5 Miembros static y STL en C++

Elaborado por: Ukranio Coronilla

En ocasiones se requiere comunicar información entre varios objetos de la misma clase, por ejemplo si se desea saber cuántas veces diversos objetos de la misma clase, han mandado llamar una función. En ese caso tendríamos que declarar una variable que lleve la cuenta y sea compartida por todos los objetos. Este tipo de variables se conocen como estáticas y deben declararse anteponiendo la palabra `static` (línea 11) así como inicializarlas fuera de la clase (línea 19). La siguiente es una muestra de su uso, donde se lleva la cuenta del número de veces que los objetos `Fecha` mandan llamar a la función `muestraFecha`. Compile y ejecute el código para probar su correcto funcionamiento.

Programa 5-1

```
1  #include <stdlib.h>
2  #include <iostream>
3  using namespace std;
4
5  class Fecha
6  {
7  private:
8     int dia;
9     int mes;
10    int anio;
11    static int vecesMuestF;
12 public:
13    static int getVecesMuestF();
14    Fecha(int = 3, int = 4, int = 2014);
15    void inicializaFecha(int, int, int);
16    void muestraFecha();
17 };
18
19 int Fecha::vecesMuestF = 0;
20
21 int Fecha::getVecesMuestF()
22 {
23    return vecesMuestF;
24 }
25
26 Fecha::Fecha(int dd, int mm, int aaaa)
27 {
28    mes = mm;
29    dia = dd;
30    anio = aaaa;
31 }
32
33 void Fecha::inicializaFecha(int dd, int mm, int aaaa)
34 {
35    anio = aaaa;
36    mes = mm;
37    dia = dd;
38    return;
39 }
40
41 void Fecha::muestraFecha()
```

```
42 {
43    cout << "La fecha es(dia-mes-año): " << dia << "-" << mes << "-" << anio << endl;
44    vecesMuestF++;
45    return;
46 }
47
48 int main()
49 {
50    Fecha a, b, c(21, 9, 2000);
51    b.inicializaFecha(17, 6 , 2000);
52    a.muestraFecha();
53    cout << "Se ha mandado llamar muestraFecha() " << a.getVecesMuestF() << " veces\n";
54    b.muestraFecha();
55    cout << "Se ha mandado llamar muestraFecha() " << b.getVecesMuestF() << " veces\n";
56    c.muestraFecha();
57    cout << "Se ha mandado llamar muestraFecha() " << c.getVecesMuestF() << " veces\n";
58
59    return 0;
60 }
```

Por otra parte si se desea hacer funciones que no van a acceder a ninguna variable miembro de la clase, o solo a variables miembro estaticas, se utilizan las funciones estáticas. En este caso solo es necesario anteponer la palabra static al nombre de la función como se muestra en la línea 13. En la línea 21 al implementar la función `getVecesMuestF` es importante recordar que solo podrá acceder a variables miembro `static`.

Observe también que en la línea 44 se incrementa la bandera cada que se manda llamar a la función `muestraFecha`.

Ejercicio 1: Intente acceder a todas las variables miembro desde la función `getVecesMuestF`.

Introducción a STL

En C++ los vectores pueden ser vistos como arreglos que pueden crecer o decrecer en tamaño de manera dinámica durante la ejecución de un programa. Los vectores se forman a partir de una clase genérica llamada **Template**, la cual puede trabajar con muchos tipos de datos. Los Templates son una de las características más importantes de C++ y existe una librería completa de Templates conocida como Standard Template Library (STL).

Suponga que se requiere declarar una variable llamada `var`, para almacenar un vector de enteros, entonces escribimos:

```
vector<int> v;
```

La parte *vector<Tipo>* se conoce como clase template, y *Tipo* puede ser cualquiera, inclusive otra clase. La instrucción anterior manda llamar de manera automática al constructor y crea un vector sin elementos. Al igual que con los arreglos, es posible inicializar e imprimir respectivamente el elemento i-esimo del vector v mediante:

```
v[i] = 2014;
cout << v[i];
```

Igual que sucede con un arreglo, el vector debe tener el tamaño suficiente para poder hacer una asignación. Para añadir un elemento al vector se usa la función `push_back`. Por ejemplo para añadir e inicializar los elementos 0, 1 y 2 haríamos:

```
v.push_back(313);
v.push_back(54);
v.push_back(98);
```

Para utilizar este template se deben incluir al inicio las líneas:

```
#include <vector>
using namespace std;
```

Se puede declarar un vector con 15 elementos mediante:

```
vector <int> v(15);
```

El template vector tiene además de `push_back()` una gran cantidad de métodos, por ejemplo `size()` el cual devuelve el tamaño del vector. Para mayor información véase:

http://en.wikipedia.org/wiki/Standard_Template_Library

Ejercicio 2: Basándose en el programa 4-1, elabore la clase `PoligonoIrreg` la cual representa un polígono irregular en los cuatro cuadrantes del plano cartesiano, cuyos vértices se componen de un vector de objetos `Coordenada`. Se debe disponer de los métodos `anadeVertice` para añadir un vértice al polígono, e `imprimeVertices` para imprimir el conjunto de vértices que componen al polígono.

En cualquier momento un vector tiene una capacidad, la cual es el número de elementos para el cual el sistema operativo ha reservado memoria. La capacidad se obtiene con la función `capacity()` y no necesariamente coincide con la que devuelve `size()`. Cuando se añade un nuevo elemento al vector, automáticamente se incrementa la capacidad, normalmente reservando memoria al doble del tamaño actual. Este proceso de incremento lleva tiempo, de modo que si la eficiencia es importante, conviene mejor incrementar un bloque grande a varios bloques pequeños. La función `reserve` tiene esta facultad, por ejemplo para ajustar la capacidad a 100 elementos del vector v se hace:

```
v.reserve(100);
```

Con la siguiente instrucción aumentamos la capacidad en 20 elementos a los que tiene actualmente:

```
v.reserve(v.size( ) + 20);
```

Es posible también cambiar el tamaño de un vector utilizando la función `resize`, por ejemplo para reajustar el tamaño del vector a 30 elementos:

```
v.resize(30);
```

Si el tamaño del vector era menor a 30, este se incrementa, pero si era mayor entonces se pierden los últimos elementos para ajustarse a 30.

Ejercicio 3: Para verificar el desempeño al usar la función reserve, construya un vector de n objetos `PoligonoIrreg`, donde cada objeto tiene un número aleatorio de hasta m vértices. Agregue un miembro `static` para almacenar el número de vértices creados por todos los objetos `PoligonoIrreg`.

En un caso almacene elementos al vector solo con `push_back()`, y en otro utilice el método `reserve`. Ambos programas deben probarse con el comando time. Observe que no tiene que asignarse una semilla a la función `rand()` o de lo contrario la comparativa sería incorrecta.