Programación de Sistemas de Archivos Secuenciales con Objetos en Lenguaje C++

Bruno López Takeyas

***Resumen*— Este documento presenta una alternativa de programación de sistemas de archivos secuenciales mediante la técnica orientada a objetos y está dirigido a programadores interesados en diseñar sistemas de administración de archivos cuyo objetivo es facilitar su comprensión y posterior codificación. Como ejemplo de aplicación se describe el diseño y codificación de un archivo secuencial que almacena registros de empleados de una empresa.**

***Palabras Clave*—Programación orientada a objetos, archivo, registro, clase, objeto, atributo, constructor.**

I. Introducción

Existen varias técnicas para representar y almacenar registros llamadas organizaciones de archivos. Hay dos aspectos importantes en que difieren las organizaciones de archivos: la secuenciación de registros y el conjunto de operaciones para manipularlos [1].

La forma más sencilla de almacenar un conjunto de registros en un archivo es mediante la organización secuencial. En este tipo de archivos, los registros son escritos consecutivamente cuando el archivo es creado, por lo tanto, deben ser accesados de ese modo cuando se consultan.

La característica mas importante de esta técnica de organización de archivos es que solo permite el acceso secuencial, es decir, para accesar al registro k, se deben recorrer los k-1 registros anteriores. Esta característica impide que el archivo secuencial se use en procesos en línea debido a que no se pueden consultar rápidamente los registros, sin embargo, muestra buen rendimiento en procesos en lote donde se aprovecha la capacidad para accesar al siguiente registro rápidamente. Otra ventaja de esta organización de archivos radica en que la dirección del registro esta implícita en el sistema; es decir, no se pierden registros por la desaparición de direcciones.

Es posible programar sistemas de archivos mediante lenguajes de tercera generación (como el lenguaje C++) para aplicaciones relativamente sencillas y como antecedente teórico-práctico del uso de manejadores de bases de datos robustos. Sin embargo, es recomendable aprovechar las técnicas de programación de estos lenguajes para facilitar el diseño y codificación de dichas aplicaciones.

Bruno López Takeyas es profesor del Depto. de Sistemas y Computación del Instituto Tecnológico de Nuevo Laredo, Nuevo Laredo, Tam. MÉXICO. (Tel: 01 (867) 7140022; e-mail: takeyas@itnuevolaredo.edu.mx).

II. Fundamentos de programación orientada a objetos en lenguaje C++

La programación orientada a objetos (POO) simula objetos reales con equivalentes de software. Utiliza las relaciones de clase en la que sus objetos tienen las mismas características, atributos y comportamientos.

La POO encapsula datos (atributos) y métodos (comportamientos) en paquetes llamados clases. Las clases tienen la propiedad de ocultamiento, esto es, que aunque los objetos se comunican entre sí por medio de interfaces bien definidas, normalmente no se les permite saber la manera en que se implementan en otros objetos. La diferencia entre clase y objeto es sutil pero importante. La clase es el concepto abstracto, es una especie de molde para crear objetos, define los atributos y métodos. Por otro lado, el objeto es la entidad concreta; es decir, a partir de una clase, se pueden crear instancias u objetos donde cada uno tiene valores distintos para sus atributos [2].

Mientras que en lenguaje C y otros lenguajes procedimentales, la programación esta orientada a funciones y procedimientos, en lenguaje C++ esta orientada a objetos. En C, la unidad de programación es la función, mientras que en C++ es la clase, a partir de la cual los objetos son instanciados [3].

La clave para desarrollar sistemas utilizando los conceptos de orientación a objetos es definir los objetos que lo forman, sus atributos, sus métodos y crear las abstracciones apropiadas para ellos (clases), separando la implementación interna de su comportamiento externo. La abstracción consiste en ocultar los detalles irrelevantes del objeto, esto provoca que el usuario maneje los datos del objeto sin necesidad de conocer los detalles.

A. Partes públicas y privadas de las clases

Una clase puede tener partes públicas y privadas. Regularmente los componentes de las clases son privados, esto significa que no pueden ser accesados desde afuera de la clase, sino únicamente por los métodos de la misma. Estos componentes se declaran con la palabra “private” [3].

Los elementos públicos de la clase pueden ser accesados desde afuera de la clase, pueden consultar sus componentes privados y se declaran con la palabra “public” (Fig. 1).

```
class Nombre_Clase
{
   public:
     // declaración de atributos y métodos públicos

   private:
     // declaración de atributos y métodos privados
};
```

Fig. 1. Formato para declarar una clase

B. Funciones para asignar y consultar valores de los atributos del objeto

Regularmente las clases contienen métodos para asignar y consultar valores de sus atributos. A los métodos para introducir valores se les llama "mutators" mientras que los métodos para accesar los valores de los atributos son conocidos como "accesors".

C. Declaración de los métodos de una clase

Normalmente los métodos son declarados como partes públicas de las clases y son los únicos que tienen derecho a acceder las partes privadas de la misma (atributos) y deben declararse como funciones o procedimientos de la clase (Fig. 2).

```
class Nombre_Clase
{
   public:

     // declaración del constructor
        Nombre_Clase(lista_parametros);

     // declaración de métodos públicos
        tipo Nombre_metodo1(lista_parametros);
        tipo Nombre_metodo2(lista_parametros);
                        .
                        .
   private:
     // declaración de atributos privados
        tipo atributo1;
        tipo atributo2;
                        .
                        .
};
```

Fig. 2. Formato para declarar una clase, sus métodos y sus atributos

Nótese que en esta declaración no se desarrolla la codificación completa de los métodos, sino ésta debe hacerse aparte usando un formato en el que se tiene que especificar la clase al que pertenece (Fig. 3).

```
tipo Nombre_Clase::Nombre_metodo(lista_parametros)
{
   estatutos;
}
```

Fig. 3. Formato para codificar los métodos

D. Constructores

Es muy común que algunos atributos del objeto deban ser inicializados antes de utilizarlos. Para ello se utiliza el constructor, el cual es una función especial que pertenece a la clase y se ejecuta automáticamente cuando se crea una instancia de la clase, es decir, cuando se declara un objeto. El constructor tiene las siguientes características:

- Tiene el mismo nombre que la clase.
- No tiene definido el tipo del valor de retorno.

El constructor puede o no tener parámetros. Cuando se declara un objeto cuya clase tiene un constructor con parámetros, debe usarse el formato de declaración mostrado en la Fig. 4.

```
Nombre_Clase Nombre_objeto(lista_parametros);
```

Fig. 4. Formato para declarar un objeto con parámetros

E. Declaración de registros

En lenguaje C++, los registros se declaran por medio de la sentencia struct. En él se definen los campos que lo forman utilizando la declaración de los tipos de datos tradicionales que proporciona el lenguaje (*int*, *char*, *float*, *double*, *long*, etc.). Aquí se declaran los datos del registro que posteriormente se grabarán en el archivo (Fig. 5).

```
struct tipo_registro
{
    // Declaración de los campos del registro
          tipo campo1;
          tipo campo2;
                        .
                        .

};
```

Fig. 5. Formato para declarar un tipo de registro.

III. DISEÑO DE SISTEMAS DE ARCHIVOS CON OBJETOS

Para ilustrar mejor la programación de archivos con objetos, se muestra el diseño de un sistema de un archivo secuencial en el que se requiere almacenar datos de empleados de una empresa. En este ejemplo de aplicación se hacen las declaraciones del registro y la clase para la manipulación del archivo de empleados. Primero se definen los datos o campos que se desean almacenar y se desarrollan los métodos de altas y listado de registros.

A. Declaración del registro de empleados

Los datos (campos) requeridos para los registros de empleados son mostrados en la Fig. 6.

Campo	Descripción	Tipo	Formato
Num	Número de empleado	Entero	
Nombre	Nombre	Alfanumérico	25 caracteres
Depto	Departamento	Caracter	
Sueldo	Sueldo	Real	4.2

Fig. 6. Campos del registro de Empleados.

La declaración de la estructura del registro de Empleados se hace de ámbito global y se muestra en la Fig. 7.

```
struct tipo_registro
{
  int num;
  char nombre[25];
  char depto;
  float sueldo;
};
```

Fig. 7. Declaración global del tipo de registro y sus campos

B. Declaración de la clase

Una vez declarada la estructura global que define los campos del registro, se declara la clase *Archivo_Secuencial* que contiene los métodos y atributos del archivo secuencial (Fig. 8).

```
class Archivo_Secuencial
{
   public:
     Archivo_Secuencial(char *nom); // Constructor
     void Altas();//Método para insertar registros
     void Listado(); //Método para enlistar

   private:
     struct tipo_registro registro;
     FILE *alias; // Alias del archivo
     char *nombre_archivo;// Nombre del archivo
};
```

Fig. 8. Declaración global de la clase del archivo secuencial de empleados

La clase del archivo secuencial tiene tres atributos privados:

1. El registro con los campos de los empleados.
2. El apuntador *alias* que controla el archivo.
3. El nombre del archivo que contiene la ruta de acceso.

Además se declaran sus métodos públicos:

1. El constructor *Archivo_Secuencial* que recibe como parámetro el nombre del archivo.
2. El método *Altas()* para insertar registros con los datos de los empleados
3. El método *Listado()* para mostrar en la pantalla los datos de todos los empleados almacenados en el archivo.

C. Desarrollo del constructor del archivo secuencial

Una manera de inicializar el objeto del archivo secuencial es definiendo el nombre del archivo que manipulará. Para ello, se define un constructor que acepta como parámetro una cadena de caracteres con la ruta y nombre del archivo (Fig. 9).

```
Archivo_Secuencial::Archivo_Secuencial(char *nom)
{
// Inicializar el nombre del archivo
   strcpy(nombre_archivo,nom);
}
```

Fig. 9. Constructor del archivo secuencial.

Aquí el constructor se diseña como un procedimiento que acepta una cadena definida como un apuntador de caracteres y lo copia o inserta en el atributo del nombre del archivo del objeto; es decir, se establece el nombre del archivo del objeto.

D. Desarrollo del método Altas() en el objeto del archivo secuencial

El método *Altas()* declarado en la clase del archivo secuencial tiene como finalidad insertar secuencialmente los registros de empleados en el archivo del objeto (Fig. 10).

```
  void Archivo_Secuencial::Altas()
  {
    int num;
    clrscr();
    if((alias=fopen(nombre_archivo,"rb+"))==NULL)
      alias=fopen(nombre_archivo,"wb");

    cout << "\n<<< ALTAS >>>";
    cout << "\n\n\nNo. de Ctrl: ";
    cin >> num;
    alias=fopen(nombre_archivo,"rb+");
    if(alias==NULL)
      alias=fopen(nombre_archivo,"wb");

    fread(&registro,sizeof(registro),1,alias);
    while(!feof(alias))
    {
       if(registro.num==num)
       {
  cout << "\n\n\nYa existe ese registro !!!";
  cout  <<  "\n\n\nOprima  cualquier  tecla  para
continuar...";
  fclose(alias);
  getch();
  return;
       }
       fread(&registro,sizeof(registro),1,alias);
    }

    registro.num=num;
    cout << "\nNombre: "; gets(registro.nombre);
    cout << "\nDepto : "; cin >> registro.depto;
    cout << "\nSueldo: "; cin >> registro.sueldo;
    fwrite(&registro,sizeof(registro),1,alias);
    fclose(alias);
    cout << "\n\nDatos registrados !!!";
    cout  <<  "\n\n\nOprima  cualquier  tecla  para
continuar...";
    getch();
    return;
  }
```

Fig. 10. Método para insertar secuencialmente los registros de empleados.

E. Desarrollo del método Listado() en el objeto del archivo secuencial

El método *Listado()* declarado en la clase del archivo secuencial tiene como finalidad enlistar todos los registros de empleados en el archivo del objeto (Fig. 11).

```
  void Archivo_Secuencial::Listado()
  {
    clrscr();
    if((alias=fopen(nombre_archivo,"rb+"))==NULL)
    {
      cout << "Error: No existe el archivo ...";
      cout  <<  "\n\n\nOprima  cualquier  tecla  para
continuar...";
      getch();
      return;
    }

    cout << "\n<<< LISTADO >>>";
```

```
    cout << "\n\n\nNo.Ctrl                    Nombre
Dep      Sueldo ";
    cout                                          <<
"\n=============================================
===";

    fread(&registro,sizeof(registro),1,alias);
    while(!feof(alias))
    {
       printf("\n%5d   |   %25s   |   %c   |     $%8.2f
|",registro.num,registro.nombre,registro.depto,regis
tro.sueldo);
       fread(&registro,sizeof(registro),1,alias);
    }

    fclose(alias);
    cout << "\n\n\nFin de listado...";
    cout  <<  "\n\n\nOprima  cualquier  tecla  para
continuar...";
    getch();
    return;

  }
```

Fig. 11. Método para enlistar los registros de empleados.

F. Declaración del objeto de empleados

Una vez hecha la declaración de la clase y desarrollado su constructor y sus métodos, se procede a declarar en forma global el objeto *Empleados* perteneciente a la clase *Archivo_Secuencial*. Como el constructor de la clase espera una cadena como parámetro, es necesario incluir la ruta y el nombre del archivo en la declaración del objeto (Fig. 12).

```
Archivo_Secuencial Empleados("c:\\EMPLEADO.DAT");
```

Fig. 12. Declaración del objeto Empleados de la clase Archivo_Secuencial.

G. Llamadas a los métodos del objeto de Empleados

Para insertar registros de empleados en el archivo secuencial, basta con invocar o llamar el método de *Altas()*. Para lograrlo, es necesario especificar el objeto al que se hace referencia, ya que las características de la clase *Archivo_Secuencial* pueden ser heredadas a varios objetos (Fig. 12).

```
Empleados.Altas();
```

Fig. 12. Llamada del método Altas() para insertar registros en el archivo secuencial del objeto Empleados.

De forma semejante, se invoca el método *Listado()* para enlistar en pantalla todos los registros del archivo de Empleados.

```
Empleados.Listado();
```

Fig. 13. Llamada del método de Listado() para mostrar en pantalla todos los registros del archivo secuencial del objeto Empleados.

H. Menú principal del sistema

En la rutina principal del sistema, se codifica un menú que muestra las opciones y se invocan los métodos correspondientes (Fig. 14).

```
int main()
{
  int op;
  do
  {
     clrscr();
     cout  <<  "Programa  de  manejo  de  archivos
secuenciales con objetos\n\n"<<endl;
     cout << "1.- Altas."<<endl;
     cout << "2.- Listado."<<endl;
     cout << "0.- Salir."<<endl;

     cout << "\n\n\nOpcion ? ";
     cin >> op;

     switch(op)
     {
       case 1: Empleados.Altas();   break;
       case 2: Empleados.Listado(); break;
     }
  }while(op!=0);
  return 0;
}
```

Fig. 14. Rutina principal del sistema.

IV. CONCLUSIONES

La POO es una técnica de programación sumamente eficiente que permite dar versatilidad a los sistemas y facilitar la codificación para los programadores.

En la aplicación de archivos secuenciales mostrada en este documento, fácilmente se pueden agregar otros archivos sin necesidad de reprogramar las subrutinas o agregar nuevos procedimientos. Para crear otros archivos secuenciales con las mismas características, basta declarar otros objetos de la misma clase con la ruta y nombre de los archivos deseados. P. ejem. Si se deseara un archivo secuencial de *Secretarias* y otro de *Obreros*, se hace la declaración correspondiente de los objetos y se encuentran listos para ser utilizados (Fig. 15).

```
Archivo_Secuencial Secretarias("c:\\SECRE.DAT");
Archivo_Secuencial Obreros("d:\\OBREROS.DAT");
```

Fig. 15. Declaración de nuevos objetos de archivos secuenciales.

Esta declaración permite que estos dos nuevos objetos hereden las características de la clase a la que pertenecen y por lo tanto ya tienen declarados los métodos y atributos listos para ser utilizados.

El código fuente del sistema mostrado en este documento puede descargarse accesando el sitio web http://www.itnuevolaredo.edu.mx/takeyas o solicitándolo enviando un correo electrónico al autor, escribiendo a takeyas@itnuevolaredo.edu.mx.

REFERENCIAS

[1] López Takeyas, Bruno. "Conceptos básicos de administración de archivos". Artículo. ITNL. 2003. http://www.itnuevolaredo.edu.mx/takeyas

[2] Lafore, Robert. "Object-Oriented Programming in Turbo C++". Waite Group. 1994

[3] Deitel/Deitel. "C++. Cómo programar". 4a. edición. Editorial Pearson Prentice Hall. 2003.

[4] Lafore, Robert. "Turbo C. Programming for the PC". Waite Group. 1990.

[5] López Takeyas, Bruno. "Manual de manejo de archivos en Lenguaje C++". Apuntes. ITNL. 2004. http://www.itnuevolaredo.edu.mx/takeyas

[6] Staugaard, Andrew. "Técnicas estructuradas y orientadas a objetos. Una introducción utilizando C++". Addison Wesley. 1995.

Bruno López Takeyas se tituló de Ingeniero en Sistemas Computacionales en el Instituto Tecnológico de Nuevo Laredo en 1993. Obtuvo el grado de Maestro en Ciencias de la Administración con especialidad en Sistemas en la Universidad Autónoma de Nuevo León en el año 2000. Desde 1994 es profesor del Depto. de Sistemas y Computación del Instituto Tecnológico de Nuevo Laredo. Es autor de varios artículos en las áreas de programación, algoritmos genéticos e inteligencia artificial.