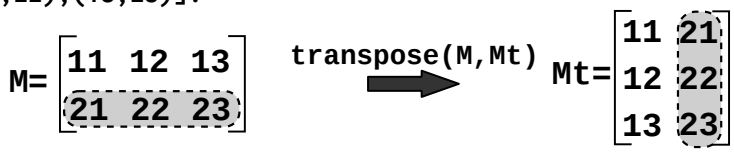
Algoritmos y Estructuras de Datos.
TPL1. Trabajo Práctico de Laboratorio. [2016-09-08]

PASSWD PARA EL ZIP: `D1G4 5IG8 MDY7`

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar**.

[Ej. 1] **[transpose (33pt)]** Sea `vector<list<int>> M` un vector de listas que almacena los coeficientes de una matriz `A` de `m*n` entradas, es decir $A \in \mathbb{R}^{m \times n}$, donde la lista `M[j]` contiene la **fila** `j-1` (el `-1` viene de la base 0 de `C++`). Escribir una función
`void transpose(vector<list<int>> &M,vector<list<int> > &Mt);` que retorna los coeficientes de la matriz transpuesta es decir la lista `Mt[j]` contiene la **columna** `j-1`.
Por ejemplo: si `M=[(11,12,13),(21,22,23)]` entonces `transpose(M,Mt)` debe dar:
`Mt=[(11,21),(12,22),(13,23)].`



Ayuda:

- Determinar el tamaño `n` de las listas de `M` (cualquiera de ellas todas deberían ser iguales).
- Resizear `Mt` a `n`.
- Recorrer cada una de las listas de `M` e ir apendizando (con `push_back()`) los elementos a la entrada correspondiente de `Mt`.

[Ej. 2] **[homogeniza (33pt)]**: Implemente una función
`void homogeniza(list<int> &C, int hmin, int hmax);`
que recibe una lista `C` de enteros ordenados en forma ascendente y la modifica de tal manera de que entre cada elemento no exista una diferencia menor a `hmin` ni mayor a `hmax`. Se deben seguir los siguientes lineamientos:

- Se recorre la lista desde el primer elemento.
- Dado un elemento `a_j`:
  - Eliminar todos los elementos siguientes hasta que el `a_{j+1}` sea `a_{j+1}>=a_j+hmin`.
  - Si `a_{j+1}>a_j+hmax` entonces insertar después de `a_j` los elementos `a_j+hmax`, `a_j+2*hmax`, `a_j+3*hmax`,... mientras sean menores que `a_{j+1}`.

Ejemplo:

```
1) recibe-> C = [0, 1, 4, 5, 10, 18], hmin=2, hmax=3
   sale-> C = [0, 3, 5, 8, 10, 13, 16, 18]

2) recibe-> C = [1, 10], hmin=2, hmax=3
   sale-> C = [1, 4, 7, 10]

3) recibe-> C = [1, 2, 3, 4, 5], hmin=2, hmax=3
   sale-> C = [1, 3, 5]
```

[Ej. 3] **[bool-opers (34pt)]**: Dadas dos listas **ordenadas** `L1` y `L2`, escribir una función `void bool_opers(list<int> &Lxor, list<int> &Land, list<int> &L1, list<int> &L2);` el algoritmo debe generar en `Lxor` una nueva lista ordenada con todos los elementos que estén en **sólo una** de las dos listas originales, y en `Land` una nueva lista ordenada con todos los elementos que estén en **ambas**.
Por ejemplo, si `L1=(1,3,5,7,9)` y `L2=(3,4,5,6,7)`, entonces el algoritmo debe generar las listas `Lxor=(1,4,6,9)`, y `Land=(3,5,7)`.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado `program.cpp`. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si `f` es la función a evaluar tenemos

  ```
  ev.eval<j>(f,vrbs);
  hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
  ```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (`eval<1>` y `evalr<1>`). La primera `ev.eval<j>(f,vrbs);` toma una serie de casos de prueba de entrada, le aplica la función del usuario `f` y compara la salida del usuario (`user`) con respecto a la esperada (`ref`). Si la verbosidad (el argumento `vrbs`) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

  ```
  m: 10, k: 3
  T(ref): (10 (7 (4 1) 1) (4 1) 1)
  T(user): (10 (7 (4 1) 1) (4 1) 1)
  EJ1|Caso0. Estado: OK
  ```

- `ucase`: Además las funciones `eval()` tienen dos parámetros adicionales:
  `Eval::eval(func_t func,int vrbs,int ucase);`
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en `ucase=-1` que quiere "hacer todos". Por ejemplo `ev.eval4(prune_to_level,1,51);` corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función `eval<j>` están almacenados en un archivo `test1.json` o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. `datain` son los datos pasados a la función y `output` la salida producida por la función de usuario. `ucase` es el número de caso.

  ```
  { "datain": {
    "T1": "( 0 (1 2) (3 4 5 6) )",
    "T2": "( 0 (2 4) (6 8 10 12) )",
    "func": "doble" },
   "output": { "retval": true },
   "ucase": 0 },
  ```

- La segunda función **`evalr<j>`** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **`seed`**, se los pasa a la función del usuario **`f()`**. Las respuestas de la **`f()`** van siendo procesadas por la función interna de hash que genera un **checksum** **`H`** de las respuestas. Por ejemplo para el primer ejercicio si **`seed=123`** entonces el checksum es **`H=523`**. Una vez que el alumno termina su tarea se le pedirá que corra la función **`evalr<j>()`** de la clase evaluadora con un valor determinado de la semilla **`seed`** y se comprobará que genere el valor correcto del checksum **`H`**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **`seed`** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **`H`**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  `void Eval::dump(list <int> &L,string s="")`: Imprime una lista de enteros por **`stdout`**. Nota: Es un método de la clase **`Eval`** es decir que hay que hacer **`Eval::dump(VX);`**. El string **`s`** es un label opcional.
  - **`void Eval::dump(list <int> &L,string s="")`**
- Después del parcial deben entregar el programa fuente (sólo el **`program.cpp`**) renombrado con su apellido y nombre (por ejemplo **`messilionel.cpp`**). Primero el apellido.