Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Parcial 1. [2014-09-18]

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)]

a) **[list (10pt)]** Escribir la implementación en **C++** del TAD lista (clase **`list`**) implementado por punteros **ó** cursores. Los métodos a implementar son **`insert(p,x)`**, **`erase(p)`**, **`iterator::operator++(int)`**.
b) **[stack-queue (10pt)]** Escribir la implementación en **C++** de los métodos **`push`**, **`pop`**, **`front`**, y **`top`** de los TAD pila y cola (clases **`stack`** y **`queue`**), según corresponda.
c) **[map (10pt)]** Escribir la implementación en **C++** del TAD correspondencia (clase **`map`**) implementado con listas ordenadas. Métodos a implementar: **`find(key)`**.

[Ej. 2] [Programación (50pt)] Recordar que en los ejercicios de programación **deben usar la interfaz STL**.

a) **[stack-sep (10pt)]**. Dada una pila **`S`**, escribir una función que deja en el tope los elementos pares y en el fondo los impares. El algoritmo debe ser **estable** es decir que los pares deben estar en el mismo orden entre si y los impares también. Por ejemplo si **`S=(0,1,2,3,4,5,6)`** entonces debe quedar **`S=(0,2,4,6,1,3,5)`**.
Consigna: Escribir una función **`void stacksep(stack<int> &S);`** que realiza la tarea indicada.
Restricciones: Como estructuras auxiliares sólo pueden usar pilas. El algoritmo no debe alocar memoria adicional, es decir la memoria adicional alocada debe ser $O(1)$.

b) **[exists-path (20pt)]**. Sebastián un buen día queda en juntarse para comer un asado en la casa de Santiago. Pero resulta que si bien son buenos para organizar reuniones, no son buenos para recordar fechas, y olvidaron que ese día habrán varios desfiles por el carnaval, quedando solo algunas calles de un único sentido habilitadas para los vehículos.

Sebastián es un conductor muy prudente, y no se arriesgará a que lo multen, por lo que solo tomará los caminos en el sentido correcto.

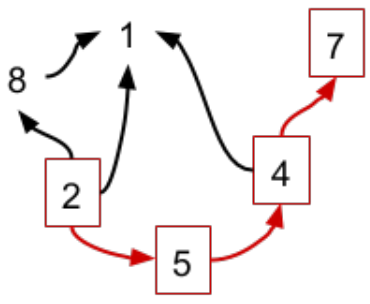
Entonces con rapidez arma un mapa (grafo) de las calles habilitadas que tendrá a las intersecciones de las calles como nodos y a las calles como aristas. Seba le pide a usted (ya que él está muy ocupado haciendo la ensalada para la cena) que escriba una función
`bool exists_path (map <int, list <int> > &M, int m, int n);` que dado un mapa **`M`**, le haga saber si existe al menos un camino que él pueda hacer desde su casa (ubicada en la intersección **`m`**) hacia la casa de Santiago, que está en la intersección **`n`**.

¿Podrán juntarse, o deberán posponer la reunión?

Ejemplo: Dado un mapa **`M`**

Entonces si **`m=2`**, y **`n=7`** el resultado debe ser verdadero.

```
1 >( )
2 >( 1, 5, 8 )
4 >( 1, 7 )
5 >( 4 )
7 >( )
8 >( 1 )
```



c) **[extractm (20pt)]**.
Dada una lista **L**, y un entero **m**, escribir una función
`void extractm(list<int> &L,int m,list<int> &Lm);` que genere una lista **Lm** con todos los elementos que aparecen exactamente **m** veces en **L**. Por ejemplo, si **`L=(0,1,2,3,4,5,1,2,4,2)`** entonces **`extractm(L,2,Lm)`** debe dejar **`Lm=(1,4)`** ya que son los únicos dos elementos que están exactamente 2 veces. **`extractm(L,1,Lm)`** debe retornar **`Lm=(0,3,5)`**.
Restricciones: Construir una correspondencia **`map<int,int> F`** cuyas claves son los elementos de **L** y la imagen la **frecuencia**, es decir cuántas veces aparece en **L**. Luego se recorre **F** y se insertan en **Lm** sólo las que tienen frecuencia **m**.

[Ej. 3] [Preguntas (total = 20pt, 4pt por pregunta)]

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

$$
\begin{aligned}
T_1 &= 3n^4 + 4n^3 + 20, \\
T_2 &= 4^n + 5n! + 3n^{10} + 4n^5, \\
T_3 &= 10 + 2^n + n^2, \\
T_4 &= 4 + 2\log_{10} n + 50!, \\
T_5 &= 2\log_3 n + 5n.
\end{aligned}
$$

b) Sea la correspondencia **`M={(2->1),(8->5)}`** y ejecutamos el código int **`x= M[8]`**. ¿Qué ocurre? ¿Qué valores toman **x** y **M**? ¿Y si hacemos **`x = M[1]`**?

c) ¿Cuál es la complejidad algorítmica (mejor/promedio/peor) de las siguientes funciones? (asumir listas implementadas con celdas enlazadas por punteros o cursores)

1) **`list<T>::begin()`**,
2) **`list<T>::insert(p,x)`**,
3) **`list<T>::clear(p,x)`**,
4) **`map<K,V>::find(x)`**, para la implementación con vectores ordenados,
5) **`map<K,V>::find(x)`**, para la implementación con listas ordenadas.

d) Discuta las ventajas y desventajas de utilizar listas doblemente enlazadas con respecto a las simplemente enlazadas. ¿Cuáles son los métodos cuyo tiempo de ejecución cambia y por qué?

e) ¿Qué ventajas o desventajas tendría implementar la clase **stack** (pila) en términos de **lista simplemente enlazada** poniendo el tope de la pila en el fin de la lista? ¿Cuáles serían los tiempos de ejecución para **`top`**, **`pop`**, **`push`**?
Lo mismo para cola: ¿Qué ventajas o desventajas tendría implementar la clase **queue** (cola) en términos de **lista simplemente enlazada** poniendo el frente de la pila en el fin de la lista? ¿Cuáles serían los tiempos de ejecución para **`front`**, **`pop`**, **`push`**?