ejercicio: Añadir en lista ordenada de nodos

Hay varias formas distintas de hacer este ejercicio. Esta es una de ellas. Usaremos dos punteros auxiliares para ir recorriendo la lista, para poder añadir en el sitio correcto. Por ejemplo, si queremos añadir un 9 a la siguiente lista, lo que haremos es ir moviendo aux hasta que nos encontremos un valor más grande. Entonces sabremos dónde tenemos que insertar, el problema es que aux apunta ya al siguiente (al 10) y nos hemos pasado (queríamos añadirlo en la posición anterior). Por eso es importante tener un segundo puntero que guarde la posición anterior.




```
Acción añadirOrdenado(E/S puntero a Nodo p, Ent/ entero n)
Variables
      Puntero a Nodo aux, anterior, nuevo
Principio
      nuevo = reservar(Nodo)
      dest(nuevo).dato=n
      anterior=NULL
      aux=p
      mientras que aux != NULL AND dest(aux).dato<n
            anterior=aux
            aux=dest(aux).sig
      fmq
      dest(nuevo).sig=aux
      si anterior != NULL
            dest(anterior).sig=nuevo
      si_no
            p=nuevo //Con esto añadimos al principio de la lista
      fsi
fin
```