

Curso de Programación en C++.

TPL0. Trabajo Práctico de Laboratorio. [2019-09-11]

PASSWD PARA EL ZIP: **RFP7 MCW6 J43P**

Ejercicios

[Ej. 1] **[contained]** Dados dos arreglos de enteros $v1, v2$ determinar si $v1$ está contenido en $v2$ o viceversa:
`bool contained(vector<int> &v1, vector<int> &v2);` Por ejemplo

```
v1:[1,2,3],    v2:[1,2,3,4]    => true
v1:[5,1,2,3],  v2:[2,5]        => true
v1:[5,1,2,3],  v2:[2,5,5]      => false
v1:[1,2,3,4],  v2:[0,1,2,3]    => false
v1:[1,2,3],    v2:[3,2,1]      => true
```

Si un valor está repetido n veces en $v1$ debe estar repetido al menos n veces en $v2$ y viceversa (respetar multiplicidad). Por ejemplo en el 3er caso arriba los valores de $v2$ están contenidos en $v1$ pero no con su multiplicidad, ya que 5 está dos veces en $v2$ pero una sola vez en $v1$.

Ayuda:

- Escribir una función `bool isin(int x, vector<int> &v);` que busca el elemento x en el vector v . Si lo encuentra lo elimina de v y retorna `true`.
- Escribir una función auxiliar `bool contained2(vector<int> x, vector<int> y);` que realiza la tarea indicada pero sólo si x está contenido en y (es decir no si y está incluida en x). Para hacer el `contained2` recorre los elementos de $v1$ y se fija si están en $v2$ con `isin` y eliminándolo. Si todos los elementos de $v1$ están en $v2$ entonces retorna `true`, si no retorna `false`.
Nota: Para borrar el j -ésimo elemento de un vector debe hacer `v.erase(v.begin()+j)`.
- Escribir `contained` como un *wrapper* que llama a `contained2` con $(v1, v2)$ y $(v2, v1)$.

[Ej. 2] **[findsum]** Escribir una función `bool findsum(vector<int> &v, int sumtarget);` que dado un vector de enteros v y un entero `sumtarget` determina si existe alguna tripleta de números $(x1, x2, x3)$ en v tal que la suma de sus cubos $(x_1^3 + x_2^3 + x_3^3)$ es igual a `sumtarget`.

Atención: Los enteros $(x1, x2, x3)$ deben ser distintos.

Ejemplos:

```
sumtarget:36,    v:[1,2,3,4,5,6]    => true    36=1^3+2^3+3^3
sumtarget:37,    v:[1,2,3,4,5,6]    => false
sumtarget:36,    v:[6,5,4,3,2,1]    => true    36=3^2+2^3+1^3
sumtarget:856,   v:[6,8,1,7]        => true    856=8^3+1^3+7^3
sumtarget:293,   v:[6,5,7,4,9,2]    => false
sumtarget:91,    v:[5,7,3,0,6,4]    => true    91=3^3+0^3+4^3
```

Ayuda:

- Hacer un triple lazo sobre los enteros del vector, evitando los valores repetidos. Una posibilidad es hacer los lazos anidados sobre índices **j, k, l** de la siguiente manera
 - **j** en **[0, n-2]**,
 - **k** en **[j+1, n-1]**,
 - **l** en **[k+1, n]**.
 Si los valores (**v[j], v[k], v[l]**) satisfacen la condición retorna verdadero.
- Si ninguna tripleta satisface la condición retorna falso.
- Otra posibilidad para evitar los valores repetidos es hacer los tres lazos anidados sobre **j, k, l** evitando las repeticiones con condicionales del tipo **if (j!=k) { ... }**
- *Nota:* Para hacer el cubo de un entero conviene escribir un macro o función auxiliar. No se recomienda usar la función **pow(x, y)** ni tampoco usar el operador **x^3** ya que en C++ tiene otro significado (es el **xor** bit a bit).

[Ej. 3] [findperm] Escribir una función

bool findperm(vector<string> vs, string target); que determina si existe alguna combinación de strings de **vs** que forme el string **target**.

Ejemplos:

target=abc	vs=[c, b, a]	=> true
target=abcd	vs=[c, b, a]	=> false
target=abc	vs=[e, c, b, a]	=> true
target=brabraaca	vs=[a, bra, ca, da, bra]	=> true
target=bracabra	vs=[a, bra, ca, da, bra]	=> true
target=cadabraboom	vs=[a, bra, ca, da, bra]	=> false
target=adabrabra	vs=[a, bra, ca, da, bra]	=> true
target=caada	vs=[a, bra, ca, da, bra]	=> true

Ayuda:

- Recorremos por fuerza bruta todos las posibles combinaciones de los strings en **vs** y comprobamos si coincide con **target**. Si lo encontramos retornamos **true** caso contrario retornamos **false**.
- Para recorrer las combinaciones utilizamos una estrategia recursiva. Supongamos que tenemos una de las combinaciones **partial** de **m** de los strings, y **rest** es el resto de los strings. Por ejemplo si **vs=[a, bra, ca, da, bra]** y **partial="dabra"**, entonces **rest=[a, ca, bra]**. Podemos generar las combinaciones de 3 strings combinando **partial** con cada uno de los strings en **rest**, es decir **[dabraa, dabraca, dabrabra]**.
- Es decir, escribimos una función recursiva auxiliar
bool findperm(string partial, vector<string> rest, string target); que
 - Verifica si **partial** coincide con **target**, si es así retorna true.
 - Caso va llamando recursivamente a **findperm** con la concatenación de **partial** con **rest[j]** y pasando el vector **rest** con **rest[j]** eliminado. Si en algún caso retorna **true** entonces **findperm** retorna true. Si se recorren todos los **rest[j]** y ninguna retorna true, entonces retorna **false**.

```
findperm(partial=dabra,rest=[a,ca,bra])  
-> findperm(partial=dabra+a, rest=[ca,bra])  
-> findperm(partial=dabra+ca ,rest=[a,bra])  
-> findperm(partial=dabra+bra,rest=[a,ca])
```