Curso de Programación en C++.
TPL5-2017. Trabajo Práctico de Laboratorio. [2018-06-08]

PASSWD PARA EL ZIP: `3W25 6CWX R556`

Ejercicios

[Ej. 1] **[findsum]** Implemente la función `vector<int> findsum(vector<int>& v,int S);` que dado un vector de enteros `v` y un entero `S`, encuentre y retorne un subvector contiguo cuya suma sea `S`. Si no existe ningún subvector con dicha suma, retorne un vector vacía. En caso de haber varios vectores que cumplan retorne el primero y más corto.

Ejemplos:

```
v: [1,2,6,5,8,3,4,6] y S=13 debe retornar [2,6,5]
v: [1,2,6,5,8,3,4,6] y S=15 debe retornar [8,3,4]
v: [1,2,6,5,8,3,4,6] y S=17 debe retornar []
```

[Ej. 2] **[xinyect]** Dada un vector de conjuntos `VS_IN` de enteros, y una función `int f(int)`, copiar en otro vector de conjuntos `VS_OUT` solo aquellos conjuntos para los cuales la función `f` es inyectiva (esto es, que genera imágenes diferentes para todos los elementos del conjunto).

Ejemplo: Si `VS_IN=[ (1,2,3), (2,-2,10), (0,5) ]` y `f` es `f(x)=x^2` entonces `VS_OUT` debería quedar `[ (1,2,3), (0,5) ]`, ya que para el segundo conjunto del vector de entrada hay dos elementos (`2` y `-2`) que generan la misma imagen (`4`), razón por la cual ese conjunto se omite.

Ayuda:

- Por cada conjunto de `S` de `VS_IN`
  - Generar un conjunto auxiliar con las imágenes de la función aplicada a sus elementos
  - Comparar los tamaños de ambos conjuntos para determinar si es debe insertar `S` en `VS_OUT`.

[Ej. 3] **[getjaco]** Consideremos una clase virtual `lin_transform_t` que representa transformaciones lineales afines en un espacio de dimension `n`.

```
class lin_transform_t {
  public:
  virtual int dim()=0;
  virtual void transform(VectorXd &x,VectorXd &y)=0;
};
```

El método `dim()` retorna la dimensión del espacio, y `void transform(VectorXd &x,VectorXd &y);` retorna el resultado `y` de aplicar la transformación a un vector `x`.

Consigna 1: Escribir implementaciones `shift_t`, `ident_t`, `scale_t`, `rotate_t` de acuerdo a las siguientes características

- **shift_t(n,c)**: $y_j = x_j + c$
- **ident_t(n)**: $y_j = x_j$
- **scale_t(n,c)**: $y_j = cx_j$
- **rotate_t(n)**: $y_j = x_k$, con $k = \mathrm{mod}(j-1, n)$, es decir si **x=[3,5,2]** entonces **y=[5,2,3]**.

En todos los casos **int n** es la dimensión del espacio y c es un **double**. Estos valores deberían estar almacenados en el objeto y ser definidos en el constructor.

Consigna 2: Escribir una función **void getjaco(lin_transform_t &lt,MatrixXd &A,VectorXd &b);** que calcula en **A** y **b** el Jacobiano y término constante de la transformación **lt**; es decir **y=A*x+b**.

Para los ejemplos de la **Consigna 1** **getjaco** debe retornar:

- **shift_t**: $A = I, b = [c, c, c, c....]^T$.
- **ident_t**: $A = I, b = 0$.
- **scale_t**: $A = cI, b = 0$.
- **rotate_t**: $b = 0$,

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Por ejemplo si hacemos

```
void check1(lin_transform_t &lt) {
  cout << "================\n";
  MatrixXd A;
  VectorXd b;
  getjaco(lt,A,b);
  cout << "A: " << endl;
  cout << A << endl;
  cout << "b: " << endl;
  cout << b << endl;
}

void check_getjaco() {
  shift_t shift(10,3.0); check1(shift);
  ident_t ident(5); check1(ident);
  scale_t scale(4,8.3); check1(scale);
  rotate_t rotate(5); check1(rotate);
}
```

Debemos obtener:

```
================
A:
1 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1
b:
3
```

```
3
3
3
3
3
3
3
3
3
================
A:
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
```

```
0 0 0 1 0
0 0 0 0 1
b:
0
0
0
0
0
================
A:
8.3   0   0   0
0 8.3   0   0
0   0 8.3   0
0   0   0 8.3
b:
0
```

```
0
0
0
================
A:
0 0 0 0 1
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
b:
0
0
0
0
0
```