Void borrar\_nodo(raiz, dato)

ap= buscar\_nodo(raiz, dato) //ap apuntador al nodo por borrar

//padre= padre(raiz, ap) recomendación

// hijo por la izq o por la derecha (ap, padre) recomendación

Caso 1 ap es terminal ?

borrar\_terminal(..)

Return

Caso 2

Ap tiene un sólo hijo?

Borrar\_nodo\_un\_hijo (..)

Return

Caso 3 ap tiene dos hijos?

Borrar\_nodo\_dos\_hijo (ap ,)

}

Void Borrar\_nodo\_dos\_hijo ( ap , ){

Aux = ap -> izq //Ir al hijo izq de ap,

// recursión Irse toda derecha; preferente función

Aux = recursion\_derecha( Aux )

// sencillo, sólo copiar info/dato de aux en ap ;

ap->dato = aux ->dato;

//Hijo izq de aux sube al lugar de aux, ojo no es simple copia de dato!

Padre de aux // pendiente por resolver, modificar esta rutina para saber padre de aux,

Padre\_aux -> der = aux -> izq ; //nos saltamos a aux

free aux;

}

main ( ){

//.. Aquí va Tarea anterior..

raiz= crear\_arbol (..)

//..

//Borrando …

arre\_borrar= [ , , ] // valores del ejemplo 6.14 del libro

ciclo en arre\_borrar {

Print ‘borrando dato’ , arre\_borrar[i]

borrar\_nodo(raiz, arre\_borrar[i] )

Imprimir\_in\_orden(..)

}

Imprimir\_indice (raiz)

}

//para caso 3, ¿por qué no sólo subir algún hijo? , ¿porqué complicar con recursion\_derecha()?