Algoritmos y Estructuras de datos
Ejercicios previos al tercer examen parcial

- Escribe una implementación de la función `kesimo_valor`, la cual toma un apuntador al nodo inicial una lista enlazada y un entero `k` y devuelve el valor del `k`-ésimo nodo de la lista enlazada (es decir, el valor del nodo al que llegamos después de avanzar `k` veces sobre la lista). Puedes suponer que $k \geq 0$ y que la lista enlazada tiene al menos `k`+1 nodos.

```
struct nodo {
   int valor;
   nodo* sig;
};

int kesimo_valor(nodo* ini, int k) {
   for (int i = 0; i < k; ++i) {
      ini = ini->sig;
   }
   return ini->valor;
}
```

- Escribe una implementación de la función `cuenta_nodos`, la cual toma un apuntador a *algún* nodo válido de una lista enlazada y devuelve cuántos nodos tiene dicha lista. El primer nodo de la lista apunta a `nullptr` como su nodo anterior y el último apunta a `nullptr` como su siguiente nodo.

```
struct nodo {
   nodo* ant;
   nodo* sig;
};

int cuenta_nodos(nodo* p) {
   int res = 0;
   for (nodo* i = p; i != nullptr; i = i->ant) {
      ++res;
   }
   for (nodo* i = p; i != nullptr; i = i->sig) {
      ++res;
   }
   return res - 1;
}
```

- Escribe una implementación de la función `altura`, la cual toma un apuntador al nodo raíz de un árbol binario y devuelve la altura de dicho árbol. Las hojas (los nodos sin hijos) apuntan a `nullptr` como sus nodos hijos. Tienes disponibles las funciones de `algorithm` la biblioteca de C++.

```
struct nodo {
   nodo* izq;
   nodo* der;
};

int altura(nodo* p) {
   return (p == nullptr ? 0 : std::max(altura(p->izq), altura(p->der)) + 1);
}
```

- Escribe una implementación de la función `cuenta_nodos`, la cual toma un apuntador al nodo raíz de un árbol binario, y devuelve cuántos nodos tiene el árbol. Las hojas (es decir, los nodos sin hijos) apuntan a `nullptr` como sus nodos hijos.

```
struct nodo {
   nodo* izq;
   nodo* der;
};

int cuenta_nodos(nodo* p) {
   return (p == nullptr ? 0 : cuenta_nodos(p->izq) + cuenta_nodos(p->der) + 1);
}
```

- Escribe una implementación de la función `cuenta_apariciones`, la cual toma un apuntador al nodo raíz de un árbol binario además de un entero y devuelve cuántas veces aparece el entero en el árbol. Las inserciones en el árbol se realizaron como sigue: todos los valores del subárbol izquierdo de un nodo son menores o iguales al valor de dicho nodo, mientras que todos los valores del subárbol derecho son mayores. Las hojas (es decir, los nodos sin hijos) apuntan a `nullptr` como sus nodos hijos. Tu función no debe buscar en partes del árbol donde se sabe (por la regla de inserción) que el entero no aparece.

```
struct nodo {
   int valor;
   nodo* izq;
   nodo* der;
};

int cuenta_apariciones(nodo* p, int v) {
   if (p == nullptr) {
      return 0;
   } else if (v > p->valor) {
      return cuenta_apariciones(p->der, v);
   } else {
      return cuenta_apariciones(p->izq, v) + (p->valor == v);
   }
}
```

- Escribe una implementación de la función `cuenta_hojas`, la cual toma un apuntador al nodo raíz de un árbol binario y devuelve la cantidad de hojas de dicho árbol. Las hojas (es decir, los nodos sin hijos) apuntan a `nullptr` como sus nodos hijos.

```
struct nodo {
   nodo* izq;
   nodo* der;
};

int cuenta_hojas(nodo* p) {
   if (p == nullptr) {
      return 0;
   } else if (p->izq == nullptr && p->der == nullptr) {
      return 1;
   } else {
      return cuenta_hojas(p->izq) + cuenta_hojas(p->der);
   }
}
```