Clases en C++

Elaborado por: Ukranio Coronilla

Una clase es muy similar a una estructura en C, solo que además de las variables miembro, también tiene funciones miembro o métodos. A una variable del tipo clase se le da el nombre de objeto.

Un ejemplo de su implementación se muestra en el programa 2-1, el cual utiliza la clase Fecha para instanciar tres variables del tipo Fecha mejor conocidas como objetos, los inicializa de distintas maneras y los imprime. Compile y ejecute el código para probar su correcto funcionamiento.

Programa 2-1

```
1  #include <iostream>
2  using namespace std;
3
4  class Fecha
5  {
6  private:
7     int dia;
8     int mes;
9     int anio;
10 public:
11    Fecha(int = 3, int = 4, int = 2014);
12    void inicializaFecha(int, int, int);
13    void muestraFecha();
14 };
15
16 Fecha::Fecha(int dd, int mm, int aaaa)
17 {
18    mes = mm;
19    dia = dd;
20    anio = aaaa;
21 }
22
23 void Fecha::inicializaFecha(int dd, int mm, int aaaa)
24 {
25    anio = aaaa;
26    mes = mm;
27    dia = dd;
28    return;
29 }
30
31 void Fecha::muestraFecha()
32 {
33    cout << "La fecha es(dia-mes-año): " << dia << "-" << mes << "-" << anio << endl;
34    return;
35 }
36
37 int main()
38 {
39    Fecha a, b, c(21, 9, 1973);
40
41    b.inicializaFecha(17, 6 , 2000);
```

```
42    a.muestraFecha();
43    b.muestraFecha();
44    c.muestraFecha();
45 }
```

Descripción del funcionamiento

De la línea 4 a la 14 se define la clase Fecha en un formato muy parecido a una estructura, en las líneas 11, 12 y 13 se tienen prototipos de funciones mejor conocidos como métodos, los cuales son parte de la clase. El prototipo en la línea 11 se conoce como constructor porque tiene el mismo nombre que la clase, y se manda llamar en el momento que se crea un objeto de esta clase. En este caso el constructor recibe tres parámetros y provoca que se inicialicen las variables miembro `dia`, `mes` y `anio` con los valores 3, 4 y 2014 respectivamente.

En las líneas 16, 23 y 31 se definen las funciones o métodos, sin embargo note que antes del nombre de la función se debe indicar el nombre de la clase; esto debido a que puede haber más de una clase con el mismo nombre para el método miembro. El operador "`::`" que sigue al nombre de la clase se conoce como operador de resolución de alcance e indica el sentido de pertenencia del método a la clase.

De la misma forma que se accede a una variable miembro en una estructura, en una clase es posible mandar a llamar un método de la clase con el operador punto ".", tal y como se muestra en las líneas 41 a 44. La línea 39 simplemente instancia tres objetos de la clase Fecha, el primero y el segundo automáticamente toman los valores que se especifican en el constructor de la línea 11. En el tercer objeto se proporcionan los valores iniciales, de modo que no se inicializarán los datos por defecto de la línea 11.

La palabra `public:` en la línea 10 indica que todos los miembros de las subsiguientes líneas de código, y dentro de la clase, pueden ser accedidos o mandados a llamar con el operador punto, ya sean estos variables o funciones. Lo interesante en una clase es la sección que abarca la palabra `private:` en la línea 6; la cual indica que todas los miembros de las subsiguientes líneas de código, y antes de la palabra `public:`, solo podrán ser accedidos por funciones que sean miembros de la clase. Observe que el acceso a los datos privados, sea para lectura o para escritura, se da dentro de cualquier método, sin necesidad del operador punto. En general en una clase todos los **datos** miembro se declararán como privados. También es común que exista un método de nombre ***get*** que devuelve el valor de dichos datos, así como el método ***set*** para modificar los datos.

Ejercicio 1: Intente acceder dentro de la función principal `main` a los datos privados, ya sea para leer y/o escribir e indique lo que sucede.

Una pregunta importante es ¿por qué se necesitan miembros privados en una clase?. La respuesta tiene que ver básicamente con facilitar la construcción de programas grandes.

Un programa es más sencillo de construir, si está compuesto de abstracciones que modelen la realidad en la cual se sitúa el problema que se quiere resolver. Una clase es una abstracción de un componente de la realidad, y está compuesta por datos y operaciones que se pueden ejecutar sobre los datos. Para dar consistencia a una clase, es más seguro que no se pueda acceder a sus datos, más que a través de funciones miembro de la clase. Esto garantiza la consistencia puesto que no es posible modificarlos desde cualquier lugar, haciendo un código más limpio y menos propenso a errores. A esto se le conoce como **ocultamiento de la información** o **encapsulamiento** y es uno de los conceptos clave en la programación orientada a objetos POO.

Es común que un programador elabore una clase, para que esta pueda ser utilizada en otros programas. Esta característica se conoce como reutilización de código, y para conseguirla es necesario que la clase mantenga bien separada la **implementación** de la **interfaz**. La interfaz es el conjunto de funciones o métodos públicos que ofrece la clase, mientras que la implementación es el código que conforma los métodos miembros de la clase. De esta manera solo es necesario que otros programadores solo tengan la interfaz, mientras que la implementación podrían desconocerla y aun así reutilizar el código de la clase. En lenguaje C es común guardar la interfaz en los archivos de encabezado como `stdio.h` .

Ejercicio 2: Para el correcto encapsulamiento de la clase Fecha en el programa 2-1, separaremos la interfaz de la implementación. En este caso la interfaz de la clase Fecha la guardaremos en el archivo `Fecha.h` y sería la siguiente:

```
#ifndef FECHA_H_
#define FECHA_H_

class Fecha
{
private:
   int dia;
   int mes;
   int anio;
public:
   Fecha(int = 3, int = 4, int = 2014);
   void inicializaFecha(int, int, int);
   void muestraFecha();
};

#endif
```

La implementación quedará en el archivo Fecha.cpp y consiste en lo siguiente:

```
#include "Fecha.h"
#include <iostream>

using namespace std;

Fecha::Fecha(int dd, int mm, int aaaa)
{
   mes = mm;
   dia = dd;
   anio = aaaa;
```

```
}

void Fecha::inicializaFecha(int dd, int mm, int aaaa)
{
   anio = aaaa;
   mes = mm;
   dia = dd;
   return;
}

void Fecha::muestraFecha()
{
   cout << "La fecha es(dia-mes-año): " << dia << "-" << mes << "-" << anio << endl;
   return;
}
```

Analice el programa 2-1 y escriba la función principal que haga uso de esta clase en el archivo `programa2_2.cpp`. En este caso para que el compilador reconozca la clase Fecha, es imprescindible como en C que la primera línea de código sea

```
#include "Fecha.h"
```

y que los tres archivos se encuentren en la misma carpeta(las comillas hacen que el compilador busque el archivo en la misma carpeta). Para compilar y enlazar tenemos dos opciones.

Opción 1:

```
g++ Fecha.cpp -c
g++ programa2_2.cpp Fecha.o -o programa2_2
```

Opción 2:

```
g++ Fecha.cpp programa2_2.cpp -o programa2_2
```

Con la opción uno se compila de manera separada los archivos de implementación y código principal. Pruebe a eliminar en `Fecha.cpp` las líneas 1 y/o 2 para observar que sucede al intentar compilar.

En todos los proyectos y códigos subsiguientes será obligatorio separar en distintos archivos, los códigos de implementación e interfaz.

Ejercicio 3: Añada un método llamado `convierte()`, que devuelva un entero resultado de calcular `anio*10000+mes*100+dia` , por ejemplo si la fecha es 1/Abril/2014, el valor devuelto será 20140401. Muestre que funciona en el código principal.

Ejercicio 4: Añada un método llamado `leapyr()`, que devuelva un `true` (tipo `bool`) si el año es bisiesto y un `false` si no es bisiesto. Un año es bisiesto si es divisible por 4 pero no por 100, con la excepción de que todos los años divisibles por 400 son bisiestos.

Proyecto 1 para subir al MOODLE (continúa):

4.- Defina una clase Temperatura que almacene de manera interna un valor de temperatura en grados Kelvin. También elabore las funciones `setTempKelvin`, `setTempFahrenheit` y `setTempCelsius` que tomen la temperatura en la escala especificada por el usuario y la almacenen en grados Kelvin. Asimismo escriba las funciones que impriman el valor almacenado en la clase y en los grados Centigrados, Kelvin o Fahrenheit, según lo solicite el usuario. Escriba una función principal para probar la clase.

5.- Defina una clase llamada `Fraccion`, la cual almacena un numerador y un denominador enteros. Debe incluir una función para que el usuario pueda inicializar a ambos, así como una función que devuelva el resultado de la división entre el numerador y el denominador como un `double`. Por último debe existir una función que devuelva a la fracción en su mínima expresión. Por ejemplo la fracción 18/15 deberá imprimirse como 6/5. Pruebe la clase con el código mínimo en el programa principal.