2

Lab

File và I/O Stream trong C#

File and Stream I/O in C#

Thực hành Lập trình mạng căn bản

GVHD: Nguyễn Xuân Hà

Lưu hành nội bộ

A. TỔNG QUAN

1. Mục tiêu

- Cung cấp khả năng khởi tạo, đọc, viết và khả năng cập nhật File.
- Hiểu được luồng thông tin (Stream) trong C#.
- Có thể sử dụng được các lớp FileStream, lớp StreamReader, lớp StreamWriter và lớp BinaryFormatter để đọc và viết các đối tượng vào trong các File.

2. Môi trường

- IDE Microsoft Visual Studio 2010 trở lên.

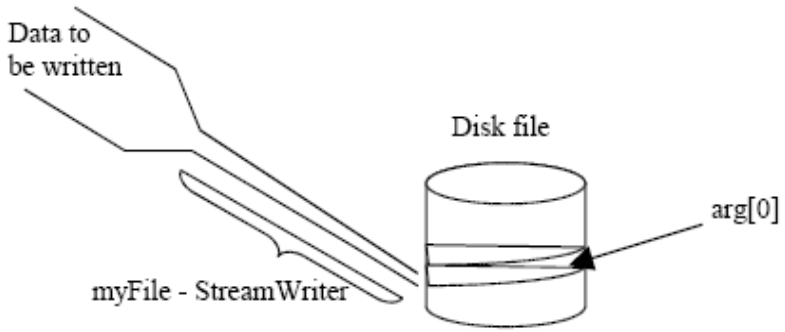
3. Liên quan

- Sinh viên cần nắm được các kiến thức nền tảng về lập trình. Các kiến thức này đã được giới thiệu trong các môn học trước và trong nội dung lý thuyết đã học do đó sẽ không được trình bày lại trong nội dung thực hành này.
- Tham khảo tài liệu (Mục E) để có kiến thức cơ bản về C#, Winforms.

B. KIẾN THỨC NỀN TẢNG

1. Luồng (Stream) và Tập tin (File) trong C#

Luồng (Stream) là luồng của thông tin, chứa thông tin sẽ được chuyển qua, còn **tập tin (File)** thì để lưu trữ thông tin, dữ liệu. **File** và Stream I/O (Input/Output) đề cập đến việc truyền dữ liệu đến hoặc từ một phương tiện lưu trữ. Trong .NET Framework, namespace **System.IO** bao gồm các loại có khả năng đọc và ghi (đồng bộ và không đồng bộ) đối với các luồng dữ liệu và file. Những namespace này cũng chứa các loại namespace thực hiện việc nén và giải nén các file.



Hình 1 : Mô tả thực hiện tạo tập tin và đưa dữ liệu vào

Dữ liệu được truyền theo hai hướng:

- Đọc dữ liệu: đọc dữ liệu từ bên ngoài vào chương trình.

- Ghi dữ liệu: đưa dữ liệu từ chương trình ra bên ngoài.

Streams

Lớp Stream hỗ trợ đọc và ghi byte. Tất cả các lớp đại diện cho các luồng đều kế thừa từ lớp Stream. Lớp Stream và các lớp dẫn xuất của nó cung cấp một cái nhìn chung về các nguồn dữ liệu và giúp lập trình viên không cần phải đi quá chi tiết về các đặc điểm của hệ điều hành và các thiết bị bên dưới.

Streams bao hàm ba thao tác cơ bản:

- **Đọc**: đưa dữ liệu từ một luồng vào một cấu trúc dữ liệu, chẳng hạn như một mảng byte.
- **Ghi**: đưa dữ liệu vào một luồng từ một nguồn dữ liệu.
- **Tìm kiếm**: truy vấn và sửa đổi vị trị hiện tại trong một luồng.

Dưới đây là một số các lớp stream thường được sử dụng:

- **FileStream** (System.IO.FileStream) – để đọc và ghi một file.
- **NetworkStream** (System.Net.Sockets.NetworkStream) – để đọc và ghi thông qua các socket mạng.

File và Thư mục

Chúng ta có thể sử dụng các loại dưới đây có trong namespace System.IO để tương tác với file và thư mục.

System.IO.File – cung cấp các phương thức tĩnh cho việc tạo, sao chép, xóa, di chuyển và mở file, cũng như giúp khởi tạo một đối tượng FileStream.

System.IO.FileInfo – cung cấp các phương thức instance để tạo, sao chép, xóa, di chuyển và mở file cũng như giúp khởi tạo một đối tượng FileStream.

System.IO.Directory – cung cấp các phương thức tĩnh để tạo, di chuyển và liệt kê thư mục và thư mục con.

System.IO.DirectoryInfo – cung cấp các phương thức instance để tạo, di chuyển và liệt kê thư mục và thư mục con.

System.IO.Directory – cung cấp các phương thức và thuộc tính để xử lý chuỗi thư mục theo cách đa nền tảng.

2. Các Class liên quan

a) FileStream:

Là một lớp dẫn xuất từ Stream, được sử dụng đọc và viết dữ liệu vào một file hay đọc và viết dữ liệu từ 1 file.

Ví dụ:

```
FileStream fs = new FileStream(FileName, mode);
```

Trong đó:

- FileName: tập tin mà chúng ta muốn truy xuất đến.
- Mode: chế độ mở file như thế nào (Append, Create, CreateNew, Open, OpenOrCreate…).

Ví dụ:

```
FileStream fs = new FileStream("thuchanh.txt", FileMode.CreateNew);
```

b) StreamReader và StreamWritter

Là các lớp dẫn xuất từ Stream, là luồng đọc tập tin. Để đọc file ta dùng lớp StreamReader. Để ghi file ta dùng lớp StreamWriter. Đây là lớp được dùng để viết và ghi 1 tập tin dạng văn bản.

```
StreamReader sr = new StreamReader(FileStream fileName);
StreamWriter sw = new StreamReader(FileStream fileName);
```

Ví dụ:

```
StreamReader sr = new StreamReader(fs);
```

c) BinaryStream:

Nếu chúng ta sử dụng một tập tin văn bản, thì khi chúng ta lưu dữ liệu kiểu số thì phải thực hiện việc chuyển đổi sang dạng chuỗi ký tự để lưu vào trong tập tin văn bản và khi lấy ra ta cũng lấy được giá trị chuỗi ký tự do đó ta phải chuyển sang dạng số. Đôi khi chúng ta muốn có cách thức nào đó tốt hơn để lưu trực tiếp giá trị vào trong tập tin và sau đó đọc trực tiếp giá trị ra từ tập tin.

Ví dụ: khi viết một số lượng lớn các số integer vào trong tập tin như là những số nguyên, thì khi đó ta có thể đọc các giá trị này ra như là số integer. Trường hợp nếu chúng được viết vào tập tin với dạng văn bản, thì khi đọc ra ta phải đọc ra văn bản và phải chuyển mỗi giá trị từ một chuỗi đến các số integer. Tốt hơn việc phải thực hiện thêm các bước chuyển đổi, ta có thể gắn một kiểu luồng nhị phân **BinaryStream** vào trong một tập tin, rồi sau đó đọc và ghi thông tin nhị phân từ luồng này.

Ghi chú: *Thông tin nhị phân là thông tin đã được định dạng kiểu lưu trữ dữ liệu.*

Ví dụ:

```
FileStream fs = new FileStream(sfd.FileName,FileMode.CreateNew);
BinaryWriter bw = new BinaryWriter(fs);
```

d) BinaryFormatter:

Sử dụng 2 phương thức Serialize và Deserialize để viết và đọc đối tượng từ trong luồng:

- ***Serialize***: chuyển đổi một đối tượng sang một định dạng, và có thể được viết vào File mà không mất dữ liệu.
- ***Deserialize***: đọc dữ liệu đã định dạng từ một File và chuyển nó về dạng ban đầu

Ví dụ:

Serialize

```
BinaryFormatter binaryFormatter = new BinaryFormatter();
FileStream fileName = File.Create("..\\student.txt");
binaryFormatter.Serialize(fileName, st);
```

Deserialize

```
BinaryFormatter bf = new BinaryFormatter();
FileStream fs = File.OpenRead("..\\student.txt");
Student student = (Student)bf.Deserialize(fs);
```

Lưu ý: Đối với các phiên bản mới nhất, ***BinaryFormatter*** đã bị obsolete vì các mối nguy hiểm tới vấn đề an toàn. Do đó, các bạn có thể thay thế bằng ***JsonSerializer*** với ví dụ dưới đây.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text.Json;
// Define a class to be serialized/deserialized
class Person
{
    public string Name { get; set; }
    public int Age { get; set; }

    public Person(string name, int age)
    {
        Name = name;
        Age = age;
    }

    public override string ToString()
    {
        return $"Name: {Name}, Age: {Age}";
    }
}

class Program
{
    static void Main(string[] args)
    {
        // Create some sample data
        List<Person> people = new List<Person>
        {
            new Person("Alice", 30),
            new Person("Bob", 25),
            new Person("Charlie", 40)
        };

        // Serialize the data to a file using JSON serialization
        SerializeToFileJson("people.json", people);

        // Deserialize the data from the file using JSON serialization
        List<Person> deserializedPeopleJson =
DeserializeFromFileJson<List<Person>>("people.json");

        // Display the deserialized data
        Console.WriteLine("Deserialized data using JSON serialization:");
        foreach (var person in deserializedPeopleJson)
        {
            Console.WriteLine(person);
        }

        // Serialize the data to a file using StreamWriter
        SerializeToFileStream("people.txt", people);
```

```
        // Deserialize the data from the file using StreamReader
        List<Person> deserializedPeopleStream =
DeserializeFromFileStream("people.txt");

        // Display the deserialized data
        Console.WriteLine("\nDeserialized data using StreamReader and
StreamWriter:");
        foreach (var person in deserializedPeopleStream)
        {
            Console.WriteLine(person);
        }
    }

    // Serialize an object to a file using JSON serialization
    static void SerializeToFileJson<T>(string filePath, T obj)
    {
        try
        {
            // Serialize the object to JSON format
            string json = JsonSerializer.Serialize(obj);

            // Write the JSON data to the file
            File.WriteAllText(filePath, json);

            Console.WriteLine($"Serialized data successfully written to
{filePath}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error occurred while serializing data:
{ex.Message}");
        }
    }

    // Deserialize an object from a file using JSON deserialization
    static T DeserializeFromFileJson<T>(string filePath)
    {
        try
        {
            // Read the JSON data from the file
            string json = File.ReadAllText(filePath);

            // Deserialize the JSON data to the specified type
            return JsonSerializer.Deserialize<T>(json);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error occurred while deserializing data:
{ex.Message}");
            return default;
        }
    }

    // Serialize an object to a file using StreamWriter
    static void SerializeToFileStream(string filePath, List<Person> people)
    {
```

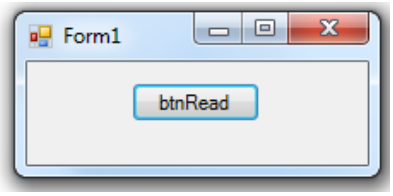
```
        try
        {
            // Create a new StreamWriter and FileStream
            using (StreamWriter writer = new StreamWriter(filePath))
            {
                // Write each person to the file
                foreach (var person in people)
                {
                    writer.WriteLine($"{person.Name},{person.Age}");
                }
            }

            Console.WriteLine($"Serialized data successfully written to
{filePath}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error occurred while serializing data:
{ex.Message}");
        }
    }

    // Deserialize an object from a file using StreamReader
    static List<Person> DeserializeFromFileStream(string filePath)
    {
        List<Person> people = new List<Person>();

        try
        {
            // Create a new StreamReader and FileStream
            using (StreamReader reader = new StreamReader(filePath))
            {
                string line;
                // Read each line from the file and create a Person object
                while ((line = reader.ReadLine()) != null)
                {
                    string[] parts = line.Split(',');
                    if (parts.Length == 2)
                    {
                        string name = parts[0];
                        int age = int.Parse(parts[1]);
                        people.Add(new Person(name, age));
                    }
                }
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error occurred while deserializing data:
{ex.Message}");
        }

        return people;
    }
}
```

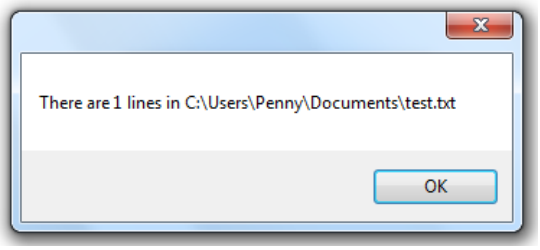
C. VÍ DỤ MINH HỌA

1. Chương trình đếm số dòng trong file

Khi nhấn vào button btnRead sẽ đếm và thông báo số dòng có trong một tập tin bất kỳ.

Giao diện minh họa.





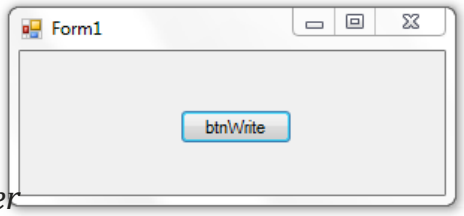
Gợi ý: bắt sự kiện cho nút btnRead, sử dụng lớp StreamReader.

```
private void btnRead_Click(object sender, System.EventArgs e)
{
   OpenFileDialog ofd = new OpenFileDialog();
   ofd.ShowDialog();
   FileStream fs = new FileStream(ofd.FileName, FileMode.OpenOrCreate);
   StreamReader sr = new StreamReader(fs);
   int lineCount = 0;
   while (sr.ReadLine() != null)
   {
      lineCount++;
   }
   fs.Close();
   MessageBox.Show("There are " + lineCount + " lines in " + ofd.FileName);
}
```

2. Chương trình ghi file nhị phân

Chương trình ghi thành file nhị phân với nội dung bất kỳ.

Hình minh họa



Gợi ý: bắt sự kiện cho nút btnWrite, sử dụng BinaryWriter

```
private void btnWrite_Click(object sender, EventArgs e)
{
   SaveFileDialog sfd = new SaveFileDialog();
   sfd.ShowDialog();
   FileStream fs = new FileStream(sfd.FileName,  FileMode.CreateNew);
   BinaryWriter bw = new BinaryWriter(fs);
   int[] myArray = new int[1000];
```

```
    for (int i = 0; i < 1000; i++)
    {
        myArray[i] = i;
        bw.Write(myArray[i]);
    }
    bw.Close();
}
```

D. BÀI TẬP

Các bài thực hành yêu cầu viết chương trình dưới dạng *Windows Forms App.*

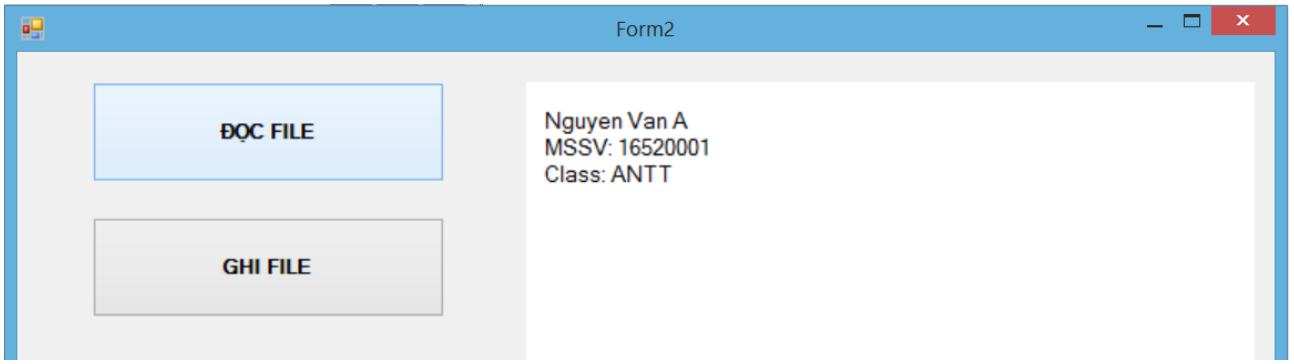
Sinh viên có thể tùy biến cách sắp xếp giao diện khác sao cho hợp lý.

Tất cả các bài thực hành đặt chung trong 1 Project duy nhất.

1. Bài 01 – Ghi và Đọc file

Viết chương trình đọc nội dung một file "input1.txt" và xuất ra màn hình. Sau đó ghi nội dung (chuyển toàn bộ ký tự sang kiểu in hoa) xuống file "output1.txt".

Giao diện minh họa:



Gợi ý:

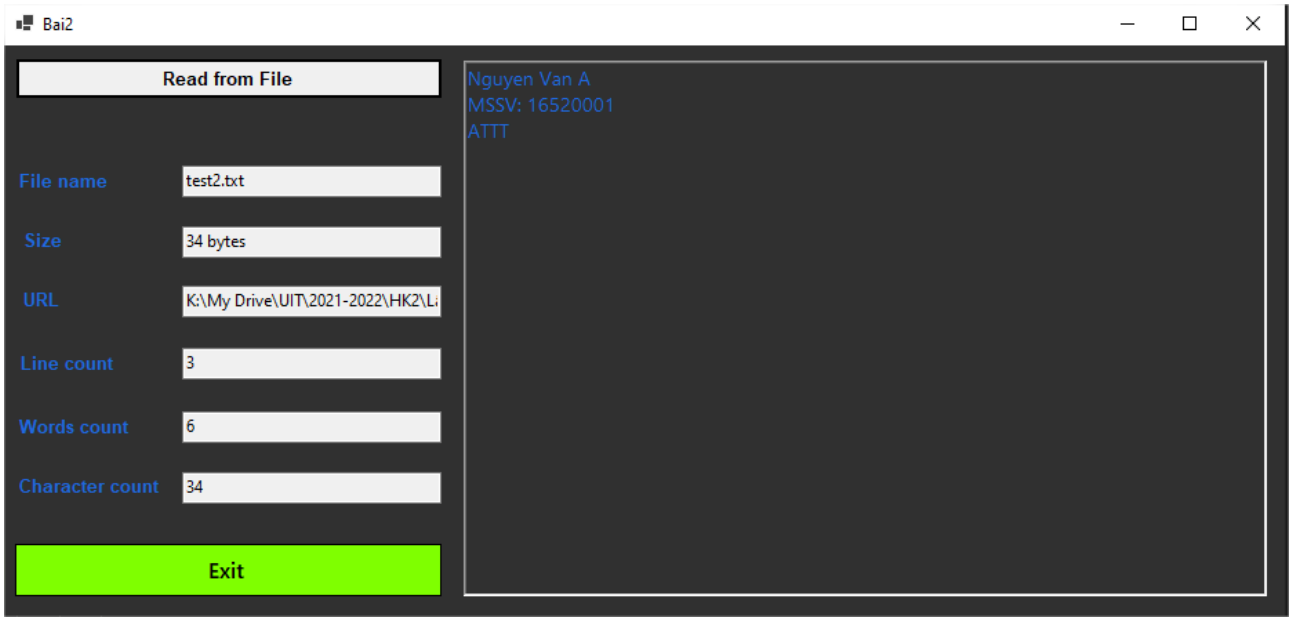
- Sử dụng hàm ReadToEnd để đọc toàn bộ dữ liệu và đẩy dữ liệu vào RichTextBox.
- Sử dụng StreamWritter để ghi nội dung xuống file.

2. Bài 02 – Đọc thông tin một file .txt

Viết chương trình đọc file và hiển thị các thông tin sau:

- Tên file
- Kích thước file
- Đường dẫn Url
- Số dòng, số từ, số ký tự
- Hiển thị nội dung của file

Giao diện minh họa.



Gợi ý: *bắt sự kiện click cho nút ĐỌC FILE, sử dụng lớp StreamReader*

- Có thể sử dụng .SafeFileName Property của OpenFileDialog để lấy tên file

  VD: name = ofd.SafeFileName.ToString();
- Có thể sử dụng .Name Property của FileStream để lấy đường dẫn

  VD: url = fs.Name.ToString();

3. Bài 03 - Đọc và Ghi file và tính toán

Đọc nội dung từ file "input3.txt" với nội dung theo định dạng, sau đó thực hiện các phép tính và ghi kết quả xuống file "output3.txt". Các phép tính bao gồm: cộng trừ, nhân, chia, ngoặc đơn. **Lưu ý:** Không sử dụng phương thức DataTable.Compute.

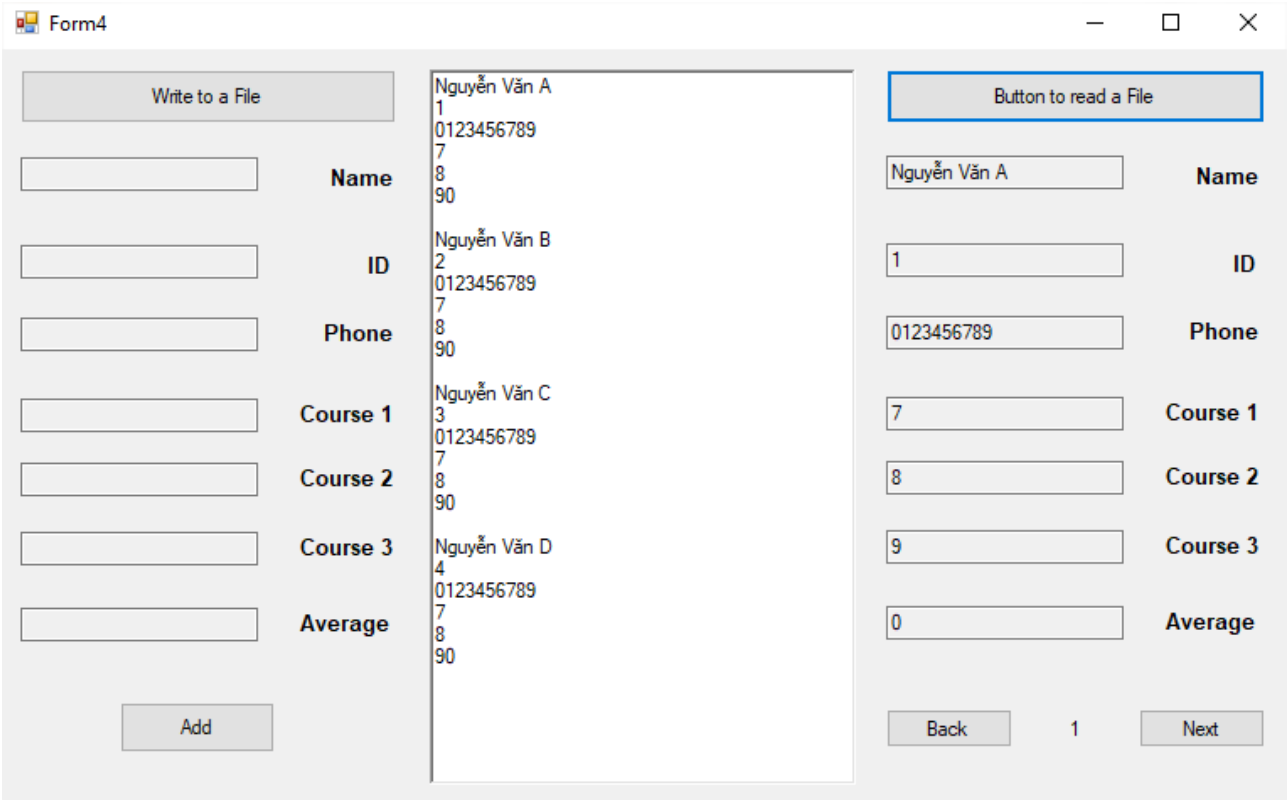
Ví dụ : Nội dung file "input3.txt" :	Nội dung file "output3.txt" :
1 + 2 + 3 + 4	1 + 2 + 3 + 4 = 10
12 – 7 – 5 + 2 – 3	12 – 7 – 5 + 2 – 3 = –1
2024 – 1 – 2 + 3	2024 – 1 – 2 + 3 = 2024
222 + 333 – 444 + 1	222 + 333 – 444.2 + 1 = 111.8

4. Bài 4 - Đọc và Ghi file sử dụng BinaryFormatter (JsonSerializer)

Viết chương trình sử dụng BinaryFormatter cho phép : Nhập 1 mảng các sinh viên (không nhập điểm trung bình) và ghi xuống file "input4.txt". Cấu trúc của Sinh viên như sau :

- Họ và tên : String
- MSSV : Int
- Điện thoại : String
- Điểm môn 1 : Float
- Điểm môn 2 : Float
- Điểm môn 3 : Float
- Điểm trung bình : Float

Đọc thông tin mảng Học Viên từ file "input4.txt" và tính điểm trung bình cho từng sinh viên sau đó ghi xuống file "output4.txt" và xuất ra màn hình.



Lưu ý: Khi xuất ra màn hình, cần tính điểm trung bình của 3 môn học. Ràng buộc điều kiện số điện thoại phải có 10 chữ số và bắt đầu bởi số 0. Mã số sinh viên là một số có 8 chữ số. Các điểm của từng học phần từ 0 đến 10. Có chức năng hiển thị số trang hiện tại và có thể điều chỉnh để xem lần lượt danh sách sinh viên.

5. **Bài 05 – Quản lý phòng vé** (phiên bản số 2)

Lấy ý tưởng từ bài 4 - bài thực hành số 1, tuy nhiên các thông tin của phòng vé sẽ được nhập bằng file "input5.txt" với cấu trúc:
<tên phim>
<giá vé chuẩn>

<phòng chiếu>
...
Viết chương trình cho phép nhập và ghi thành file, đọc file để lấy các thông tin cần thiết. Các yêu cầu tương tự, bổ sung tính năng xuất nội dung thống kê theo phim "output5.txt" bao gồm: tên phim, số lượng vé bán ra, số lượng vé tồn, tỉ lệ vé bán ra, doanh thu, xếp hạng doanh thu phòng vé. Trong đó, doanh thu là tổng số tiền thu được khi bán vé của phim đó, xếp hạng doanh thu phòng vé là xếp hạng phim có doanh thu theo thứ tự từ cao đến thấp.
Lưu ý: Tìm hiểu **ProgressBar** và thêm vào khi xuất file "output5.txt" để hỗ trợ người dùng dễ theo dõi tình trạng ứng dụng.

6. Bài 06 – Hôm nay ăn gì? (phiên bản số 2)

Lấy ý tưởng từ bài 8 - bài thực hành số 1, tuy nhiên cách thức nhập, đọc thông tin về món ăn được cấu trúc ở dạng như sau:
MonAn (**IDMA**, TenMonAn, HinhAnh, IDNCC)
NguoiDung (**IDNCC**, HoVaTen, QuyenHan)
Tìm hiểu về các sử dụng **SQLite**, thực hiện nhập dữ liệu trên vào cơ sở dữ liệu SQLite, thực hiện các câu truy vấn để trích xuất thông tin. Chương trình cho phép hiển thị danh sách các món ăn hiện tại được đọc từ cơ sở dữ liệu (sử dụng **ListView** hay **TreeView**...). Kết quả cuối cùng của ứng dụng là ngẫu nhiên chọn ra 1 món ăn, hình ảnh của món ăn đó và tên người đóng góp món ăn.

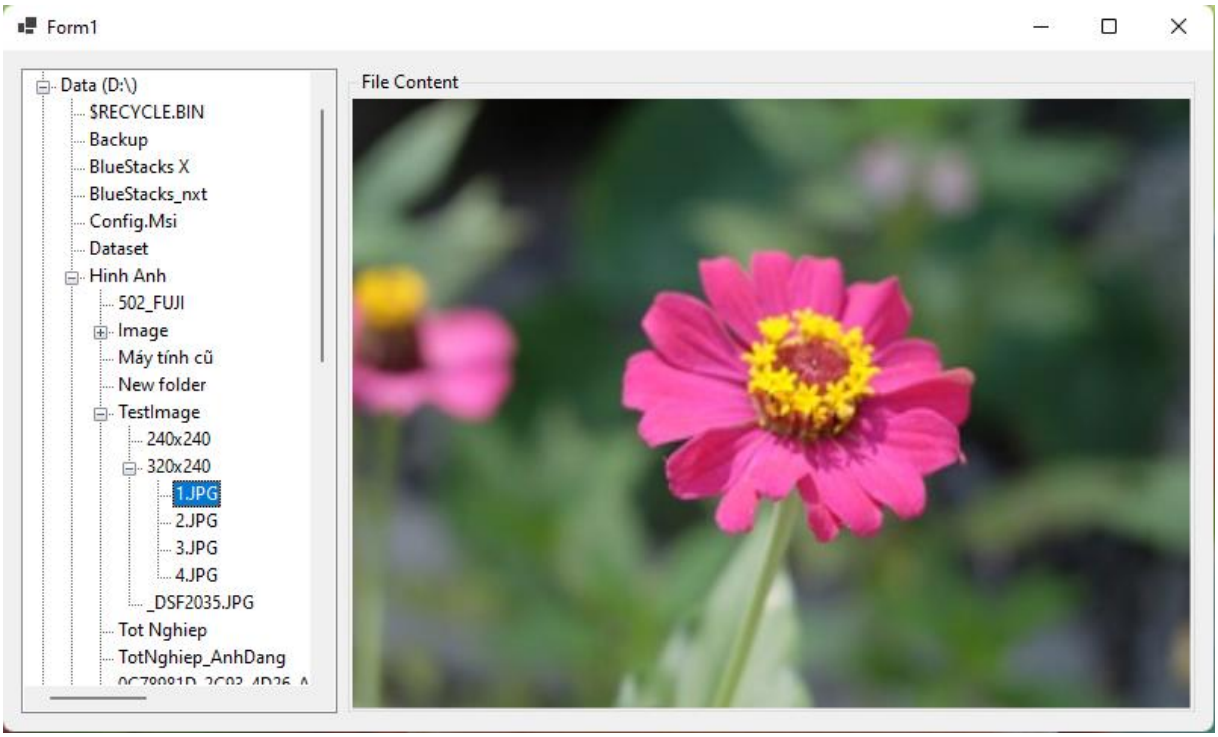
Gợi ý: Xem các hướng dẫn [4] và [5] trước

7. Bài 07 – Duyệt thư mục

Viết ứng dụng cho phép duyệt tất cả file có trong máy tính, hiển thị danh sách cái file, thư mục. Cho phép đi đến folder tiếp theo khi nhấp đúp chuột và hiển thị nội dung của file khi bấm chọn.

Giao diện minh họa


Windows (C:\)
Data (D:\)
Entertaiment (E:\)
Development (F:\)
File Content


Form1
Data (D:\)
$RECYCLE.BIN
Backup
BlueStacks X
BlueStacks_nxt
Config.Msi
Dataset
Hinh Anh
502_FUJI
Image
Máy tính cũ
New folder
TestImage
240x240
320x240
1.JPG
2.JPG
3.JPG
4.JPG
_DSF2035.JPG
Tot Nghiep
TotNghiep_AnhDang
File Content

E. YÊU CẦU & NỘP BÀI

1. Yêu cầu

- Các giao diện ở trên chỉ mạng tính chất minh họa, sinh viên tiến hành thiết kế giao của riêng mình đảm bảo các tiêu chí: dễ nhìn, thể hiện hết được các yêu cầu cần thực hiện, đẹp.
- Code "sạch" [2], đặt tên biến rõ ràng.
- Nộp bài không đầy đủ; lỗi, không chạy được; nộp trễ; sao chép code bạn khác, nguồn có sẵn: *xử lý tùy theo mức độ*.

2. Nộp bài

- Sinh viên thực hành và nộp bài cá nhân tại website môn học theo thời gian quy định.
- Source code được nộp tại GitHub và báo được nộp dưới định dạng:

Toàn bộ các file liên quan đặt vào 1 file nén (.zip) với tên theo quy tắc sau:

Mã lớp-LabX-MSSV

Ví dụ*: NT106.M21.MMCL.1-Lab02-25520001*

F. THAM KHẢO

[1] Microsoft (2018). C# Guide. [Online] Available at: https://docs.microsoft.com/en-us/dotnet/csharp/

[2] Martin, R. C. (2009). *Clean code: a handbook of agile software craftsmanship*. Pearson Education.

[3] Lập trình Winform cơ bản [Online] Available at: Lập trình Winform cơ bản | How Kteam

[4] Setup SQLite Setup SQLite Database in Visual Studio (14/23) (youtube.com)

[5] Sử dụng SQLite trong Winform Sqlite database in C# windows form (youtube.com)

HẾT