

Curso de Programación en C++.

TPL1-2019. Trabajo Práctico de Laboratorio. [2019-10-03]

PASSWD PARA EL ZIP: **VCP5 QRF6 R6QJ**

Ejercicios

[Ej. 1] **[int-set]** Queremos representar subconjuntos de los números enteros a través de una clase polimórfica de la forma:

```
class int_set_t {  
    public:  
    virtual bool isin(int x)=0;  
};
```

El método **predicado** `s.isin(x)` indica si `x` pertenece al conjunto o no. Notar que de esta forma se pueden representar conjuntos infinitos, por ejemplo los números impares se pueden representar de la forma

```
class odd_t : public int_set_t {  
    public:  
    bool isin(int x) {  
        return modulo(x,2)==1;  
    }  
};
```

donde `modulo(x,2)` es el resto de dividir `x` por 2 (como el operador `%` pero siempre retorna 0/1).

Consigna: Instanciar las siguientes clases derivadas de `int_set_t`:

```
class even_t : public int_set_t { ... };  
class all_t : public int_set_t { ... };  
class empty_t : public int_set_t { ... };  
class positive_t : public int_set_t { ... }  
class stride_t : public int_set_t {  
    int start,end,step;  
    ...  
};  
class vector_t : public int_set_t {  
    vector<int> w;  
    ...  
};  
class intersection_t : public int_set_t {  
    unique_ptr<int_set_t> s1,s2;
```

```

...
};
class union_t : public int_set_t {
    unique_ptr<int_set_t> s1,s2;
    ...
};
class difference_t : public int_set_t {
    unique_ptr<int_set_t> s1,s2;
    ...
};
class complement_t : public int_set_t {
    unique_ptr<int_set_t> s1;
    ...
};

```

- **stride_t**, dados los valores **start**, **end**, **step** debe retornar los elementos $k = \text{start} + j \cdot \text{step}$ tales que $k \geq \text{start}$ && $k < \text{end}$, por ejemplo si **start**=10, **step**=3, **end**=18 los valores son [10, 13, 16].
- **vector_t** simplemente contiene un **vector<int>** **w** que contiene los elementos aceptables. En este caso la función **isin(x)** simplemente debe buscar el elemento **x** en **w**.
- **intersection_t** contiene dos punteros a subobjetos **s1**, **s2** de la clase **int_set_t**. Esta clase debe hacer la intersección de los subobjetos. Los punteros son guardados en **unique_ptr<int_set_t> s1,s2** recordar que los punteros subyacentes se pueden obtener haciendo **s1.get()**.
- Las clases **union_t** y **difference_t** también hacen las correspondientes operaciones de conjuntos y son binarias (contienen dos subobjetos **s1** y **s2**).
- La clase **complement_t** corresponde al complemento del conjunto **s1** es decir todos los elementos del conjunto universal que no está en **s1**.
- Todos los campos dato (**w**, **s1**, **s2**, **start**, **step**, **end**) deben ser públicos.
- Al definir una dada clase **XYZ** hay que activar su evaluación definiendo el macro correspondiente **HAS_XYZ** por ejemplo

```

class positive_t : public int_set_t { ... }
#define HAS_POSITIVE_T

```

[Ej. 2] [nilpot] Queremos representar permutaciones de los elementos de un vector de enteros. Por ejemplo la permutación de tipo *shift* desplaza los elementos de un vector una posición hacia el final **shift1([0,1,2,3,4]) => [4,0,1,2,3]**. Estas operaciones las representamos a través de una clase virtual

```

class perm_t {
public:
    virtual void mix(vector<int> &w) { }
};

```

El método virtual **mix(w)** debe retornar la permutación correspondiente en el vector de enteros **w**.

Consigna 1: Implementar las siguientes clases

```
class shiftm_t : public perm_t {
    int m;
    ...
};
class oddeven_t : public perm_t { };
class reflex_t : public perm_t { };
class shift2_t : public perm_t {
    int m1,m2;
    ...
};
```

- **shiftm_t** hace un shift pero de salto **m** por ejemplo si **m=2**
shiftm([0,1,2,3,4]) => [3,4,0,1,2].
- **oddeven_t** intercambia las posiciones sucesivas: 0 con 1, 2 con 3, $2*j$ con $2*j+1$. Por ejemplo
oddeven([0,1,2,3,4]) => [1,0,3,2,4]. (Nota: Si el número de elementos es impar el último queda en su lugar.)
- **reflex_t** hace una reflexión con respecto a la posición media. Por ejemplo si
reflex([0,1,2,3,4,5]) => [5,4,3,2,1,0]. (Nota: si el número de elementos es impar el del centro queda en su lugar.)
- **shiftm1m2_t**: Asumiendo que $m1+m2 \leq n$ (donde **n** es la longitud del vector) hace un *shift* de una posición en los primeros **m1** posiciones y un shift de una posición en los siguientes **m2**. Si $m1+m2 < n$ entonces los elementos restantes quedan en su posición.
Por ejemplo: Si **m1=3, m2=2** entonces **shift2([0,1,2,3,4,5,6,7,8,9])**
=>[2,0,1,4,3,5,6,7,8,9]

Nota: Después de implementar las clases deben activar su evaluación definiendo el macro correspondiente **HAS_XYZ** por ejemplo

```
class reflex_t : public perm_t { ... };
#define HAS_REFLEX_T
```

Nilpotencia: Puede verse que una permutación es una operación *nilpotente*, es decir si es aplicada repetidamente, existe un cierto entero $np \geq 1$ para el cual la aplicación de **np** veces es equivalente a la identidad. Por ejemplo: la permutación **oddeven** es de nilpotencia 2, y el **shift** de una posición para **N** objetos es de nilpotencia **N**. La identidad tiene nilpotencia 1.

Consigna 2: Implementar una función **int nilpot(perm_t &perm, int N)**; que dada una permutación **perm** y una longitud del vector **N** retorna su nilpotencia.

Ayuda: Construir un vector de longitud **N** inicialmente con la identidad **[0,1,2,...,N-1]** y aplicar la permutación repetidamente hasta que el vector vuelva a ser la identidad.