Universidad de Jaén
Departamento de Informática

v 1.0

Programación Orientada a Objetos

http://tiny.cc/poouja

Curso 2022/2023

José Ramón Balsas Almagro
Víctor Manuel Rivas Santos
Ángel Luis García Fernández
Juan Ruiz de Miras

Práctica 3. Clases y Objetos (2)

Índice

Objetivos

- Continuar practicando la creación y destrucción de nuevos objetos
- Utilizar objetos en memoria dinámica usando tanto punteros individuales como vectores de punteros.
- Realizar sobrecarga de operadores.

Contexto de la práctica

El contexto es exactamente el mismo que el de la práctica 2, al ser esta la segunda sesión en la que vamos a seguir trabajando en el programa del *DJ Segismundo Barcetres Rodríguez.*

Material de partida

- Puedes usar el proyecto que realizaste en la práctica anterior. Cópialo a un nuevo proyecto antes de hacer ninguna modificación y podrás conservar los dos.
- Si lo deseas, también puedes descargarte el código que hemos creado en GitHub o

incluso hacerle un fork y empezar a trabajar a partir de él. Lo puedes localizar en http://bit.ly/poouja1617pr3

Ejercicios

1. Añadir a la clase *Temazo* los siguientes atributos:
- *nombreUltimoGarito*: tipo string
- *fechaUltimoUso*: de tipo *Fecha*.

2. Modificar los constructores de la clase *Temazo* para tener en cuenta los nuevos atributos.

3. Añadir los setter y getter correspondientes a los nuevos atributos.

4. Modificar la función *mostrarTemazo* del módulo *djutils*, para que muestre también los nuevos atributos utilizando las otras funciones del módulo.

5. Añadir a la clase *Fecha* los operadores < y ==.

6. Añadir a la clase *Fecha* los operadores <=, >=, >, != .
- Opcionalmente… intenta definir estos operadores utilizando solo llamadas a los dos definidos en el ejercicio 5, es decir a < y a ==

7. Crear una función *main* en el fichero *main.cpp* en la cual:
a. Se cree un vector de 20 temazos.
b. Se cree un puntero a objeto de tipo Garito, llamado *pMiCasa*; usarlo para crear un objeto de clase Garito con los datos de la casa del DJ.
c. Se cree un vector de 10 punteros a garitos y, a continuación, se creen objetos para los 2 primeros punteros.
d. Para los 3 primeros temazos del vector creado en el apartado a), que se modifiquen los datos de nombre, intérprete, duración en segundos, fecha de último uso y nombre del último garito, de acuerdo con lo siguiente:
    - Para el primer temazo, el último garito en que se ha usado debe ser la casa del DJ.
    - Para los otros dos, el último garito en que se han usado puede ser cualquiera de los que has instanciado en el vector.
    - La fecha de último uso del primer temazo debe ser la de tu cumpleaños; la del segundo temazo el día en que el Real Madrid ganó su última *champions*; y la del tercero la del día que lo hizo el Barcelona.
e. Implementar una función *mostrarTemazosAnteriores (Temazo[], int, Fecha)* que visualice los temazos del vector cuya fecha de último uso sea anterior a la fecha indicada como tercer parámetro. Utilizarla para mostrar los temazos usados por última vez antes del 1 de enero de 2016.

Contenidos

Declaración y construcción de objetos en memoria dinámica

La construcción de objetos en memoria dinámica no se realiza en el momento en que definimos la variable (como ocurre con los objetos creados de forma automática), sino cuando usamos la instrucción *new*.

Veamos el siguiente ejemplo:

```
/**-------------------------------------
  *   @file Plato.h
  *-------------------------------------*/
const float IVA=21/100;

class Plato
{  private:
      string _nombre = "Sin nombre";
      float _precio = 0;
   public:
      Plato() = default;
      Plato ( string nombre, float precio );
      string getNombre ();
      float getPrecio ();
   private:
      void aplicaDescuento ( float descuento );
};
```

```
/**-------------------------------------
  *   @file main.cpp
  *-------------------------------------*/
int main()
{  Plato entremes ( "Ensalada tropical", 4.50 );
   Plato *postre = nullptr;
}
```

En el código anterior, al declarar la variable (el objeto de clase *Plato*) *entremes* se llama al constructor de la clase *Plato*. Sin embargo, la segunda variable, *postre*, no es un objeto de tipo *Plato*, sino un puntero a un objeto de dicho tipo. Si realmente queremos que se cree el objeto que va a almacenar el postre habremos de usar la instrucción *new*:

```
postre = new Plato ( "Flan con nata", 2.75);
```

Destrucción de objetos

Al igual que el resto de variables del programa, los objetos automáticos dejan libre la porción de memoria que ocupan cuando finaliza el ámbito para el cual han sido definidos. Igual que el resto de variables, si los objetos se han almacenado en memoria estática (es decir, son objetos creados automáticamente) no es necesario hacer ninguna acción especial para que se libere la memoria que se les ha asignado. Sin embargo, si se han almacenado en memoria dinámica (mediante el uso de punteros) será necesario indicar al sistema operativo que libere la memoria en la que estaba almacenado el objeto. Para ello se utiliza la instrucción *delete*, como en el siguiente ejemplo:

```
delete postre;
postre = nullptr; // Conveniente siempre que hacemos delete
```

El destructor es un método que se ejecuta normalmente de forma automática (aunque se puede también llamar de forma expresa) cuando la memoria de un objeto se va a liberar, ya sea porque el programador lo decide expresamente o porque el sistema libera los recursos ocupados por el objeto de forma automática (p.e. al terminar el bloque donde está declarado una variable que contiene un objeto automático).

Si una clase no define expresamente un destructor en su implementación, cuando la memoria del objeto vaya a liberarse se ejecutará un destructor por defecto que no hará nada.

En el caso de que tengamos un vector de objetos, hay que tener en cuenta que se llamará a todos los destructores de los objetos almacenados antes de liberar su memoria.

Si fuera necesario implementar el destructor de una clase, normalmente este se encargará de liberar recursos que tenga asignados el objeto que se va a liberar, por ejemplo archivos abiertos, conexiones de red, memoria dinámica reservada, etc.

Paso de mensajes con objetos en memoria dinámica

En prácticas anteriores hemos tratado el paso de mensajes a objetos automáticos. Utilizamos el signo . para indicar el método que deseamos que el objeto ejecute, como en el siguiente caso:

```
// Ejemplo de paso de mensaje a objeto AUTOMÁTICO, no dinámico
cout << "Para comenzar " << entremes.getNombre() << endl;
```

Por el contrario, si los objetos están ocupando memoria dinámica usaremos el signo -> , como en el siguiente ejemplo:

```
// Ejemplo de paso de mensaje a objeto DINÁMICO: primera forma
cout << " Y finalmente " << postre->getNombre() << endl;
```

No obstante, también puede pasarse un mensaje a un objeto dinámico con el signo . si anteponemos el signo * al nombre de la variable. Así, el siguiente ejemplo es equivalente al anterior:

```
// Ejemplo de paso de mensaje a objeto DINÁMICO: segunda forma
cout << " Y finalmente " << (*postre).getNombre() << endl;
```

Vectores de objetos

Como ya sabemos, el constructor por defecto de un objeto es aquel que no requiere parámetros actuales, ya sea porque no los tiene o porque sus valores se pueden omitir y se asignan por defecto. El constructor por defecto existe únicamente en los siguientes casos: si no hay definido ningún otro constructor en la clase, si se indica explícitamente el uso de su implementación por defecto (*=default* en su cabecera) o bien si se ha implementado expresamente. Si este aspecto ya es importante cuando declaramos un objeto de una clase, cobra especial importancia en el caso de que estemos definiendo vectores de objetos de dicha clase.

En C++ es posible declarar un vector de objetos de una clase tal y como se declara cualquier vector de un tipo simple:

```
   Fecha diasFestivos2019[100];
```

Es muy importante observar que cuando se declaran vectores de objetos se deben crear e inicializar las instancias de cada uno de los objetos que se almacenan en cada posición del vector (no olvidemos que un vector no es ni más ni menos que un grupo de variables del mismo tipo a las que se accede utilizando un índice). Por lo tanto, el sistema debe llamar a algún constructor de la clase de los objetos para inicializar cada uno de ellos. Por este motivo, es obligatorio que aquellas clases para las que se vayan a definir vectores de objetos cuenten con un constructor por defecto. De hecho, como es el único constructor que se puede utilizar en la definición del vector, se generaría un error en tiempo de compilación si el compilador no pudiera encontrarlo.

En lo referente a la memoria que utilizan los vectores de objetos, debemos tener en cuenta las mismas consideraciones que para variables normales. Es decir, para cualquier variable (o vector) declarado en una función o bloque de código en general, la memoria se reserva en la pila del programa en el momento de ejecutar dicha función o bloque. De igual forma, esta memoria se liberará de forma automática en el momento de que concluya la ejecución de dicha función o bloque de código.

Vectores de punteros a objetos

En la asignatura Estructuras de Datos se estudiarán otras estructuras especialmente optimizadas para el almacenamiento y recuperación eficiente de datos. Sin embargo, por ahora el uso de vectores puede seguir siendo una herramienta apropiada si los utilizamos de una forma adecuada. En particular, muchos problemas se pueden solventar utilizando vectores

de punteros a objetos.

Un vector de punteros a objetos únicamente almacena direcciones de memoria de objetos. Frente a almacenar objetos grandes, esto permite que sean más fáciles de copiar de una posición a otra al tratarse los punteros de tipos simples, puesto que sólo se copiará la dirección de memoria y no se hará por tanto uso del operador de asignación de la clase de los objetos. Además, cada objeto se puede identificar tanto por la posición de su dirección de memoria en el vector, aunque esta podría cambiar como vimos anteriormente, como también por su dirección de memoria, que será permanente.

Por otra parte, podemos crear un vector de punteros sin tener obligatoriamente que crear inicialmente los objetos que se almacenarán. En este caso, podemos simplemente inicializar todas las posiciones a *nullptr* (dirección de memoria nula). Posteriormente, cuando queramos crear un objeto, a diferencia de la creación de un vector de objetos, podemos elegir el constructor en particular que queramos utilizar al utilizar el operador new. Además, los vectores así creados, al ser de tamaño inferior, se verán afectados en menor medida por el problema de la fragmentación de memoria. De hecho, los objetos que se vayan creando se irán organizando en diferentes huecos existentes al no tener que estar almacenados de forma consecutiva.

Sin embargo, los vectores de punteros presentan un inconveniente frente a los vectores de objetos y es que, a la hora de liberarlos, ya sea de forma automática si están en la pila o de forma expresa con el operador delete[] si están en memoria dinámica, no se llama de forma automática a los destructores de los objetos apuntados. Por lo tanto, es responsabilidad del programador encargarse de esta tarea.

Ejemplos

```
int main ()
{  //(Vector 1) Vector automático; objetos instanciados con
   //           constructor por defecto. OBJETOS EN LA PILA
   Plato primeros[31];

   //(Vector 2) Vector automático de punteros a objetos Plato.
   //           Punteros no inicializados. PUNTEROS EN LA PILA
   Plato* segundos[31];

   //(Vector 2) Ejemplo de inicialización de vector de punteros
   //           a dirección nula
   for ( int i = 0; i < 31; i++)
   {  segundos[i] = nullptr;
   }

   //(Vector 2) Ejemplo de instanciación de un nuevo objeto,
   //           usando un puntero del vector. OBJETO DINÁMICO
   segundos[2] = new Plato ( "Chipirones fritos", 9.50 );
```

```
   /* ... */
   // (Vector 1) Paso de mensaje a un objeto almacenado en el
   //            vector 1. OBJETO EN LA PILA
   cout << primeros[0].getPrecio();

   // (Vector 2) Paso de mensaje a un objeto cuya dirección
está
   //            almacenada en el vector 2. OBJETO DINÁMICO
   cout << segundos[0]->getPrecio();

   // (Vector 2) Debemos liberar expresamente los objetos del
   //            vector automático de punteros.
   //            OBJETOS DINÁMICOS
   for (int i = 0; i < 31;i++)
   {  if ( segundos[i] != nullptr )
      {  delete segundos[i];
         segundos[i] = nullptr;
      }
   }

   return 0;
}
// Acaba el programa:
// (Vector 2) Se libera el vector segundos
// (Vector 1) El vector primeros se libera automáticamente
//            llamando antes al destructor de cada objeto
```