

Algoritmos y Estructuras de Datos.

1er Parcial. Tema: 2A. [27 de abril de 2004]

[Ej. 1] [Clases (20 puntos)] Escribir la implementación en C++ del TAD CORRESPONDENCIA (clase map) implementado por contenedores lineales (listas o vectores, ordenados o no ordenados). Las funciones a implementar son `find(key)`, `insert(key, val)`, `retrieve(key)`, `erase(p)`, `key(p)`, `value(p)`, `begin()`, `end()`, `clear()`. Observaciones:

- Escribir tanto las declaraciones como las funciones (archivos `.h` y `.cpp`).
- Incluir las definiciones de tipo (`typedef`) y clases auxiliares necesarias.
- Se puede escribir la interface avanzada (con templates, clases anidadas, sobrecarga de operadores).

[Ej. 2] [Programación (total = 60 puntos)]

- a) [mayorar (30 puntos)] Escribir una función `void mayorar(list<int> &L1, list<int> &L2);` que modifica las listas L1 L2 de tal manera que si a_{1j}, a_{2j} son los elementos de L1 y L2 *antes de aplicar la función* y a'_{1j}, a'_{2j} los elementos *después* de aplicar la función, entonces $a'_{1j} = \max(a_{1j}, a_{2j})$, $a'_{2j} = \min(a_{1j}, a_{2j})$. Si las listas no tienen la misma longitud, entonces los elementos restantes quedan inalterados. Ejemplo: si $L1=(14, 0, 6, 13, 11, 12, 3, 17, 14, 18)$ y $L2=(6, 4, 4, 11, 12, 15, 8, 17, 18, 11, 23, 1, 2, 5, 15)$ entonces después de hacer `mayorar(L1, L2)` debe quedar $L1=(14, 4, 6, 13, 12, 15, 8, 17, 18, 18)$ y $L2=(6, 0, 4, 11, 11, 12, 3, 17, 14, 11, 1, 2, 5, 15)$. Se sugiere el siguiente algoritmo: Recorrer ambas listas con dos posiciones e ir intercambiando los elementos si $a_{1j} < a_{2j}$.

Restricciones:

- Usar la interfase STL para listas.
 - No usar el operador `--`.
 - No usar ninguna estructura auxiliar.
 - El algoritmo debe ser $O(n)$.
 - Prestar a no usar posiciones inválidas al iterar sobre las listas.
- b) [creciente (15 puntos)] Escribir una función `void creciente(queue<int> &Q)` que elimina elementos de Q de tal manera de que los elementos que quedan estén ordenados en forma creciente. Por ejemplo, si $P=(5, 5, 9, 13, 19, 17, 16, 20, 19, 21)$, entonces después de hacer `creciente(Q)` debe quedar $P=(5, 5, 9, 13, 19, 20, 21)$. Usar una cola auxiliar. Se sugiere utilizar el siguiente algoritmo: Ir eliminando los elementos de la pila Q y ponerlos en la cola auxiliar Q2 sólo si el elemento es mayor o igual que el máximo actual. Finalmente volver a pasar todos los elementos de Q2 a Q. **Restricciones:**
- Usar la interfase STL para colas.
 - No usar más estructuras auxiliares que la indicada ni otros algoritmos de STL.
 - El algoritmo debe ser $O(n)$.
- c) [cum-sum-pila (15 puntos)] Escribir una función `void cum_sum(stack<int> &P)` que modifica a P dejando la suma acumulada de los elementos, es decir, si los elementos de P antes de llamar a `cum_sum(P)` son $(a_0, a_1, \dots, a_{n-1})$, entonces después de llamar a `cum_sum(P)` debe quedar $P=(a_0, a_0 + a_1, \dots, a_0 + a_1 + \dots + a_n)$. Por ejemplo, si $P=(1, 3, 2, 4, 2)$ entonces después de hacer `cumsum(P)` debe quedar $P=(1, 4, 6, 10, 12)$. Usar una pila auxiliar.
- Restricciones:**
- Usar la interfase STL para pilas (`clear()`, `top()`, `pop()`, `push(T x)`, `size()`, `empty()`).

