

## Algoritmos y Estructuras de Datos. 1er Parcial. [2013-10-03]

**ATENCIÓN(1):** Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría (Ej 4) y 50 % en las restantes secciones.

**ATENCIÓN(2):** Recordar que tanto en las clases (Ej. 1 ) como en los ejercicios de programación (Ej 2.) **deben usar la interfaz STL.**

### [Ej. 1] [clases (20pt)]

- [lista (10pt)].** Escribir la implementación en C++ del TAD lista (clase `list`) implementado por celdas (simple o doblemente) enlazadas por punteros ó cursores. (**Indicar claramente qué implementación elige.**) Los métodos a implementar son `insert(p,x)`, `erase(p)`, `iterator::operator++(int)` (postfijo), `iterator::operator++()` (prefijo).
- [AOO (10pt)].** Declarar las clases `tree`, `cell`, `iterator`, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método `tree<T>::iterator tree<T>::insert(tree<T>::iterator n, const T& x)`

### [Ej. 2] [Programación (40pt)] Recordar que en los ejercicios de programación **deben usar la interfaz STL.**

- [nchild (20pt)].** Escribir una función predicado `bool nchild(tree<int> &T, int n)`; que dado un árbol ordenado orientado (AOO) `T` chequea si cada hijo tiene a lo sumo `n` hijos. Por ejemplo si `T=(a (b f g h) (c z (d e)))`. Entonces `nchild(T,3) -> true` ya que el máximo número de hijos es 3 para el nodo `b` mientras que `nchild(T,2) -> false`.
- [evencount (20pt)].** Escribir una función predicado `bool evencount(map<int, list<int>> &M)`; que chequea que para cada par `(k,L)` en `M` se verifique que la lista `L` contiene exactamente `k` elementos pares. Por ejemplo, si

```
M1[0] = (5,7,9)
M1[3] = (5,2,3,4,6)
```

entonces `evencount(M1)->true`. Mientras que si

```
M2[2] = (5,7,8)
M2[4] = (5,2,3,4,6,8)
```

debe retornar `evencount(M2)->>false` ya que para la clave 2, la lista correspondiente tiene un solo elemento par.

**Ayuda:** Escribir una función `int evencount(list<int> &L)`; que cuenta cuantos pares hay en la lista `L`.

### [Ej. 3] [operativos (20pt)]

- [rec-arbol (10pt)]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son
  - ORD\_PRE = {A, Z, W, B, C, D, Q, R, L, F},
  - ORD\_POST = {B, C, D, W, L, R, Q, Z, F, A},
- [particionar (10pt)].** Considerando el árbol `(a (b f g h) (c z (d e)))` decir cuál son los nodos descendientes(`z`), antecesores(`z`), izquierda(`z`) y derecha(`z`). (**Nota:** se refiere a antecesores y descendientes **propios**).

### [Ej. 4] [Preguntas (total = 20pt, 4pt por pregunta)]

- Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones  $T_1, \dots, T_5$  determinar su velocidad de crecimiento (expresarlo con la notación  $O(\cdot)$ ).

$$T_1 = n + 5 \log n,$$

$$T_2 = 10n^3 + 2n! + 5n^4 + 2^n,$$

$$T_3 = 23! + 3 \log_2 n + 4,$$

$$T_4 = 2 + 2n^2 + 5n^3,$$

$$T_5 = n^3 + 3^n + 10.$$

- b) ¿Es la **pila** un contenedor **FIFO** o **LIFO**? ¿Que significa? Responda la misma pregunta para el TAD **cola**.
- c) Enuncie la **regla de la suma** para la notación asintótica  $O()$ . De ejemplos.
- d) ¿Qué ventajas o desventajas tendría implementar la clase *pila* en términos de **lista simplemente enlazada** poniendo el tope de la pila en el fin de la lista?
- e) ¿Cual es la principal ventaja de implementar correspondencias con **vectores ordenados**? ¿Se obtiene la misma ventaja con **listas ordenadas**?