Bài 2. Dữ liệu có cấu trúc, phương thức và xử lý ngoại lệ

1. Cấu trúc mảng

1.1. Mảng một chiều

a. Cú pháp khai báo

+ Khai báo không khởi tạo kích thước và giá trị:

<Kiểu DL>[ ] Tên_mảng;

+ Khai báo có khởi tạo KT nhưng không khởi tạo giá trị:

<Kiểu DL>[ ] Tên_mảng = new <Kiểu DL>[<số PT>];

+ Khai báo có khởi tạo KT và khởi tạo giá trị:

<Kiểu DL>[ ] Tên_mảng = new <Kiểu DL>[<Số PT>]
{giá trị 1, giá trị 2, giá trị 3, ...};

hoặc:

<Kiểu DL>[ ] Tên_mảng = {giá trị 1, giá trị 2, giá trị 3, ...};

Ví dụ 1:

```
int[ ] a;
int[ ] a = new int[10];
int[ ] b = new int[5]{2,10,4,8,5};
int[ ] b = {2, 10, 4, 8, 5};
```

b. Cách sử dụng

Để truy cập vào từng phần tử trong mảng ta sử dụng:

Tên_mảng[chỉ số]

- Chỉ số của phần tử đầu tiên là 0.
- Thuộc tính *Length* của mảng cho biết số PT trong một mảng.

c. Duyệt mảng 1 chiều:

```
for (int i=0; i <  ArrayX.Length; i++) {
    xử lý ArrayX[i];
}
```

Có thể thay **for** bằng **foreach** như sau:

```
foreach (int phantu in ArrayX){
    xử lý phantu;
}
```

Ví dụ 2:

```
int [] a = {6,4,2,5,7,3};
//in danh sach cac phan tu trong array
for (int i = 0; i < a.Length; i++)
    Console.WriteLine(a[i]);
//hoặc
```

```
    foreach (int ia in a)
        Console.WriteLine(ia);
```

2. Mảng hai chiều (Ma trận)

2.1. Cú pháp khai báo

<Kiểu DL>[,] Tên_mảng;

hoặc:

<Kiểu DL>[,] Tên_mảng = new <Kiểu DL>[hàng, cột];

hoặc:

<Kiểu DL>[,] Tên_mảng = new <Kiểu DL>[,]{<các giá trị>};

hoặc

<Kiểu DL>[,] Tên_mảng = {<các giá trị>};

Ví dụ 3: Viết chương trình cho phép người dùng nhập vào một mảng n số nguyên (0<n<=100):

+ In mảng vừa nhập ra màn hình.

+ Tìm phần tử lớn nhất, in ra.

Giải:

```
      int n;
 do
     {Console.Write("Vào số phần tử của mảng:");
          n = int.Parse(Console.ReadLine());}
while (n<=0||n>100);
       int[] A = new int[n];
for (int i = 0; i < n; i++)
    { Console.Write("Nhập giá trị cho phần tử thứ {0}:", i + 1);
       A[i] = int.Parse(Console.ReadLine());  }
for (int i = 0; i < A.Length; i++)
       Console.WriteLine(A[i]);
  int  max;  max = A[0]; // phần tử đầu tiên
for (int i = 0; i < A.Length; i++)
      {    if (max < A[i])     max = A[i];}
Console.Write("Phần tử lớn nhất là:{0} ", max);
Console.ReadLine();
```

2. Tập hợp (Collections)

Đối với nhiều ứng dụng, ta muốn tạo và quản lý các nhóm đối tượng liên quan. Có hai cách để nhóm các đối tượng: bằng cách tạo mảng đối tượng và bằng cách tạo tập hợp các đối tượng.

Mảng hữu ích nhất để tạo và làm việc với một số lượng cố định các đối tượng có cùng kiểu. Collection cung cấp một cách linh hoạt hơn để làm việc với các nhóm đối tượng. Không giống như mảng, nhóm đối tượng mà ta làm việc có thể mở rộng và thu hẹp một cách linh hoạt khi nhu cầu của ứng dụng thay đổi. Đối với một số collection, ta có thể gán khóa cho bất kỳ đối tượng nào mà ta đưa vào collection để có thể nhanh chóng truy xuất đối tượng bằng cách sử dụng khóa.

Collection là một lớp, vì vậy ta phải khai báo một thể hiện của lớp trước khi ta có thể thêm các phần tử vào collection đó.

Nếu collection chỉ chứa các phần tử của một kiểu dữ liệu, ta có thể sử dụng một trong các lớp trong namespace ***System.Collections.Generic***. Khi truy xuất một phần tử từ một Generic collection, ta không phải xác định kiểu dữ liệu của nó hoặc chuyển đổi kiểu dữ liệu cho nó.

Mảng	Collection
Giống nhau: Cả hai đều có thể lưu trữ nhiều phần tử, có thể là kiểu giá trị hoặc kiểu tham chiếu	
Khác nhau: - Kích thước cố định - Các phần tử có cùng kiểu dữ liệu	- Kích thước có thể thay đổi được - Các phần tử có thể khác kiểu dữ liệu

Các lớp collection thông dụng

Tên Collection	Mô tả
List<T>	Các phần tử trong collection có cùng kiểu T. Sử dụng chỉ số để truy cập đến các phần tử.
ArrayList	Các phần tử trong collection có thể khác kiểu nhau. Sử dụng chỉ số để truy cập đến các phần tử.
SortedList<K,V>	Sử dụng khóa để truy cập tới giá trị, giá trị có thể là kiểu đối tượng bất kỳ
Queue<T>	Hàng đợi
Stack<T>	Ngăn xếp

2.1. List<T>

List trong C# được sử dụng để lưu trữ và quản lý một danh sách là tập hợp các đối tượng theo kiểu mảng. Nó có thể được truy cập bằng chỉ mục và có các phương pháp sắp xếp, tìm kiếm và sửa đổi danh sách.

List nằm trong **System.Collections.Generic**

❖ **Các thuộc tính, phương thức của List:**

+ Thuộc tính:

Thuộc tính	Mô tả
Items	Nhận hoặc đặt phần tử ở chỉ mục được chỉ định
Count	Trả về tổng số phần tử trong List

+ Phương thức:

Phương thức	Mô tả
Add()	Thêm một phần tử vào cuối Danh sách
Clear()	Loại bỏ tất cả các phần tử khỏi danh sách
Contains()	Kiểm tra xem phần tử được chỉ định có tồn tại hay không trong danh sách
Find()	Tìm phần tử đầu tiên được chỉ định
Foreach()	Lặp lại qua một danh sách
Insert(i,x)	Chèn một phần tử **X** tại chỉ mục **i** trong danh sách
Remove()	Xóa phần tử đầu tiên trong danh sách
RemoveAt(i)	Xóa phần tử tại chỉ mục i trong danh sách

❖ Các thao tác với List:

• Tạo List

Để tạo một **List** ta sử dụng từ khóa **new** cùng với đó là kiểu dữ liệu cho tham số cần lưu trữ.

Các ví dụ:

```
List<int> primeNumbers = new List<int>();
//thêm phần bằng phương thức Add
primeNumbers.Add(1);
primeNumbers.Add(3);
primeNumbers.Add(5);
primeNumbers.Add(7);

var cities = new List<string>();
cities.Add("New York");
cities.Add("London");
cities.Add("Mumbai");
cities.Add("Chicago");
cities.Add(null);
```

```
//thêm phần tử bằng cách khởi tạo
var bigCities = new List<string>()
{
  "New York",
  "London",
  "Mumbai",
  "Chicago"
};
var students = new List<Student>() {
  new Student(){ Id = 1, Name="Nam"},
  new Student(){ Id = 2, Name="Phong"},
  new Student(){ Id = 3, Name="Quyền"},
  new Student(){ Id = 4, Name="Tiến"}
};
```

- **Truy cập List**

Danh sách có thể được truy cập bằng chỉ mục (index), vòng lặp for / foreach và sử dụng các truy vấn LINQ. Các chỉ mục của danh sách bắt đầu từ số 0.

```
//khai báo và khởi tạo phần tử cho intList
List<int> intList = new List<int>() { 10, 20, 30, 40, 50 };
//sử dụng foreach để truy xuất toàn bộ phần tử
intList.ForEach(el => Console.WriteLine(el));
foreach (var el in intList)
  Console.WriteLine(el);
//sử dụng for để truy xuất phần tử
for(int i =0; i < intList.Count; i++)
  Console.WriteLine(intList[i]);
```

- **Chèn phần tử vào List**

Sử dụng phương thức Insert() để chèn phần tử vào vị trí index được chỉ định.

```
var numbers = new List<int>(){ 10, 20, 30, 40 };
// chèn phần tử 11 vào vị trí 1
numbers.Insert(1, 11);
//->> {10, 11, 20, 30, 40}
```

- **Xóa phần tử khỏi List**

Sử dụng phương thức Remove() để xóa phần tử xuất hiện đầu tiên trong List. Hoặc sử dụng phương thức RemoveAt() để xóa phần tử tại một vị trí index được chỉ định.

```
var numbers = new List<int>(){10,20,30,40,10};
//xóa phần tử 10 trong danh sách
numbers.Remove();
//xóa phần tử thứ 2 trong danh sách
numbers.RemoveAt(2);
```

- **Ví dụ (dùng List):** Viết chương trình cho phép người dùng nhập vào một danh sách n số nguyên (0<n<=100):
  + In danh sách vừa nhập ra màn hình.
  + Tìm phần tử lớn nhất, in ra.

```
int n;
do
{
    Console.Write("Vào số phần tử của List:");
    n = Int32.Parse(Console.ReadLine());
}
while (n <= 0 || n > 100);
 //int[] A = new int[n];
List<int> A = new List<int>();
for (int i = 0; i < n; i++)
{
    Console.Write("Nhập giá trị cho phần tử thứ {0}:",i+1);
    //A[i]=int.Parse(Console.ReadLine());
    A.Add(int.Parse(Console.ReadLine()));
}
for (int i = 0; i < A.Count; i++)
    Console.WriteLine(A[i]);
int max; max = A[0]; // phần tử đầu tiên
for (int i = 0; i < n; i++)
{ if (max < A[i]) max = A[i]; }
Console.Write("Phần tử lớn nhất là:{0} ", max);
Console.ReadLine();
```

3. Thao tác với file văn bản

- Namespace **System.IO** cung cấp các class để làm việc với file, quản lý thư mục
- **File văn bản**: file chứa các ký tự (chuỗi) văn bản. Các trường (field) được tách bằng các ký tự đặc biệt như **tab** hoặc |, các bản ghi (record) được phân biệt bằng ký tự xuống dòng mới.
- **Luồng dữ liệu** (stream) là dòng dữ liệu đi từ nơi này đến nơi khác. Để ghi dữ liệu sử dụng luồng ra, để đọc dữ liệu sử dụng luồng vào.

3.1. Ghi dữ liệu ra file sử dụng StreamWriter

❶ Khai báo và khởi tạo đối tượng **StreamWriter**

StreamWriter tên_đối_tượng = new StreamWriter("tên_file"[,Boolean]);

Boolean = true : ghi nối; = false: ghi đè

❷ Sử dụng phương thức **WriteLine** của đối tượng **StreamWriter** để copy dữ liệu vào **buffer** trong bộ nhớ

tên_đối_tượng.WriteLine(dữ_liệu_ghi_ra_file);

❸ Gọi phương thức **Close** của đối tượng **StreamWriter** để ghi dữ liệu từ buffer sang file và giải phóng tài nguyên.

tên_đối_tượng.Close();

3.2. Đọc dữ liệu từ file sử dụng StreamReader

❶ Khai báo và khởi tạo đối tượng StreamReader

StreamReader tên_đối_tượng= new StreamReader("tên_file");

❷ Sử dụng phương thức **ReadLine** để đọc dữ liệu. Cần sử dụng vòng lặp để kiểm tra đến cuối file để đọc nhiều phần tử.

- **Phương thức ReadLine:** *tên_đối_tượng.ReadLine();*

• **Kiểm tra đọc đến cuối file dùng phương thức Peek()**

- **Peek()** kiểm tra phần tử tiếp theo
- Trả về giá trị -1 nếu đã kiểm tra qua phần tử cuối

❸ Đóng luồng sử dụng phương thức **Close** của đối tượng StreamReader

tên_đối_tượng.Close();

Ví dụ 1: cho file văn bản có tên là **input.txt**. Chương trình sau sử dụng đối tượng StreamReader để đọc file **input.txt** in ra màn hinh.

System.IO

```
StreamReader sr = new StreamReader(@"E:\ViDu\input.txt");
    while (sr.Peek() != -1)
      {
        Console.WriteLine(sr.ReadLine());
      }
    sr.Close();
    Console.ReadLine();
```

Ví dụ 2: ghi tiếp dữ liệu vào file **input.txt**

```
StreamWriter sw = new StreamWriter(@"E:\ViDu\input.txt", true);

   sw.WriteLine("Ghi tiep dong nay vao file");
   Console.WriteLine("Ghi file thanh cong!");
   sw.Close();
   Console.ReadLine();
```

4. Phương thức

Phương thức (method) là một chuỗi các câu lệnh được đặt tên, nhằm thực hiện các tác vụ cụ thể nào đó.

- Có thể có tham số hoặc không
- Có thể trả về hoặc không trả về giá trị
- Được thực thi bằng cách gọi tên phương thức và cung cấp các đối số cần thiết

- **Phương thức trả về giá trị:**

```
[private/public] static <kiểu> <Tên_phương_thức>([danh_sách_tham_số])
  {
    <Khối lệnh>;
    return(giá trị);
  }
```

- **Phương thức không trả về giá trị (kiểu void)**

```
[private/public] static void <Tên_phương_thức>([danh_sách_tham_số])
  {
    <Khối lệnh>;
  }
```

Trong đó:

- **private**: phương thức chỉ có thể được gọi từ một phương thức khác trong cùng lớp.
- **public**: phương thức có thể được gọi từ bên ngoài lớp.
- **static:** để phương thức được gọi trực tiếp từ hàm **Main()**

• Gọi phương thức:

+ Phương thức không trả về giá trị: ***Tên_phương_thức ([danh_sách_đối_số])***

+ Phương thức có trả về giá trị: ***Biến=Tên_phương_thức ([danh_sách_đối_số])***

Ví dụ 1:

```
public static double TinhLuong()
      {
         double kq=HSLuong*850000;
         return kq;
      }
  //gọi phương thức
    double kq = TinhLuong();
```

Ví dụ 2:

```
public static int Cong(int so1, int so2)
  {
      return so1+ so2;
  }
  //gọi phương thức
     int kq = Cong(2, 4);
```

Ví dụ 3:

```
public static void Chao()
  {
     Console.WriteLine("Chào các bạn!");
  }
 //gọi phương thức
      Chao();
```

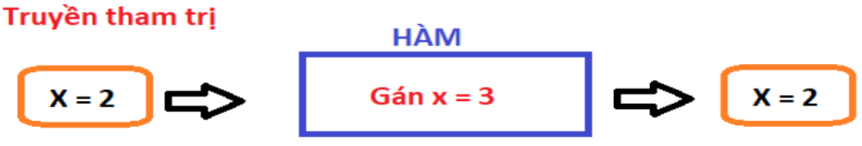
• Tham số và đối số

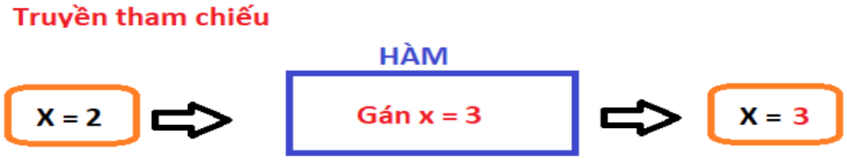
- Định nghĩa phương thức xác định tên và kiểu của các tham số (parameter).
- Khi gọi phương thức, ta cung cấp các giá trị cụ thể gọi là đối số (argument) cho mỗi tham số.
- Các đối số phải tương thích với các kiểu của tham số.

• Truyền tham số cho phương thức

Các tham số của phương thức có thể được truyền theo 2 cách: tham trị và tham chiếu.

- **Tham trị**: một bản sao của biến sẽ được tạo ra, sao chép giá trị của biến, truyền biến đã





được sao chép này vào hàm, dù có thực hiện bao nhiêu phép tính toán cũng không ảnh hưởng đến biến gốc.

- **Tham chiếu**: truyền ngay địa chỉ của biến lưu trên bộ nhớ vào hàm (hay hiểu cách khác là truyền chính biến đó vào hàm), khi thực hiện tính toán thì giá trị biến này thay đổi theo.

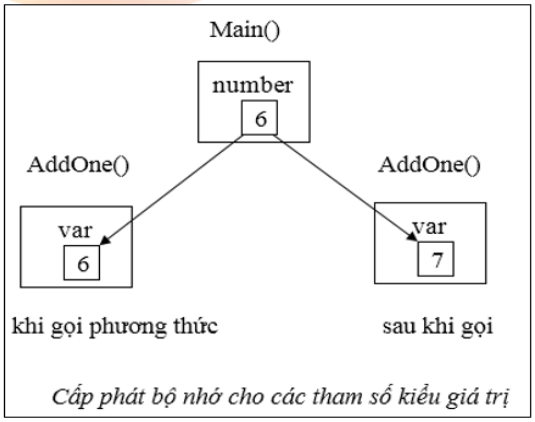
+ Cú pháp khai báo tham số :

[ref] <kiểu_dữ_liệu> tên_tham_số

Ví dụ 4: Truyền tham số theo giá trị

- Ví dụ:

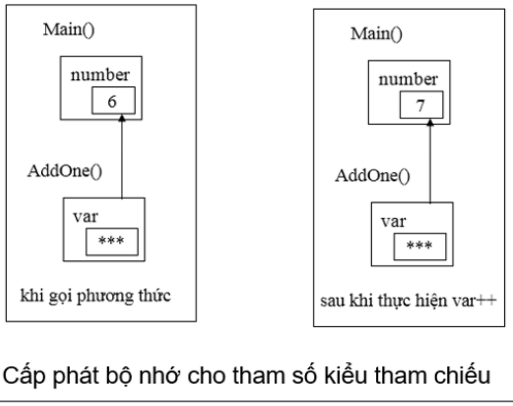
```
static void Main(string[] args)
{
      int number = 6;
      AddOne(number);
      Console.WriteLine(number);
}
static void AddOne(int var)
{
      var++;
}
```



Ví dụ 5: Truyền tham số theo tham chiếu

- Ví dụ:

```
static void Main(string[] args)
{
      int number = 6;
      AddOne(ref number);
      Console.WriteLine(number);
}
static void AddOne(ref int var)
{
      var++;
}
```



Ví dụ 6:(method) Viết chương trình nhập vào 2 số và tính tổng của 2 số. Với 2 phương thức có tham số.

+ Phương thức **Nhap** (a,b là các số nhập từ bàn phím)

+ Phương thức **TinhTong**

+ Trong phương thức Main gọi 2 phương thức trên

Cách 1:

```
class Program
   {

        static void Main(string[] args)
        {
            double a = 0, b = 0;
            Nhap(ref a, ref b);
            TinhTong(a, b);
            Console.Read();
        }
        private static void Nhap(ref double x, ref double y)
        {
            Console.Write("Nhap vao a: ");
            x = double.Parse(Console.ReadLine());
            Console.Write("Nhap vao b: ");
            y = double.Parse(Console.ReadLine());
        }
        private static void TinhTong(double a, double b)
        {
            Console.Write("Tổng 2 số là:{0}", a + b);
        }
    }
```

Cách 2:

```
class Program
   {

         static void Main(string[] args)
          {
             double a = 0, b = 0;
             Nhap(ref a, ref b);
            double T=TinhTong(a, b);
            Console.Write("Tổng 2 số là: {0} ", T);
            Console.Read();
          }
         private static void Nhap(ref double x, ref double y)
          {
             Console.Write("Nhập vào số thứ 1: ");
             x = double.Parse(Console.ReadLine());
             Console.Write("Nhập vào số thứ 2: ");
             y = double.Parse(Console.ReadLine());
          }
         private static double TinhTong(double a, double b)
          {
                return a + b;
          }
       }
```

5. Xử lý ngoại lệ

Ngoại lệ (Exception) là khái niệm được sử dụng trong các ngôn ngữ lập trình bậc cao dùng để chỉ những biến cố không mong đợi khi chạy chương trình. Bản chất ngoại lệ là những lỗi xảy ra trong quá trình thực thi (runtimed error). Trong .NET cung cấp một cơ chế xử lý ngoại lệ (gọi là cơ chế try/catch)

Cú pháp:

```
try {
    // các lệnh có thể gây ra ngoại lệ (exception)
    }
catch( tên_ngoại_lệ e1 )
  { // phần code để xử lý lỗi }
[catch( tên_ngoại_lệ e2 )
  { // phần code để xử lý lỗi }
catch( tên_ngoại_lệ eN )
  { // phần code để xử lý lỗi } ]
```

Các lệnh có nguy cơ gây lỗi được đặt trong khối **try**. Các lệnh này có thể bao gồm các lệnh chuyển đổi kiểu dữ liệu (từ kiểu chuỗi sang kiểu số), lệnh liên quan đến nhập xuất và thậm chí cả toán tử chia… Khi gặp một lệnh gây ra lỗi trong khối try, chương trình thực thi sẽ lập tức triệu gọi các lệnh trong khối **catch** tương ứng để thực thi. Khối **catch** là khối lệnh thực hiện các ứng xử khi có lỗi, trong khối này, thông thường người lập trình thực hiện việc hiển thị lỗi. Khi xây dựng khối catch, đầu tiên phải khai báo loại ngoại lệ (Exception-type). Tất cả Exception-type là những lớp kế thừa từ lớp Exception. Trong mỗi lớp đều có phương thức ToString(), phương thức này được dùng để hiển thị thông tin lỗi cho người sử dụng.

Các ngoại lệ hay dùng

Tên ngoại lệ	Mô tả
MethodAccessException	Lỗi truy cập, do truy cập đến thành viên hay phương thức không được truy cập
ArgumentException	Lỗi tham số đối mục
ArgumentNullException	Đối mục Null, phương thức được truyền đối mục null không được chấp nhận
ArithmeticException	Lỗi liên quan đến các phép toán
ArrayTypeMismatchException	Kiểu mảng không hợp, khi cố lưu trữ kiểu không thích hợp vào mảng
DivideByZeroException	Lỗi chia zero
FormatException	Định dạng không chính xác một đối mục nào đó

IndexOutOfRangeException	Chỉ số truy cập mảng không hợp lệ, dùng nhỏ hơn chỉ số nhỏ nhất hay lớn hơn chỉ số lớn nhất của mảng
InvalidCastException	Phép gán không hợp lệ
MulticastNotSupportedException	Multicast không được hỗ trợ, do việc kết hợp hai uỷ quyền không đúng
NotFiniteNumberException	Không phải số hữu hạn, số không hợp lệ
NotSupportedException	Phương thức không hỗ trợ, khi gọi một phương thức không tồn tại bên trong lớp.
NullReferenceException	Tham chiếu null không hợp lệ.
OutOfMemoryException	Out of memory
OverflowException	Lỗi tràn phép toán
StackOverflowException	Tràn stack
TypeInitializationException	Kiểu khởi tạo sai, khi bộ khởi dựng tĩnh có lỗi.

Ví dụ 1: Tính bình phương của một số nguyên được nhập từ bàn phím. Nếu dữ liệu nhập vào không phải số nguyên thì chương trình sẽ báo lỗi.

```
static void Main(string[] args)
  {
      Console.WriteLine("Nhap 1 so x:");
      string temp = Console.ReadLine();
      int x = 0;
      try
      {
          x = int.Parse(temp);
          Console.WriteLine("x^2= " + x*x);
      }
      catch (FormatException e)
      {
          Console.WriteLine("So nguyen nhap vao khong dung dinh dang!");
          Console.WriteLine("Error:" + e.ToString());
      }
      Console.ReadLine();
  }
```

Ví dụ 2: Tính số dư khi chia số nguyên x cho số nguyên y, hai số nguyên này được nhập từ bàn phím. Trong ví dụ này, chương trình có hai lệnh có khả năng sinh lỗi. Lệnh thứ nhất là lệnh chuyển kiểu khi thực hiện việc chuyển từ kiểu chuỗi ký tự sang kiểu số nguyên, lệnh này gây ra ngoại lệ khi chuỗi ký tự được nhập vào không đúng định dạng số nguyên. Lệnh thứ hai là lệnh thực hiện phép chia lấy số dư (%), lệnh này gây ra ngoại lệ khi số chia bằng 0.

```
static void Main(string[] args)
   {
       Console.WriteLine("Nhap so nguyen thu nhat:");
       string tempX = Console.ReadLine();
       Console.WriteLine("Nhap so nguyen thu hai:");
       string tempY = Console.ReadLine();
       int x = 0; int y = 0;
       try
       {
           x = int.Parse(tempX);
           y = int.Parse(tempY);
           int z = 0;
           z = x % y;
           Console.WriteLine("x % y = " + z);
```

```
    }
    catch (FormatException e1)
    {
        Console.WriteLine("So nguyen nhap vao khong dung dinh dang!");
        Console.WriteLine("Error:" + e1.ToString());
    }
    catch (DivideByZeroException e2)
    {
        Console.WriteLine("Khong the chia cho so 0!");
        Console.WriteLine("Error:" + e2.ToString());
    }
    Console.ReadLine();
}
```