Chương 7

Ghi/đọc dữ liệu của ứng dụng C# ra file

7.1 Tổng quát về đời sống của dữ liệu ⊂ ứng dụng VC#

Khi chương trình bắt đầu chạy, nó sẽ tạo ra dữ liệu, xử lý dữ liệu cho đến khi hoàn thành nhiệm vụ và dừng chương trình.

Khi chương trình kết thúc, thường các dữ liệu của chương trình sẽ bị mất. Do đó, trong lúc chương trình chạy, khi cần thiết, ta sẽ ghi các dữ liệu ra file để lưu giữ lâu dài. Sau này khi cần dùng lại, ta sẽ đọc dữ liệu từ file vào các biến chương trình để xử lý tiếp.

VC# cung cấp 3 class đối tượng FileStream, BinaryWriter, BinaryReader (trong namespace System.IO) để phục vụ việc ghi/đọc biến dữ liệu thuộc các kiểu định sẵn phổ biến ra file ở dạng nhị phân (dạng được mã hóa trong chương trình). Nếu biết được cách thức mã hóa dữ liệu (được trình bày trong môn Nhập môn điện toán), ta sẽ kiểm tra trực tiếp được kết quả được ghi ra file.

7.2 Ghi dữ liệu ra file ở dạng nhị phân

Qui trình điển hình để ghi dữ liệu trong chương trình ra file ở dạng nhị phân (không giãi mã dữ liệu) :

```
//1. tạo đối tượng quản lý file
FileStream stream = new FileStream("C:\\data.bin",
    FileMode.Create);
//2. tạo đối tượng phục vụ ghi file
BinaryWriter writer = new BinaryWriter(stream);
//3. xử lý dữ liệu theo yêu cầu chương trình
int i = -15;
double d = -1.5;
String s = "Nguyễn Văn Hiệp";
bool b = true;
//4. ghi dữ liệu ra file
writer.Write(b); writer.Write(i); writer.Write(d); writer.Write(s);
```

```
//5. đóng các đối tượng được dùng lại
writer.Close(); stream.Close();
```

Tác vụ write của class BinaryWriter có 14 biến thể 1 tham số để ghi đýợc 14 kiểu dữ liệu định sẵn phổ biến sau đây :

- Boolean
- Byte, SByte
- Int16, Int32, Int64
- UInt16, UInt32, UInt64
- Single, Double, Decimal
- Byte[] , Char[]
- Char, String

Muốn ghi nội dung của biến thuộc 1 trong 14 kiểu dữ liệu định sẵn trên, ta gọi tác vụ write theo dạng sau :

```
writer.Write(varname); //writer là biến đối tượng
    BinaryWriter
```

Tác vụ write của class BinaryWriter còn có 2 biến thể 3 tham số để ghi được các phần tử chọn lọn trong danh sách :

```
//ghi count byte từ vị trí index trong danh sách buffer
BinaryWriter.write(Byte[] buffer, int index, int count);
//ghi count ký tự từ vị trí index trong danh sách buffer
BinaryWriter.write(Char[] buffer, int index, int count);
```

7.3 Đọc dữ liệu từ file ở dạng nhị phân

Qui trình điển hình để đọc dữ liệu từ file nhị phân vào chýõng trình (không mã hóa dữ liệu) :

```
//1. tạo đối tượng quản lý file
FileStream stream = new FileStream("C:\\data.bin",
    FileMode.Open);
//2. tạo đối tượng phục vụ đọc file
BinaryReader reader = new BinaryReader(stream);
//3. định nghĩa các biến dữ liệu theo yêu cầu chương trình
int i; double d; String s; bool b;
//4. đọc dữ liệu từ file vào các biến
```

```
b= reader.ReadBoolean();       //đọc trị luận lý
i = reader.ReadInt32();    //đọc số nguyên 32 bit
d = reader.ReadDouble();       //đọc số thực chính xác kép
s = reader.ReadString();  //đọc chuỗi
//5. đóng các đối tượng được dùng lại
reader.Close(); stream.Close();
```

7.4 Ghi dữ liệu ra file ở dạng text

Mặc dù việc ghi/đọc dữ liệu ra file ở dạng nhị phân (y như trong máy) là rất đơn giản, hiệu quả (khỏi phải thực hiện mã hóa/giải mã dữ liệu). Tuy nhiên, file nhị phân cũng có 1 số nhược điểm :

- người dùng khó xem, khó kiểm tra nội dung của file.
- người dùng khó tạo dữ liệu dưới dạng nhị phân để chương trình đọc vào xử lý.

Trong trường hợp cần nhập nhiều thông tin cho chương trình, ta không thể dùng các đối tượng giao diện như textbox, listbox. Trong trường hợp này, ta sẽ dùng trình soạn thảo văn bản để soạn dữ liệu dưới dạng văn bản hầu xem/kiểm tra/sửa chữa dễ dàng. File văn bản chứa dữ liệu là danh sách gồm nhiều chuỗi, mỗi chuỗi miêu tả 1 dữ liệu (luận lý, số nguyên, số thực, chuỗi,...), các chuỗi sẽ được ngăn cách nhau bởi 1 hay nhiều dấu ngăn. Dấu ngăn thường dùng là ký tự gióng cột TAB, ký tự xuống hàng.

VC# cung cấp các class đối tượng có tên là FileStream, StreamWriter, StreamReader (trong namespace System.IO) để phục vụ việc ghi/đọc biến dữ liệu thuộc các kiểu định sẵn phổ biến ra file ở dạng text. Trong trường hợp này, tác vụ ghi dữ liệu sẽ tự động giải mã dạng nhị phân sang dạng chuỗi tương đương trước khi ghi ra file. Khi đọc lại chuỗi miêu tả dữ liệu, ta phải mã hóa dữ liệu từ dạng chuỗi thành dạng nhị phân trước khi chứa vào biến dữ liệu bên trong chương trình.

Qui trình điển hình để ghi dữ liệu trong chương trình ra file ở dạng text (giãi mã dữ liệu nhị phân thành dạng chuỗi) :

```
//1. tạo đối tượng quản lý file
FileStream stream = new FileStream("C:\\data.txt",
    FileMode.Create);
//2. tạo đối tượng phục vụ ghi file
StreamWriter writer = new StreamWriter(stream,
    Encoding.Unicode);
//3. xử lý dữ liệu theo yêu cầu chương trình
int i = -15;
double d = -1.5;
String s = "Nguyễn Văn Hiệp";
bool b = true;
//4. ghi dữ liệu ra file
writer.Write(b); writer.Write("\t");      //ghi 1 dữ liệu và dấu ngăn
writer.Write(i); writer.WriteLine(); //ghi 1 dữ liệu và dấu ngăn
writer.Write(d); writer.Write("\t"); //ghi 1 dữ liệu và dấu ngăn
writer.Write(s); writer.Write("\t"); //ghi 1 dữ liệu và dấu ngăn
//5. đóng các đối tượng được dùng lại
writer.Close();
stream.Close();
```

7.5 Đọc dữ liệu từ file text

Qui trình điển hình để đọc dữ liệu từ file text vào chương trình (mã hóa dữ liệu từ chuỗi thành dữ liệu nhị phân) :

```
//1. tạo đối tượng quản lý file
FileStream stream = new FileStream("C:\\data.txt",
    FileMode.Open);
//2. tạo đối tượng phục vụ đọc file
StreamReader reader=new
    StreamReader(stream,Encoding.Unicode);
//3. định nghĩa các biến dữ liệu theo yêu cầu chương trình
int i; double d; String s; bool b; String buf=null;
//4. đọc dữ liệu từ file vào các biến
```

```
ReadItem(reader,ref buf); b = Boolean.Parse(buf);        //đọc trị luận lý
ReadItem(reader,ref buf); i = Int32.Parse(buf);      //đọc số nguyên 32 bit
ReadItem(reader,ref buf); d = Double.Parse(buf); //đọc số thực
ReadItem(reader,ref buf); s = buf;    //đọc chuỗi
//5. đóng các đối tượng được dùng lại
reader.Close(); stream.Close();
//hàm đọc chuỗi miêu tả 1 dữ liệu nào đó
static void ReadItem(StreamReader reader, ref String buf) {
    char ch;
    //thiết lập chuỗi nhập được lúc đầu là rỗng
    buf = "";
    //lặp cho đến khi hết file
    while (reader.EndOfStream != true) {
        ch = (char)reader.Read(); //đọc 1 ký tự
        if (ch != '\t' && ch != '\r' && ch!='\n') //nếu là ký tự bình thường
                buf += ch.ToString();
        else { //nếu là dấu ngăn thì kết thúc việc đọc chuỗi
            if (ch == '\r') reader.Read(); //đọc bỏ luôn ký tự '\n'
            return;       //trả kết quả về nơi gọi
        }
    }
}
```

7.6 Thí dụ về đọc/ghi dữ liệu cổ điển

Giả sử ta có 2 file A.txt và B.txt chứa thông tin về 2 ma trận theo qui ước như sau :

- chuỗi đầu tiên miêu tả số hàng
- chuỗi kế tiếp miêu tả số cột
- các chuỗi còn lại miêu tả giá trị từng phần tử, từng hàng từ trên xuống, mỗi hàng từ trái sang phải.
- các chuỗi dữ liệu được ngăn cách nhau bởi dấu ngăn ',', '\r', '\n'

Thí dụ ma trận (5,7) được chứa như sau :

```
5, 7
2, 3, 4, 5, 6, 7, 8
9, 10, 11, 12, 13, 14, 15
16, 17, 18, 19, 20, 21, 22
23, 24, 25, 26, 27, 28, 29
30, 31, 32, 33, 34, 35, 36
```

Ta hãy viết chương trình đọc 2 ma trận A và B vào bộ nhớ, tính ma trận tổng rồi xuất kết quả ra file văn bản S.txt.

Qui trình điển hình để xây dựng chương trình theo yêu cầu trên như sau :

1. Chạy VS .Net, chọn menu File.New.Project để hiển thị cửa sổ New Project.
2. Mở rộng mục Visual C# trong TreeView "Project Types", chọn mục Windows, chọn icon "Console Application" trong listbox "Templates" bên phải, thiết lập thư mục chứa Project trong listbox "Location", nhập tên Project vào textbox "Name:" (td. TongMT), click button OK để tạo Project theo các thông số đã khai báo.
3. Ngay sau Project vừa được tạo ra, cửa sổ soạn code cho chương trình được hiển thị. Thêm lệnh using sau đây vào đầu file :

```
using System.IO;
```

4. Viết code cho thân class Program như sau :

```
class Program {
        static double[,] A;     //ma trận A
        static double[,] B;     //ma trận B
        static double[,] S;     //ma trận S
        static int hang, cot;
        //hàm đọc ma trận vào biến bộ nhớ
        static void ReadMT(string path, ref double[,] A, ref int hang, ref int
cot) {
```

```
        //1. tạo đối tượng quản lý file
        FileStream stream = new FileStream(path, FileMode.Open);
        //2. tạo đối tượng phục vụ đọc file
        StreamReader reader = new StreamReader(stream,
Encoding.ASCII);
        //3. định nghĩa các biến dữ liệu theo yêu cầu chýõng
trình
        int i, j; string buf = "";
        //4. đọc dữ liệu từ file vào các biến
        ReadItem(reader, ref buf); hang = Int32.Parse(buf);      //đọc      số
hàng
        ReadItem(reader, ref buf); cot = Int32.Parse(buf);       //đọc số cột
        //phân phối vùng nhớ cho ma trận
        A = new double[hang, cot];
        //đọc từng phần tử ma trận
        for (i = 0; i < hang; i++)
              for (j = 0; j < cot; j++) {
           ReadItem(reader, ref buf);
           A[i, j] = Double.Parse(buf);   //đọc số thực
              }
        //5. đóng các đối týợng đýợc dùng lại
        reader.Close(); stream.Close();
}
//hàm ghi ma trận ra file text
static void WriteMT(string path, double[,] A, int hang, int cot) {
        //1. tạo đối tượng quản lý file
        FileStream stream = new FileStream(path, FileMode.Create);
        //2. tạo đối tượng phục vụ ghi file
        StreamWriter writer = new StreamWriter(stream, Encoding.ASCII);
        //3. định nghĩa các biến dữ liệu theo yêu cầu chýõng
trình
        int i, j;
        //4. ghi dữ liệu từ các biến ra file
```

```
        writer.Write(hang); writer.Write(", "); //ghi số hàng và dấu
ngăn
        writer.Write(cot); writer.WriteLine();  //ghi số cột và dấu
ngăn
        //ghi ma trận
        for (i = 0; i < hang; i++) { //ghi từng hàng ma trận
                for (j = 0; j < cot; j++) {
                    writer.Write(A[i, j]); writer.Write(",");        //ghi phần tử i,j
                }
                writer.WriteLine(); //ghi dấu xuống hàng
        }
        //5. đóng các đối tượng đượC dùng lại
    writer.Close(); stream.Close();
}
//hàm đọc chuỗi miêu tả 1 dữ liệu nào đó
static void ReadItem(StreamReader reader, ref String buf) {
        char ch = '\0';
        //thiết lập chuỗi nhập đượC lúc đầu là rỗng
        buf = "";
        while (reader.EndOfStream != true) {//lặp đọc bỏ các dấu ngăn
                ch = (char)reader.Read(); //đọc 1 ký tự
                if (ch != ',' && ch != '\r' && ch != '\n')
                        break;              //nếu là ký tự bình thýờng thì dừng
        }
        buf += ch.ToString();
        //lặp đọc các ký tự của chuỗi dữ liệu
        while (reader.EndOfStream != true) {
                ch = (char)reader.Read(); //đọc 1 ký tự
                if (ch == ',' || ch == '\r' || ch == '\n')
                        return;      //nếu là dấu ngăn thì dừng
                buf += ch.ToString();       //chứa ký tự vào bộ đệm
        }
}
//điểm nhập chương trình
```

```
        static void Main(string[] args) {
            int i, j, h = 0, c = 0;
            //đọc ma trận A từ file c:\A.txt
            ReadMT("c:\\a.txt", ref A, ref hang, ref cot);
            //đọc ma trận B từ file c:\B.txt
            ReadMT("c:\\b.txt", ref B, ref h, ref c);
            if (h != hang || c != cot)  {
                Console.WriteLine("Hai ma trận A, B không cùng kích thýớc");
                return;
            }
            //phân phối ma trận tổng S
            S = new double[hang, cot];
            //tính ma trận tổng S
            for (i = 0; i < hang; i++)
                for (j = 0; j < cot; j++) S[i, j] = A[i, j] + B[i, j];
            //xuất ma trận kết quả ra file c:\S.txt
            WriteMT("c:\\S.txt", S, hang, cot);
        } //hết hàm Main
} //hết class program
```

5. Chọn menu Debug.Start Debugging để dịch và chạy ứng dụng. Lưu ý là trước khi chạy ứng dụng, hãy dùng chương trình soạn thảo văn bản (NotePad, WordPad,...) soạn thảo 2 file c:\A.txt, c:\B.txt chứa dữ liệu của 2 ma trận A và B theo qui ước ở slide 12.

7.7 Ghi/Đọc hệ thống đối tượng ra/vào file

Đọc/ghi dữ liệu trên các biến thuộc kiểu giá trị (int, double, char[],..) rất dễ vì nội dung của các biến này không chứa tham khảo đến các thành phần khác. Ngược lại, việc đọc/ghi nội dung của 1 đối tượng thường rất khó khăn vì đối tượng có thể chứa nhiều tham khảo đến các đối tượng khác và các đối tượng có thể tham khảo vòng lẫn nhau. Để hỗ trợ việc đọc/ghi nội dung của đối tượng, VC# đề nghị kỹ thuật "Serialization".

Một đối tượng chỉ có thể được "serialize/deserialize" (ghi/đọc dùng kỹ thuật Serialization) nếu nó thuộc class "serializable". Để định nghĩa 1 class "serializable" dễ dàng và đơn giản nhất, ta chỉ cần thêm mệnh đề [Serializable] trước phát biểu định nghĩa class đó :

```
[Serializable]
public class A {...}
```

Để ghi 1 đối tượng (và toàn bộ các đối tượng mà nó phụ thuộc) ở dạng nhị phân, ta viết template như sau :

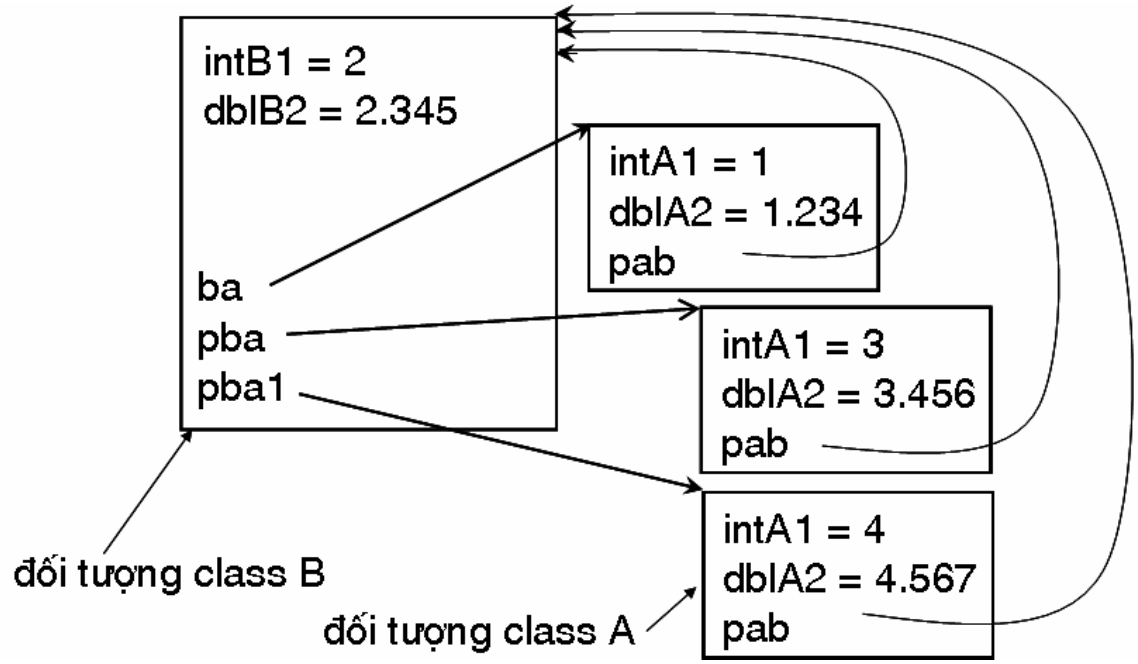
```
//1. xử lý và xây dựng đối tượng
B b;
//2. định nghĩa đối tượng FileStream miêu tả file chứa kết quả
FileStream fs = new FileStream("c:\\data.obj",
    FileMode.Create);
//3. tạo đối tượng BinaryFormatter phục vụ ghi đối tượng
BinaryFormatter formatter = new BinaryFormatter();
//4. gọi tác vụ Serialize  của formatter để ghi đối tượng
formatter.Serialize(fs, b);
//đóng file lại
fs.Flush();
fs.Close();
```

Để đọc lại 1 đối tượng (và toàn bộ các đối tượng mà nó phụ thuộc) ở dạng nhị phân ta, viết template như sau :

```
//1. định nghĩa biến đối tượng để chứa nội dung từ file
B b;
//2. định nghĩa đối tượng FileStream miêu tả file chứa dữ liệu đã có
FileStream fs = new FileStream("c:\\data.obj", FileMode.Open);
//3. tạo đối tượng BinaryFormatter phục vụ đọc đối tượng
BinaryFormatter formatter = new BinaryFormatter();
//4. gọi tác vụ Deserialize để đọc đối tượng từ file vào
b = (B) formatter.Deserialize(fs);
//đóng file lại
fs.Close();
```

7.8 Thí dụ về đọc/ghi hệ thống đối tượng

Giả sử ta có hệ thống các đối tượng với trạng thái và mối quan hệ giữa chúng cụ thể như sau. Lưu ý chúng có mối quan hệ bao gộp dạng vòng :



Giả sử biến b đang tham khảo tới đối tượng class B. Hãy viết chương trình ghi hệ thống đối tượng này lên file để khi cần, đọc lại vào bộ nhớ hầu xử lý tiếp.

Qui trình xây dựng ứng dụng giải quyết yêu cầu trên như sau :

1. Chạy VS .Net, chọn menu File.New.Project để hiển thị cửa sổ New Project.
2. Mở rộng mục Visual C# trong TreeView "Project Types", chọn mục Windows, chọn icon "Console Application" trong listbox "Templates" bên phải, thiết lập thư mục chứa Project trong listbox "Location", nhập tên Project vào textbox "Name:" (td. WRObject), click button OK để tạo Project theo các thông số đã khai báo.
3. Ngay sau Project vừa được tạo ra, cửa sổ soạn code cho chương trình được hiển thị. Thêm lệnh các using sau đây vào đầu file :

```
using System.IO;
```

```
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
```

4. Viết code cho hàm Main và các hàm dịch vụ khác nhau sau :

```
class Program {
    static String fbuf;
    static void Main(string[] args) { //điểm nhập của chương trình
      //xây dựng hệ thống đối tượng và ghi lên file
      Create_SaveObject();
      //đọc lại hệ thống đối tượng
      //ReadObject();
    }
    //hàm xây dựng hệ thống đối tượng và ghi lên file
    public static void Create_SaveObject() {
      //khở tạo đối tượng b theo hình ở silde 24
      B b = new B();
      b.init(2,2.345);
      b.Setba(1,1.234,b);
      b.Setpba(3,3.1416,b);
      b.Setpba1(4,4.567,b);
      //ghi đối tượng b dùng kỹ thuật Serialization
      try {
            //1. định nghĩa đối tượng FileStream miêu tả file chứa kết
    quả
            FileStream fs = new FileStream("c:\\data.obj", FileMode.Create);
            //2. tạo đối tượng BinaryFormatter phục vụ ghi đối tượng
            BinaryFormatter formatter = new BinaryFormatter();
            //3. gọi tác vụ Serialize  của formatter để ghi đối tượng
            formatter.Serialize(fs,b);
            //4. đóng file lại
            fs.Flush(); fs.Close();
      } catch (Exception e) { Console.WriteLine(e.ToString()); }
  }
    //hàm đọc đối tượng b dùng kỹ thuật Serialization
```

```
    public static void ReadObject() {
     try {
      //1. định nghĩa đối tượng FileStream miêu tả file chứa dữ liệu đã có
      FileStream fs = new FileStream("c:\\data.obj", FileMode.Open);
      //2. tạo đối tượng BinaryFormatter phục vụ đọc đối tượng
      BinaryFormatter formatter = new BinaryFormatter();
      //3. gọi tác vụ Deserialize để đọc đối tượng từ file vào
      B b = (B) formatter.Deserialize(fs);
      //4. đóng file lại
      fs.Close();
      } catch (Exception e) { Console.WriteLine(e.ToString()); }
    } //hết hàm Main
} //hết class program
```

5. Ấn phải chuột vào phần tử gốc của cây Project trong cửa sổ Solution Explorer, chọn option Add.Class, đặt tên là A.cs để tạo ra file đặc tả class A. Khi cửa sổ hiển thị mã nguồn của class A hiển thị, đặc tả class A như đoạn code dưới đây :

```
//thêm lệnh using sau ở đầu file
using System.Runtime.Serialization;
namespace WRObject {
  [Serializable]
  public class A {
        //định nghĩa các thuộc tính dữ liệu
        private int intA1;
        private double dblA2;
        private B pab;
        //định nghĩa các tác vụ
        public A() {}
        public void init(int a1, double a2, B p) {
              this.intA1 = a1;
              this.dblA2 = a2;
              this.pab = p;
        }
```

```
  }
}
```

6. Ấn phải chuột vào phần tử gốc của cây Project trong cửa sổ Solution Explorer, chọn option Add.Class, đặt tên là B.cs để tạo ra file đặc tả class B. Khi cửa sổ hiển thị mã nguồn của class B hiển thị, đặc tả class B như đoạn code dưới đây :

```
//thêm lệnh using sau ở đầu file
using System.Runtime.Serialization;
namespace WRObject  {
  [Serializable]
  public class B {
        //định nghĩa các thuộc tính dữ liệu
        private    int intB1;
        private    double dblB2;
        private    A ba;
        private   A pba;
        private   A pba1;
        //định nghĩa các tác vụ
        public B() { }
public void init(int b1, double b2) {
  this.intB1 = b1; this.dblB2 = b2;
  ba = new A(); pba = new A(); pba1 = new A();
}
public void Setba (int a1, double a2, B b) {
  this.ba.init (a1,a2,b);
}
public void Setpba (int a1, double a2, B b) {
  this.pba.init (a1,a2,b);
}
public void Setpba1 (int a1, double a2, B b) {
  this.pba1.init (a1,a2,b);
}
} //hết class B
```

```
} //hết namespace WRObject
```

7. Chọn menu Debug.Start Debugging để dịch và chạy ứng dụng. Hệ thống đối tượng sẽ được tạo ra và lưu lên file c:\data.obj.
8. Hiển thị cửa sổ soạn mã nguồn file Program.cs, chú thích lệnh gọi Create_SaveObject(); và bỏ chú thích lệnh gọi ReadObject();. Dời chuột về lệnh "B b = (B) formatter.Deserialize(fs);" trong hàm ReadObject(), click chuột vào lệnh trái của lệnh này để thiết lập điểm dừng. Chọn menu Debug.Start Debugging để dịch và chạy ứng dụng. Ứng dụng sẽ dừng ở lệnh dừng. Dời chuột về biến b, ta thấy cửa sổ hiển thị giá trị của biến này lúc này là null.
9. Ấn F10 để thực hiện đúng lệnh này rồi dừng lại, dời chuột về biến b, rồi mở rộng cây phân cấp miêu tả nội dung biến b và thấy nó chứa đúng thông tin như slide 14, nghĩa là chương trình đã đọc được toàn bộ hệ thống đối tượng đã lưu giữ trước đây.

7.9 Kết chương

Chương này đã giới thiệu các đối tượng phục vụ ghi/đọc dữ liệu ra/vào file cùng các tác vụ ghi/đọc dữ liệu cổ điển ra/vào file.

Chương này cũng đã giới thiệu các đối tượng phục vụ ghi/đọc hệ thống đối tượng ra/vào file cùng các tác vụ ghi/đọc hệ thống đối tượng có mối quan hệ tham khảo phức tạp ra/vào file.