Uso de la función sort para ordenar un vector

La función `sort` está definida en la librería `<algorithm>.`

Existen dos formas para esta función (http://www.cplusplus.com/reference/algorithm/sort/):

1. **Primera forma:**

```
template <class RandomAccessIterator>
  void sort (RandomAccessIterator first, RandomAccessIterator
last);
```

La llamada a la función se hace con dos parámetros:

```
vector<tTipo> v; ......
sort (v.begin(), v.end());
```

La expresión `v.begin()` y `v.end(),` se refiere a la llamada de los métodos `begin()` y `end()` del vector. Estas funciones devuelven un valor de tipo `RandomAccessIterator` refiriendose al primer elemento y al último del vector.

Si el vector está definido con un tamaño máximo, ordena los elementos hasta el tamaño máximo, aunque no estén todas las componentes utilizadas.

Para ordenar utiliza el operador <. Si este operador se encuentra sobrecargado para el vector, entonces ordena según el método sobrecargado.

2. Segunda forma:

```
template <class RandomAccessIterator, class Compare>
  void sort (RandomAccessIterator first, RandomAccessIterator
last, Compare comp);
```

La llamada a la función se hace con tres parámetros:

```
vector<tTipo> v; ......
sort (v.begin(), v.end(), comp);
```

o

```
vector<tTipo> v; ......
sort (v.begin(), v.end(), comparador());
```

En el primer caso `comp` se refiere a una función que debe estar implementada. La función debe devolver un valor `bool` y recibir dos parámetros de tipo `tTipo`.

```
bool comp(const tTipo& p1, const tTipo & p2) {
     if ......
}
```

En el segundo caso `comparador` es una clase (tipo de datos que se verá en segundo curso). La clase comparador para este tipo de aplicaciones se implementa como:

```
class comparador {
  public:
     bool operator()(const tTipo &p1, const tTipo& p2){
          if....
     }
```

En ambas funciones el orden es estricto, si hay elementos iguales no se garantiza en que orden quedan. Por ejemplo si se ordenan valores de tipo `struct` por un solo campo, no se garantiza que los valores de igual clave queden en el orden original. (orden estable).

3. Arrays definidos con corchetes:

Para los arrays definidos como:

```
int v[5];
```

No están definidas las funciones `begin()` y `end()`. La función de ordenar se llama como:

```
std::sort(v,v+4);
```

Tiene que haber una función de comparación definida sobre el tipo de las componentes del vector. Si el tipo es básico la función está definida por defecto, si el tipo es enumerado debe definirse el operador <.

Puede utilizarse el tercer argumento con funciones de comparación o clases comparadoras igual que en el caso anterior.