Capítulo 3

Introducción al lenguaje de programación C++/C++.NET

A pesar del gran crecimiento que han experimentado otros lenguajes de programación como Java y C#, o el gran número de usuarios que programan en Visual Basic, lo cierto es que C++ continúa siendo el lenguaje elegido por la gran mayoría de los desarrolladores a la hora de hacer frente a proyectos en los que prima más el rendimiento, la flexibilidad y el control que la facilidad en el diseños de aplicaciones y en la implementación. Esto es un hecho.

Junto a la herramienta Visual C++ 6.0, se convierte en uno de los métodos de desarrollos más potentes que hoy día pueden encontrarse para la plataforma Windows. La nueva versión Visual C++.NET (sobre la que se ha desarrollado la aplicación asociada a este Proyecto), añade a las posibilidades existentes en su predecesora (la bibliotecas de clases prefabricadas MFC –Microsoft Foundation Classes– y ATL –Active Template Library–) una renovación de las mismas, un importante salto al mundo de las aplicaciones Web y, como no, la incorporación de la plataforma .NET junto con los elementos necesarios para generar código MSIL (*Microsoft Intermediate Language*).

Dada la gran capacidad del lenguaje C++, lo complejo de algunas situaciones y su versatilidad para adaptarse a multitud de ellas; el objetivo de este capítulo es introducir al programador en los conceptos principales y particulares del lenguaje mediante breves indicaciones. En el Capítulo 6 (dedicado a la programación desarrollada) pueden encontrarse algunas situaciones particulares donde apreciar los conceptos aquí comentados.

3.1 Posibilidades

De todos los lenguajes incorporados en el entorno Visual Studio.NET, C++.NET es el único que permite trabajar y ejecutar código nativo Win32, junto con el código MSIL. Esta característica es lo que lo diferencia claramente del resto de lenguajes.

Además de la biblioteca de clases .NET, común a todos los lenguajes .NET, con C++.NET podemos crear aplicaciones haciendo uso de la biblioteca de clases MFC[1] muy difundida, así como de la biblioteca de plantillas C++ conocida como ATL. De esta forma, se abre un abanico de posibilidades que no puede darse con otro lenguaje diferente de C++.NET.

[1] Visual C++.NET incorpora una nueva versión de MFC 7.0.

Otra de las opciones de que dispone es la de crear aplicaciones nativa para Windows haciendo uso directamente del API de Windows a través del lenguaje C++, sin bibliotecas, ni clases, ni plantillas.

Pero como "no es oro todo lo que reluce", debe decirse en detrimento de este lenguaje, que no cuenta con diseñadores equivalentes a los de otros lenguajes como Visual Basic.NET o C#.NET, de ahí que su aprovechamiento resulte bastante más complejo.

3.2 Aplicaciones .NET

A través de entornos de programación como Visual Studio.NET puede tenerse acceso a los servicios que proporciona la plataforma .NET; en esta situación no se dispone de bibliotecas de clases a través de archivos cabecera, ni de código precompilado en una DLL, sino de un código intermedio y clases que se autodescriben, sin necesidad de lo anterior.

El núcleo de la biblioteca de clases Microsoft .NET Framework se aloja en la biblioteca *mscorlib.dll*; para poder importarla desde cualquier módulo de C++.NET y así poder hacer uso de sus servicios, se utiliza la directiva "#using". La tecnología .NET se analiza en un capítulo independiente de este mismo documento.

3.3 El lenguaje C++

La mejor forma de estudiar el lenguaje de programación C++ es partir de una buena base de su predecesor, el lenguaje C/ANSI C.

Para ello la referencia [11] y [12] se presentan como una buena elección. En esta última, se estudia el lenguaje C++ aplicando modificaciones al anterior. De esta forma pueden encontrarse modificaciones menores y modificaciones mayores, en función de la relevancia que tenga cada una.

El propósito de este apartado es resaltar alguna de esas modificaciones y de las principales propiedades del lenguaje, que han sido relevantes en el desarrollo de la aplicación asociada. Así mismo, para completar la formación de programador, no puede quedarse en este trabajo, debe acudirse a bibliografía cada vez más avanzada; en este sentido las anteriores referencias [11] y [12] constituyen un buen punto de partida, que junto con el presente documento, lo preparan para iniciarse en la tecnología .NET; el resto corre de su cuenta.

3.3.1 Primeros pasos

En este apartado se definen una serie de conceptos sencillos; no se parte de cero, se supone que el programador ya se ha iniciado y al menos conoce los principios fundamentales (acuda a la referencia [11] de no ser así). Se pretende definir:

- Visibilidad de las variables
- Declaración, definición e inclusión de funciones
- Variables por valor y referencia (punteros)

Visibilidad

Se trata de definir el ámbito de utilidad de determinadas variables. Por ejemplo, si estamos interesados en obtener el máximo de una serie de valores, una forma de operar sería crear una variable "Maximo", inicializarla con el primer elemento y realizar una comparativa entre éste y todos los demás elementos (mediante un bucle); si resulta que alguno es mayor, se actualiza la variable, si no se pasa al siguiente elemento hasta completar la lista. Para poder hace uso de la variable "Maximo" fuera del bucle que compara cada valor, debe ser definida fuera de él, puesto que si se define dentro de las sentencias pertenecientes al mismo bucle, al salir de él, la variable quedará extinguida.

Declaración, definición e inclusión de funciones

Existe diferencia entre cada una de ellas. Para una misma función, podemos diferenciar: el código que rige los cálculos de la función, los parámetros de entrada y salida, y la llamada de la función desde otra función por ejemplo. Cada una de estas partes se corresponde con uno de los conceptos anteriores.

La declaración de la función permite visualizar de un vistazo el tipo de parámetros que maneja, tanto de entrada como de salida.

La definición es el cuerpo de la función, y permite estudiar sus operaciones partiendo de las variables de entrada para devolver unas de salida.

La llamada a una función desde otra, es el propósito por el que se establecen las anteriores etapas. Esto permite que el código sea más fácilmente legible. Se propone el siguiente ejemplo: imagine una pila de objetos, algunos iguales y algunos diferentes; si agrupamos los objetos en cajas independientes según su tipo, se facilita mucho la búsqueda verdad, pues aquí es lo mismo.

Variables por valor y por referencia

La entrada de datos puede hacerse de estas dos formas. Si se pretende operar con la variable para devolver un cierto valor a otra variable, estamos ante el caso de paso por valor. Por el contrario, si se está interesado en modificar el valor de la variable, entonces se debe usar el paso por referencia.

El paso de una variable por referencia, lo que hace en realidad es mostrar la dirección que ocupa la variable en la memoria, y así puede tenerse acceso a ella.

Esta propiedad es muy útil cuando se trabaja con vectores, matrices o cualquier otro tipo de lista de datos.

3.3.2 Conceptos importantes

Los conceptos que se definen a continuación son algo más complejos que los anteriores, no obstante es importante dejar claro su significado para poder entender el conjunto de la programación. Éstos son:

- Sobrecarga de funciones

- Operadores new/delete (para objetos)

- Clases: Propiedades y Métodos
- Tipos de propiedades: protected, public y private
- Miembros básicos: constructor y destructor.

Sobrecarga de funciones

Se trata de que varias funciones pueden presentarse a través de un mismo nombre, pero con operaciones totalmente distintas; la diferencia se presenta el número o el tipo de las variables de entrada o salida.

Operadores new/delete (para objetos)

Estos operadores permiten realizar la reserva dinámica de memoria, definir una nueva variable o un nuevo objeto (caso de "new") o borrarlo (caso de "delete").

En el caso de variables de tipo estándar (int, double, flota, etc…) no tiene mucha aplicación, pero para el caso de otro tipo de objetos forma parte de la forma básica de creación, sin la cual tendría que acudirse a métodos bastante más complejos.

Clases: Propiedades y Métodos

En el Capítulo 2 se definió las propiedades principales de la OOP; entre ellas se destacaba que, para la creación de objetos debía hacerse uso de clases cuyos componentes eran "variables" –propiedades– y conjuntos de código agrupados en "funciones" que reciben el nombre de métodos, y que hacían uso tanto de la información contenida en dichas propiedades, como de otra información externa a la propia clase (que puede ser proporcionada al crear el objeto o al llamar al método correspondiente)

Tipo de propiedades: protected, public y private

Estas categorías definen de alguna manera el uso que pueda darse sobre una propiedad de una clase. En el caso de "public", la información puede usarse directamente desde fuera de la clase, bien por herencia, bien por un control que maneje el objeto sin heredarlo. Por otro lado, "private" permite que se haga uso de la información sólo desde el propio objeto o bien a través de la herencia, pero nada más. Y por último, el caso "protected" impide cualquier manipulación desde fuera del propio objeto, ni con herencia ni con cualquier otra forma; es la más restrictiva de todas.

Estas situaciones tienen una aplicación más o menos estándar, sin embargo en algunas circunstancias puede ser interesante y necesario modificar dichos estándares para poder proceder más fácilmente.

Miembros básicos: constructor y destructor

El caso del destructor no es tan interesante (para esta aplicación en particular, no en general), pero el constructor es un concepto muy utilizado a lo largo de toda la programación realizada.

Se trata de un método (el constructor) que permite inicializar un objeto de un tipo concreto de clase. Puede recibir una serie de datos de entrada para adjudicárselos a las

propiedades de la clase de forma automática; incluso puede hacer llamadas a otros métodos propios de la misma clase.

Es el primer paso para la creación de un objeto, aunque en realidad, la reserva en memoria del objeto puede hacerse de forma separada a la llamada al constructor, sin embargo, no podrá usarse el objeto hasta haber realizado este paso.