Taller 1:
Producto de matrices

El producto de $A = (a_{ij})$ y $B = (b_{ij})$, en donde A es una matriz $m \times k$ y B es una matriz $k \times n$, es la matriz $C = (c_{ij})$, en donde

$$c_{ij} = \sum_{l=1}^{k} a_{ik} b_{kj}, \quad 1 \leq i \leq m, \ 1 \leq j \leq n. \tag{1}$$

1. Cree una clase llamada `MatrixOne` que represente una matriz de enteros, y cuya interfaz se presenta a continuación:

```
1   class MatrixOne {
2   private:
3     int *array;
4     int dimx;
5     int dimy;
6
7   public:
8     MatrixOne(int m, int n);
9     ~MatrixOne();
10
11    int get_dimx() { return dimx; }
12    int get_dimy() { return dimy; }
13    int get(int x, int y) { return array[y*dimx+x]; }
14    void set(int x, int y, int val) { array[y*dimx+x] = val; }
15
16    MatrixOne& operator=(MatrixOne& M);
17    void display();
18  };
```

   El constructor de la clase debe separar la memoria a la que apunta `array` con el operador `new` e inicializar los elementos a 1. El destructor debe liberar la memoria. El operador `=` crea una nueva matriz igual a la que se pasa por argumento (copia). Finalmente, el método `display` debe imprimir la matriz en pantalla con una fila en cada renglón.

2. Defina el operador producto entre matrices

```
MatrixOne operator*(MatrixOne &M1, MatrixOne &M2);
```

que recibe como argumentos dos objetos `MatrixOne` y retorna su producto.

3. Compruebe el correcto funcionamiento del operador producto y de la clase implementando el siguiente programa

```
const int DIMX = 30;
const int DIMY = 50;

int main(int argc, char** argv) {
  int size = atoi(argv[1]);

  MatrixOne M1(DIMY, size);
  MatrixOne M2(size, DIMX);
  //M1.display();
  //M2.display();

  double tstart = gettime();
  MatrixOne R = M1*M2;
  double tstop = gettime();

  //R.display();
  printf("%d\n",R.get(0,0));
  printf("Time: %f\n", tstop-tstart);

  return 0;
}
```

en donde

```
#include <sys/time.h>
double gettime() {
  struct timeval tp;
  gettimeofday(&tp, NULL);
  return tp.tv_sec + tp.tv_usec/(double)1.0e6;
}
```

4. Implemente una versión de `operator*` que permita paralelizar el cálculo usando `thread` de C++.

5. Compruebe la correctitud de su implementación paralela comparándola con la serial. Verifique el *speedup* que logra su implementación repitiendo los cálculos seriales y paralelos n veces e imprimiendo los tiempos y sus cuadrados en un archivo de texto. Con los dato del archivo calcule el speedup promedio $\bar{S}$ y su desviación estándar σ_S.

$$\bar{S} = \frac{1}{n}\sum_{i=1}^{n} S_i$$

$$\overline{S^2} = \frac{1}{n}\sum_{i=1}^{n} S_i^2$$

$$\sigma_S = \sqrt{\overline{S^2} - \bar{S}^2}$$