1151044 - Programación Orientada a Objetos
Tarea 2 - 2026-I

Implementa un tipo de dato `conjunto` que modele un conjunto que puede contener los enteros del 0 al 99.

```
class conjunto {
   //...
public:
   //...
};
```

Las operaciones disponibles deben ser las siguientes.

- Inicialización por defecto.
  El conjunto comienza vacío.

```
conjunto c;
```

- Función miembro `bool inserta(int v);`
  Inserta el valor `v` al conjunto. Devuelve verdadero si el elemento se insertó exitosamente. Una inserción falla si el valor ya existía en el conjunto o si el valor que se solicitó insertar está fuera de rango.

```
bool b = c.inserta(3);
```

- Función miembro `bool elimina(int v);`
  Elimina el valor `v` del conjunto. Devuelve verdadero si el elemento se eliminó exitosamente. Una eliminación falla si el valor que se solicitó eliminar no existía en el conjunto.

```
bool b = c.elimina(4);
```

- Función miembro `int size( ) const;`
  Devuelve la cantidad de elementos del conjunto.

```
int tam = c.size( );
```

- Función miembro `int operator[](int i) const;`
  Devuelve una copia del valor del i-ésimo elemento del conjunto. Los elementos del conjunto se consideran implícitamente ordenados de menor a mayor y reciben índices a partir de cero. Es decir, el elemento más pequeño del conjunto es `c[0]`, el siguiente elemento más pequeño es `c[1]`, etcétera. Puedes suponer que el índice `i` es válido.

```
int v = c[0];
```

- Funciones **no miembro** `operator==` y `operator!=` (comparación de conjuntos)[1].
  Determinan si dos conjuntos son iguales o distintos, respectivamente. Dos conjuntos son iguales si y sólo si tienen el mismo tamaño y también contienen los mismos elementos. Debe permitirse comparar conjuntos que sean `const`.

```
conjunto c, d;
//...
bool b1 = (c == d);
bool b2 = (c != d);
```

- Copia y asignación[2].
  Se copia el contenido de un conjunto en otro. Si el conjunto receptor ya existía, su contenido se sobreescribe.

```
conjunto c, d;
c.inserta(5);
d.inserta(7);
conjunto e = c;
e = d;
```

[1] Si bien el requerimiento de que estas funciones sean no miembro es irrelevante en cuanto a implementar la semántica solicitada, puede ser difícil programarlas de forma eficiente si no tienen acceso a la representación interna del tipo y el `operator[]` tampoco es eficiente.

[2] Dependiendo de cómo se implemente el tipo, puede ser o no ser necesario redefinir el constructor por copia y el operador de asignación.

El tipo de dato debe implementarse de modo de que sea imposible que un conjunto adquiera un estado inválido. No puedes usar la biblioteca de C/C++ para implementarlo. Tu código no debe declarar variables estáticas ni globales. Un programa que manipule a lo mucho cien conjuntos y que haga a lo mucho mil llamadas a función debe terminar en menos de un segundo. Puedes consultar una página de prueba en https://racc.mx/uam/trimestre-actual/2026-i/poo/tarea2.html. El código que envíes no debe contener `main` y no debe leer ni imprimir nada directamente.

Envía tu código fuente desde tu cuenta institucional al formulario en https://forms.gle/hKTWWHzwj873dGN38. Tu código será evaluado con varios casos de prueba y se espera que cumpla la semántica descrita.

Ejemplo de uso

```
int main( ) {
   conjunto c;
   bool b1 = c.inserta(3);
   bool b2 = c.inserta(1);
   bool b3 = c.inserta(3);
   std::cout << b1 << " "
             << b2 << " "
             << b3 << "\n";

   std::cout << c.size( ) << "\n";
   for (int i = 0; i < c.size( ); ++i) {
      std::cout << c[i] << " ";
   }
   std::cout << "\n";

   conjunto d = c;
   std::cout << (c == d) << "\n";
}
```

Ejemplo de salida

```
1 1 0
2
1 3
1
```