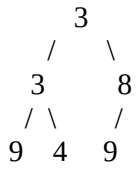
Problema 1 (40 puntos)

Un árbol binario está ***parcialmente ordenado*** si para cada nodo del árbol se cumple que el valor de información del nodo es menor o igual que el de todos los nodos que están en sus subárboles (izquierdo y derecho). El árbol vacío está parcialmente ordenado. Por ejemplo, el siguiente es un árbol binario parcialmente ordenado de enteros:



Considere la siguiente declaración de nodos de árboles binarios de enteros:

```
struct ABNode { int info;
                ABNode * left;
                ABNode * right;}
```

Implemente la función recursiva ***ordParcial*** que dado un árbol binario de enteros retorne true si y solo si el árbol está parcialmente ordenado. No se permite definir y usar operaciones auxiliares propias.

`bool ordParcial (ABNode * t)`

Solución

```
bool ordParcial (ABNode * t) {
        bool resultado = true;
        if (t == NULL) return resultado;
        else{
                if (t->left != NULL)
                        resultado = t->info <= t->left->info;
                if (t->right != NULL)
                        resultado = resultado &&  t->info <= t->right->info;
                return resultado && ordParcial(t->left) && ordParcial(t->right);
        }
}
```

Ejercicio 3 (60 puntos)

Una empresa quiere tener un sistema que permita gestionar el stock de sus productos. Cada producto se identifica con una cadena de caracteres y se puede tener una cantidad arbitraria de unidades para cada uno. El sistema que pretende la empresa debería contemplar al menos las siguientes funcionalidades básicas:

i. Agregar *n* unidades de un producto dado al stock. Si el producto no había sido ingresado al sistema se ingresa con *n* unidades.

ii. Dar de baja a lo sumo *n* unidades de un producto dado al stock. No tiene efecto si el producto no está ingresado.

iii. Saber dado un producto la cantidad de unidades disponibles en el stock. En caso de no estar ingresado devuelve 0.

Se pide:

a) Indique un TAD de los vistos en el curso que resulte particularmente adecuado para modelar el stock de productos. Especifique el TAD ***stock*** con operaciones constructoras, selectoras/destructoras y predicados para operar con el stock de productos de la empresa.

b) Proponga una implementación que resulte O(1) caso promedio para las tres funcionalidades previamente mencionadas (i, ii y iii), asumiendo que se esperan aproximadamente ***N*** productos y que se tiene disponible la función ***H: String→[0,N-1]***. Justifique, asumiendo las hipótesis que considere necesarias. Defina la representación del TAD e implemente únicamente los códigos de las operaciones i) y iii). Omita el código del resto de las operaciones del TAD.

c)

Solución

a) Dado que se pide representar una función parcial de f(id)→cant_producto se puede usar el TAD Multiconjunto o el TAD Tabla.

Especificación *TAD Stock como Multiconjunto*

```
/*Crea un multi conjunto vacío para aproximadamente N elementos distintos*/
Stock crearStock (int N);

/*Se agregan elems al producto id. Si el producto no existe se inserta con elems cantidad de elementos.
*/
void agregarElementosProducto (String id, int elems, Stock &s);

/*Da de baja a lo sumo n unidades del producto id. No tiene efecto si el producto no está ingresado. */
void quitarElementosProducto (String id, int elems, Stock &s);

/*Devuelve la cantidad de elementros asociado al producto id. En caso de no estar ingresado devuelve 0.
*/
int cantidadElementosProducto (String, Stock s);

/*Libera toda la memoria asociada al mantenimiento de la tabla t*/
void eliminarStock (Stock &s);
```

b) Dado que se requiere O(1) promedio en las funciones agregar, quitar y consultar la cantidad de elementos, y que se conoce la cantidad de elementos aproximada que va a manejar la empresa, utilizar una tabla de hash como implementación es lo más apropiado.

En particular utilizaremos una tabla de hash abierta de tamaño N. Para resolver las colisiones, en cada *bucket* de la tabla mantendremos una lista encadenada de los productos.

```
struct nodo
{
String id;
        int cantidad;
        nodo* sig;
}
```

```
struct tablaH
{	nodo ** hash;
}

typedef TablaH * Stock;


void agregarElementosProducto (String id, int elems, Stock &s)
{	int pos = H(id);
	nodo * iter = t→ hash[pos];
	bool encontre = false;
	while (iter!= NULL && !encontre)
	{
		if (strcmp(iter->id, id) == 0)
		{
			encontre = true;
			iter->cantidad+= elems;
		}
		else
			iter = iter -> sig;
	}
	if (!encontre)
	{
		nodo * nuevo_prod = new(nodo);
		nuevo_prod→id = id;
		nuevo_prod→cantidad = elems;
		nuevo_prod→sig = t→hash[pos];
		t→hash[pos] = nuevo_prod;
	}
}




/*Esta implementación deja productos con 0 unidades, también se podrían eliminiar por completo. */
void quitarElementosProducto (String id, int elems, Stock &s)
```

```
{       int pos = H(id);
        nodo * iter = t→ hash[pos];
        bool encontre = false;
        while (iter != NULL && !encontre)
        {
                if (strcmp(iter->id, id) == 0)
                {
                        encontre = true;
                        iter->cantidad = min(iter->cantidad-elems, 0);
                }
                else
                        iter = iter -> sig;
        }
}


int cantidadElementosProducto (String id, TablaStock t)
{       int cantidad = 0;
        int pos = H(id);
        nodo * iter = t→ hash[pos];
        bool encontre = false;
        while (iter != NULL && !encontre)
        {
                if (strcmp(iter->id, id) == 0)
                {
                        encontre = true;
                        cantidad = iter→cantidad;
                }
                else
                        iter = iter -> sig;
        }
        return cantidad;
}
```