Apellido y Nombre: _____

Carrera: _____ DNI: _____

*[Llenar con letra mayúscula de imprenta GRANDE]*

**Universidad Nacional del Litoral**
**Facultad de Ingeniería y Ciencias Hídricas**
**Departamento de Informática**

**Algoritmos y Estructuras de Datos**

# Algoritmos y Estructuras de Datos.
# 1er Parcial. Tema: 2A. [21 de abril de 2005]

**[Ej. 1]** **[Clases (30 puntos)]** Escribir la implementación en `C++` del `TAD LISTA` (clase `list`) implementado por punteros **ó** cursores **ó** arreglos. Las funciones a implementar son `insert(p,x)`, `erase(p,q)`, `begin()`, `end()`, `retrieve()/operator*()`, `~list()`. Observaciones:

- En caso de optar por escribir la interfase *"básica"*, debe escribir todas las declaraciones necesarias de la clase, tanto en la parte privada como pública.
- En caso de optar por la interfase *"avanzada"*, debe declarar e implementar completamente las partes privadas de la clase `list` e `iterator`.

**[Ej. 2]** **[Programación (total = 50 puntos)]**

*a)* **[lexico-stack (25 puntos)]** Considere el problema de generar todas las subsecuencias ordenadas de la secuencia $X = (1, 2, ..., n)$.

Por ejemplo, si $n = 4$ las subsecuencias ordenadas de $X = (1, 2, 3, 4)$ son: (1), (12), (123), (124), (13), (134), (14) (2), (23), (234) (24), (3), (34) y (4).

Esta construcción se puede implementar mediante el uso de una pila $S$ bajo las siguientes reglas:

- Inicializar la pila con el elemento 1.
- Si el tope $t$ de la pila verifica $t < n$ entonces apilamos $t + 1$.
- Si $t = n$, entonces lo desapilamos y, a continuación, si la pila no quedara vacía incrementamos el nuevo tope de la misma.
- El algoritmo termina cuando la pila queda vacía.

Ejemplo:

```
          4
        3 3 4     4         4
      2 2 2 2 3 3 4     3 3 4     4
    1 1 1 1 1 1 1 1 2 2 2 2 3 3 4
```

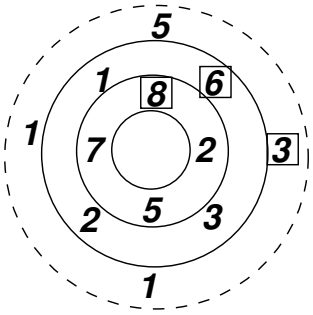
*Consigna:* Escriba un procedimiento `void lexico_stack(int & n);` el cual, ingresado el número natural $n$ imprime todos los conjuntos ordenados de $(1, 2, ..., n)$. *Sugerencia:* Implementar el algoritmo descripto, llamando a una función auxiliar `void imprime_pila(stack<int> &S)` (implementarla!!) que imprime la pila `S` en forma no-destructiva.

**Restricciones:**

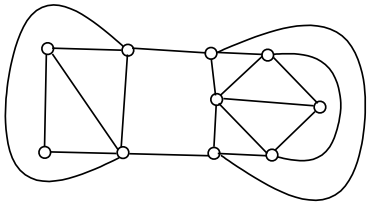
- Usar la interfase STL para pilas.
- En `lexico_stack()`: usar una sola estructura auxiliar.
- En `imprime_pila()`: usar una sola estructura auxiliar.
- No usar otros algoritmos de STL.

*b)* **[circulo (25 puntos)]** Coloquemos $n$ números enteros positivos alrededor de una circunferencia inicial. Construyamos ahora sucesivas circunferencias concéntricas *hacia el exterior*, de igual cantidad de elementos, los cuales son obtenidos restando (en valor absoluto) pares consecutivos de la última circunferencia exterior. Entonces, puede verificarse que si $n = 2^k$ en alguna iteración $p$ aparecerán $n$ números iguales. En ese momento se detiene la iteración.

Por ejemplo, supongamos $k = 2$, $(n = 4)$ y que la circunferencia "inicial" sea $C_0 = (8, 2, 5, 7)$, entonces iteramos y obtendremos sucesivamente, $C_1 = (6, 3, 2, 1)$, $C_2 = (3, 1, 1, 5)$, $C_3 = (2, 0, 4, 2)$, $C_4 = (2, 4, 2, 0)$ y $C_5 = (2, 2, 2, 2)$, por lo que el número de circunferencias iteradas es $p = 5$. Entonces, dada una lista $L = [x_0, x_1, ..., x_{n-1}]$ de $n$ números enteros que representan los valores alrededor de la circunferencia inicial, escribir una función `int circulo(list<int>& L);` que ejecuta esta tarea y devuelva además el número de circunferencias iteradas $p$.
*Restricción:* el algoritmo debe ser *in place*.
*Ayuda:* Pensar a la lista en un *"sentido circular"*. Tener cuidado al generar la diferencia correspondiente al extremo.




**[Ej. 3]** **[color-grafo (5 ptos)]** Colorear el grafo de la figura usando el mínimo número de colores posible. Usar el algoritmo heurístico ávido. ¿La coloración obtenida es óptima? Justifique.



**[Ej. 4]** **[Preguntas (total = 15 puntos, 5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

*a*) La función

```
void swap(list<int> &L,list<int>::iterator p) {
  int x; list<int>::iterator q;
  /* ... */;  }
```

debe intercambiar el elemento en la posición `p` con el de la siguiente `q` (asumiendo que son dereferenciables). ¿Cuál de los siguientes es el fragmento faltante correcto?

- ☐ `q = p; x = *++q; L.erase(q); L.insert(p,x);`
- ☐ `x = *++p; p = L.erase(p); L.insert(p,x);`
- ☐ `x = *++p; L.erase(p); L.insert(p,x);`
- ☐ `x = *p; p = L.erase(p); L.insert(p,x);`

---

*b*) El tiempo de ejecución de la función `find(key)` para correspondencias implementadas por listas ordenadas es (en el caso promedio)... ($n$ es el número de asignaciones en la correspondencia)

- ☐ ... $O(\log n)$
- ☐ ... $O(n^2)$
- ☐ ... $O(1)$
- ☐ ... $O(n)$

---

*c*) Dadas las funciones

- $T_1(n) = 3n^3 + 3 \cdot 2^3 + 5! + 2^{10}$,
- $T_2(n) = 2n! + n^2 + 6 \cdot 2^n + 20n$,
- $T_3(n) = \log n + n^{1.5} + 20n^2 + \log_2 40$ y
- $T_4(n) = 10n^3 + n^4 + 6 \cdot 2^n + 0.0001n!$

ordenarlas de menor a mayor.

$$T_{\square} < T_{\square} < T_{\square} < T_{\square}$$