PHỤ LỤC
Phụ lục A - CƠ BẢN VỀ NGÔN NGỮ C# (C - Sharp)

I. Tạo ứng dụng trong C#

Ví dụ dưới đây là một ứng dụng dòng lệnh (console application) đơn giản. Ứng dụng này giao tiếp với người dùng thông qua bàn phím, màn hình DOS và không có giao diện đồ họa người dùng, giống như các ứng dụng thường thấy trong Windows.

Ứng dụng dòng lệnh trong ví dụ sau sẽ xuất chuỗi "Hello World" ra màn hình. Thông qua ví dụ này chúng ta nắm được cấu trúc của một chương trình trong C#.

Ví dụ:

```
using System;  //Khai báo thư viện
class HelloWorld    //Khai báo lớp
{
     //Định nghĩa hàm Main
     static void Main( ) {

          //Xuất câu thông báo "Hello ra màn hình"
          System.Console.WriteLine("Hello World");

          //Chờ người dùng gõ một phím bất kỳ để
thoát          System.Console.ReadLine();
     }
}
```

Chương trình trên khai báo một kiểu đơn giản: lớp ***HelloWorld*** bằng từ khóa ***class,*** được bao bởi cặp dấu {}, trong đó có một phương thức (hàm) tên là ***Main().*** Khi chạy chương trình, hàm ***Main()*** được thực thi đầu tiên nên mỗi chương trình chỉ có duy nhất một hàm ***Main().*** Ngoài hàm ***Main()*** lớp ***HelloWorld*** có thể có thêm các hàm khác.

Mỗi lớp có các dữ liệu (thuộc tính) và các hành vi đặc trưng của lớp đó. Thuộc tính là các thành phần dữ liệu mà mọi đối tượng thuộc lớp đều có. Hành vi là phương thức của lớp (còn gọi là hàm thành viên), đó là các hàm thao tác trên các thành phần dữ liệu của lớp. Trong ví dụ này, lớp ***HelloWorld*** không có thuộc tính dữ liệu và hành vi nào (trừ hàm ***Main()***).

Trong ví dụ này, ta sử dụng đối tượng ***Console*** để thao tác với bàn phím và màn hình. Đối tượng ***Console*** thuộc không gian (name space, thư viện) *System* vì vậy ta sử dụng chỉ thị ***using System*** ở đầu chương trình.

Để truy cập đến một thành phần của lớp hoặc của đối tượng ta dùng toán tử chấm "**.**". Lệnh ***System.Console.WriteLine("Hello World");*** có nghĩa là gọi hàm ***WriteLine*** của đối tượng ***Console*** trong thư viện ***System*** để xuất chuỗi ***"Hello World"*** ra màn hình.

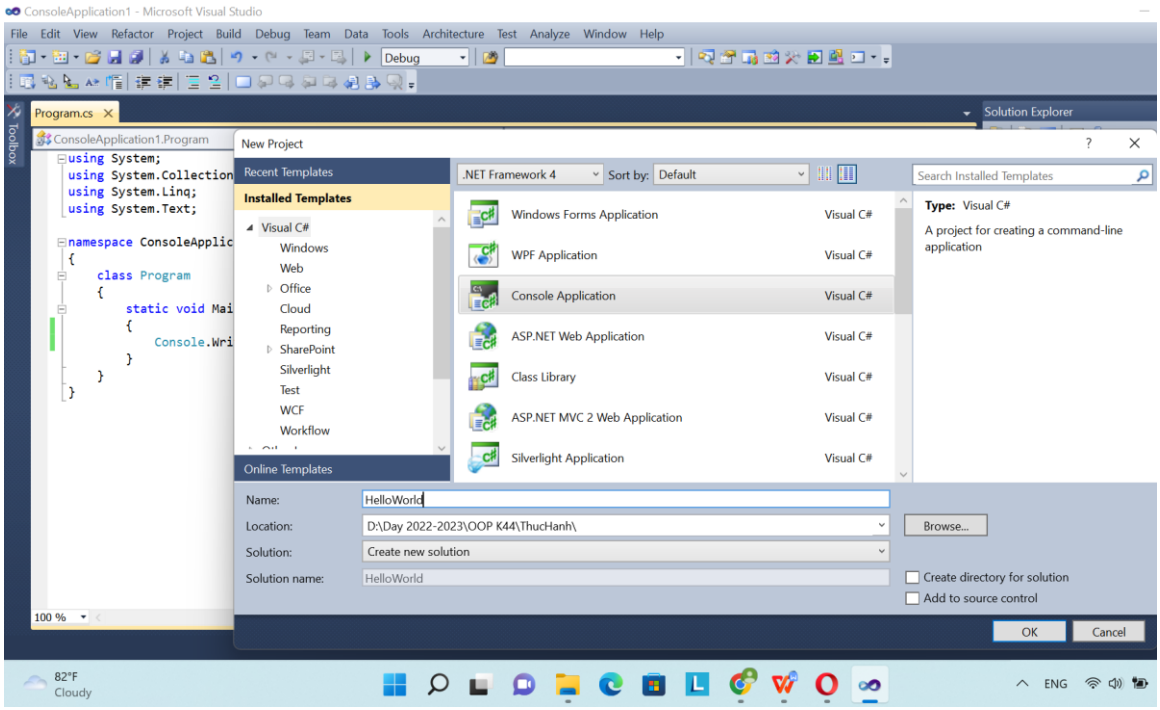
Lệnh ***System.Console.ReadLine();*** thực chất dùng để nhập một chuỗi từ bàn phím. Trong trường hợp này nó có tác dụng chờ người dùng nhấn phím Enter để kết thúc chương trình.

Chú ý:

Phần 1 này trình bày các chương trình theo phương pháp lập trình thủ tục truyền thống nhằm làm quen với ngôn ngữ C# vì vậy, các ví dụ không được trình bày theo phương pháp hướng đối tượng.

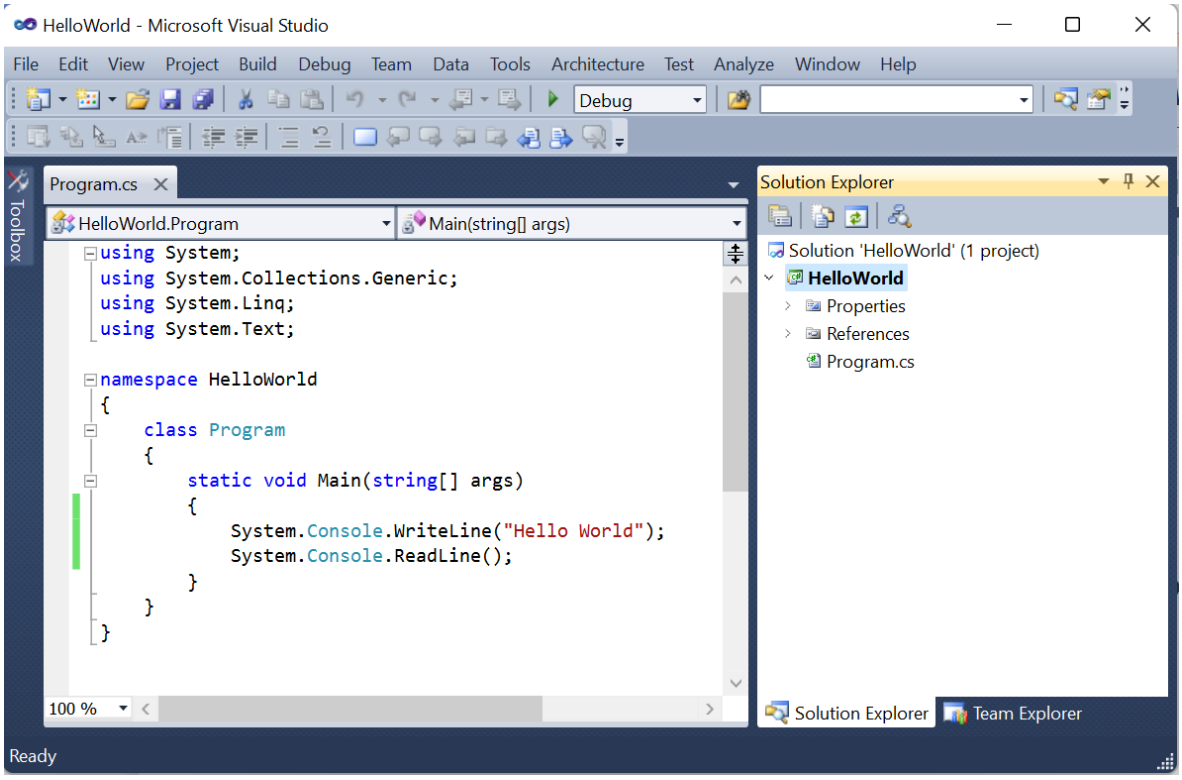
I.1. Soạn thảo chương trình "Hello World"

- Khởi động MS Visual Studio.
- Tạo ứng dụng dòng lệnh tên là Hello World qua các bước sau: **File\New \Project.** Chọn **Visual C# Project** trong ô **Project Types** và chọn **Console Application** trong ô **Templates** như hình dưới đây. Nhập vào tên dự án là ***HelloWorld*** vào ô **Name** và đường dẫn để lưu trữ dự án vào ô **Location** (ví dụ, D:\Day 2022-2023\OOP K44\ThucHanh\).



Hình I-1: Tạo ứng dụng C# Console Application trong Visual Studio.

- Sau đó đưa lệnh ***`System.Console.WriteLine("Hello World");`*** vào trong phương thức Main() như Hình I-2.



Hình I-2: Mã chương trình HelloWorld.

I.2. Biên dịch và chạy chương trình "Hello World"

- Biên dịch chương trình: **Ctrl+Shift+B Build→Build** hay **Build→Build.**
- Chạy chương trình và dò lỗi: **F5** hay **Debug→Start Debugging**.
- Biên dịch và chạy chương trình (không dò lỗi): Ctrl + F5 hay **Debug→Start Without Debugging**.

II. Cơ sở của ngôn ngữ C#

Phần này sẽ trình bày các kiến thức cơ bản cho việc lập trình trong ngôn ngữ C# như:

- Các kiểu dữ liệu xây dựng sẵn như: int, bool, string…
- Hằng, cấu trúc liệt kê
- Chuỗi, mảng
- Biểu thức và cậu lệnh
- Các cấu trúc điều khiển như if, switch, while, do...while, for, và foreach...

Nắm vững phần này sẽ giúp ta hiểu phương pháp lập trình hướng đối tượng một cách nhanh chóng, dễ dàng.

II.1. Các kiểu dữ liệu

Có nhiều cách phân loại kiểu dữ liệu trong C#. Ví dụ, kiểu dữ liệu có thể được phân làm 2 loại sau:

- Kiểu dựng sẵn: int, long …
- Kiểu người dùng tạo ra: class, struct…

Tuy nhiên, người lập trình thường quan tâm tới cách phân loại sau vì nó ảnh hưởng tới cách chúng ta gán giá trị cho biến, truyền tham số cho hàm:

- **Kiểu giá trị như: int, long, char, bool, struct.** Giá trị thật sự của chúng được cấp phát trên stack. Hai biến khác nhau sẽ chứa giá trị nằm ở hai vùng nhớ khác nhau.
- **Kiểu tham chiếu như: lớp, mảng.** Địa chỉ của kiểu tham chiếu được lưu trữ trên stack nhưng dữ liệu thật sự lưu trữ trong heap. Hai biến khác nhau có thể trỏ tới cùng một vùng nhớ.

Chú ý:

Tất cả các kiểu dữ liệu xây dựng sẵn là kiểu dữ liệu giá trị ngoại trừ các đối tượng, mảng và chuỗi. Và tất cả các kiểu do người dùng định nghĩa ngoại trừ kiểu cấu trúc đều là kiểu dữ liệu tham chiếu.

II.1.1. Các kiểu xây dựng sẵn trong C#:

Ngôn ngữ C# đưa ra các kiểu dữ liệu xây dựng sẵn rất hữu dụng, phù hợp với một ngôn ngữ lập trình hiện đại.
Mỗi kiểu nguyên thủy của C# được ánh xạ đến một kiểu dữ liệu được hỗ trợ bởi hệ thống xác nhận ngôn ngữ chung (Common Language Specification: CLS) trong MS.NET. Mỗi kiểu tương ứng trong CLS này là một class. Việc ánh xạ này đảm bảo các đối tượng được tạo ra trong C# có thể được sử dụng lại một cách tương thích với các đối tượng được tạo bởi bất cứ ngôn ngữ khác được biên dịch bởi .NET, chẳng hạn VB.NET.

Kiểu trong C#	Kích thước (byte)	Kiểu tương ứng trong .Net (CLS)	Mô tả
byte	1	Byte	Không dấu 0 → 255
char	1	Char	Ký tự unicode
bool	1	Boolean	True hay false
sbyte	1	Sbyte	Có dấu(-128 →127)

short	2	Int16	Có dấu -32.768 → 32.767
ushort	2	Uint16	Không dấu (0 → 65353)
int	4	Int32	Có dấu -2,147,483,647 → 2,147,483,647.
uint	4	Uint32	Không dấu 0 → 4,294,967,295.
float	4	Single	$+/-1.5 * 10^{-45}$ → $+/-3.4 *$
double	8	Double	$+/-5.0 * 10^{-324}$ → $+/-1.7 *$
decimal	8	Decimal	Lên đến 28 chữ số.
long	8	Int64	-9,223,372,036,854,775,808 → 9,223,372,036,854,775,807
ulong	8	Uint64	0 to 0xffffffffffffffff.

C# đòi hỏi các biến phải được khởi gán giá trị trước khi truy xuất giá trị của nó.

II.1.2. Hằng

Cú pháp:

const kiểu tên_biến = giá trị.

Ví dụ II.1.2: Khai báo hai hằng số **DoDongDac, DoSoi** (nhiệt độ đông đặc và nhiệt độ sôi).

```
using System;
class Values{
      const int DoDongDac = 32; // Nhiệt độ đông đặc
      const int DoSoi = 212;    //Độ sôi
      static void Main( )
      {
            Console.WriteLine("Nhiệt độ đông đặc của nước:
            {0}", DoDongDac);
            Console.WriteLine("Nhiệt độ sôi của nước:
            {0}", DoSoi);
      }
}
```

II.1.3. Kiểu liệt kê

Kiểu liệt kê là một kiểu giá trị rời rạc, bao gồm một tập các hằng có liên quan với nhau. Mỗi biến thuộc kiểu liệt kê chỉ có thể nhận giá trị là một trong các hằng đã liệt kê. Chẳng hạn, thay vì khai báo dài dòng và rời rạc như sau:

```
      const int DoDongDac = 32; // Nhiệt độ đông đặc
```

```
const int DoSoi = 212;      //Độ sôi
const int HoiLanh = 60;
const int AmAp = 72;
```

ta có thể định nghĩa kiểu liệt kê có tên là `NhietDo` như sau:

```
enum NhietDo{
      DoDongDac = 32; // Nhiệt độ đông đặc
      DoSoi = 212;    //Độ sôi
      HoiLanh = 60;
      AmAp = 72;
}
```

Mỗi kiểu liệt kê có thể dựa trên một kiểu cơ sở như int, short, long..(trừ kiểu char). Mặc định là kiểu int.

Ví dụ II.1.3.1: Xây dựng kiểu liệt kê mô tả kích cỡ của một đối tượng nào đó dựa trên kiểu số nguyên không dấu:

```
enum KichCo: uint {
      Nho = 1;
      Vua = 2;
      Rong = 3;
}
```

Ví dụ II.1.3.2: Ví dụ minh họa dùng kiểu liệt kê để đơn giản mã chương trình:

```
using System;
enum NhietDo{
      GiaBuot = 0, DongDac = 32, AmAp = 72, NuocSoi = 212
}

class Enum{

      static void Main(string[] args)     {
            Console.WriteLine("Nhiệt độ đông đặc của nước:
            {0}", NhietDo.DoDongDac);
            Console.WriteLine("Nhiệt độ sôi của nước: {0}",
            NhietDo.DoSoi);
      }
}
```

Mỗi hằng trong kiểu liệt kê tương ứng với một giá trị. Nếu chúng ta không chủ động gán giá trị cho các hằng số trong kiểu liệt kê thì giá trị mặc định của hằng số là 0 và tăng dần theo thứ tự cho các phần tử tiếp theo.

Ví dụ II.1.3.3

```
enum SomeValues{
            First,    Second, Third = 20, Fourth
}
```

Khi đó: `First = 0, Second = 2, Third = 20, Fourth = 21.`

II.1.4. Kiểu chuỗi

Lớp *string* (hoặc String) là lớp mô tả chuỗi các ký tự. Chuỗi là một mảng các ký tự nên ta có thể truy cập đến từng ký tự tương tự như cách truy cập đến một phần tử của mảng.
Ta khai báo một biến *string* sau đó gán giá trị cho biến string (vừa khai báo vừa khởi gán giá trị) như sau:

```
string myString = "Hello World";
```

Chú ý:

Kiểu string là kiểu bất biến (immutable), ta có thể gán (toàn bộ) một giá trị mới cho một biến kiểu string nhưng không thể thay đổi một vài ký tự trong chuỗi.

Ví dụ II.1.4.1: Có thể thực hiện các lệnh sau:

```
string S1 = "Hello World";
S1 = "how are you?";
```

Ví dụ II.1.4.2: Không thể thực hiện các lệnh sau:

```
string S1 = "Hello World";
S1[0] =' h';
```

II.2. Lệnh rẽ nhánh

II.2.1. Lệnh if

Ví dụ II.2.1: Nhập một số nguyên, kiểm tra số vừa nhập là chẵn hay lẻ.

```
using System;
namespace IfExample
{
    class IF
    {
        static void Main(string[] args)
        {
            int Value;
            Console.WriteLine("Nhap mot so nguyen!");

        //Nhập một số nguyên từ bàn phím và gán cho
    biến Value
            Value = Int32.Parse(Console.ReadLine());
            if (Value % 2 == 0)
```

```
                    Console.WriteLine("Ban nhap so
chan!");
               else
                    Console.WriteLine("Ban nhap so le!");
               Console.Read();
          }
     }
}
```

II.2.2. Lệnh switch

Cú pháp:

```
     switch (Biểu thức)
     {
          case hằngsố_1:
               Các câu lệnh
               Lệnh nhảy thoát khỏi case này
          case hằngsố_2:
               Các câu lệnh
               Lệnh nhảy thoát khỏi case này

          …
          [default: các câu lệnh]
     }
```

Ví dụ II.2.2 Hiện một thực đơn và yêu cầu người dùng chọn một

```
using System;
enum ThucDon:int {
     Xoai,Oi,Coc
}

//Minh họa lệnh switch
class Switch{
     static void Main(string[] args)     {
          ThucDon Chon;
          NhapLai:
          Console.WriteLine("Chon mot mon!");
          Console.WriteLine("{0} -
Xoai",(int)ThucDon.Xoai);
          Console.WriteLine("{0} - Oi", (int)ThucDon.Oi);
          Console.WriteLine("{0} - Coc",
(int)ThucDon.Coc);
          Chon=(ThucDon) int.Parse(Console.ReadLine());
          if (Chon < 0) goto NhapLai;

          switch(Chon)    {
               case ThucDon.Xoai:
                    Console.WriteLine("Chon 1 qua
                    xoai!");
                    break;
```

```
                case ThucDon.Oi:
                    Console.WriteLine("Chon an 1 qua
oi!");
                    break;
                case ThucDon.Coc:
                    Console.WriteLine("Chon an 1 con
coc!");
                    break;
                default:
                    Console.WriteLine("Vui long chon
lai!");
                    break;
            }
            Console.WriteLine("Chuc ban ngon mieng!");
            Console.ReadLine();
        }
}
```

Ghi chú:

Trong C#, nếu case trùng với biểu thức của lệnh switch rỗng thì cáeg chương trình tự động nhảy xuống lệnh case tiếp theo cho đến khi gặp case không rỗng. Case không rỗng cần có 1 lệnh nhảy như break, goto, return... để thoát khỏi lệnh switch.

II.2.3. Lệnh goto

Ví dụ II.2.3: Xuất các số từ 0 đến 9

```
using System;
public class Tester
{
      public static int Main( )
      {
            int i = 0;
      LapLai: // nhãn
            Console.WriteLine("i: {0}",i);
            i++;
            if (i < 10)goto LapLai; // nhảy tới nhãn LapLai
            return 0;
      }
}
```

II.2.4. Lệnh lặp while

Ví dụ II.2.4: Phân tích số nguyên dương N ra thừa số nguyên tố.

```
using System;
class While{
     static void Main(string[] args)     {
          int N, M, i;
          Console.WriteLine("Nhap so nguyen duong (>1):
");
          N= int.Parse(Console.ReadLine());
          if (N <2) {
               Console.WriteLine("So khong hop le ");
               return;
          }
          string KetQua = "";
          i = 2;
          M = N;
          while(M > 1)    {
               if (M%i == 0)   {
                    M = M/i;
                    if (KetQua.Equals(""))KetQua += i;
                    else KetQua = KetQua + "*" + i;
               }
               else i++;
          }
          Console.WriteLine("So {0} o dang thua so nguyen
          to la:{1}", N, KetQua);
          Console.ReadLine();
     }
}
```

II.2.5. Lệnh do…while

Cú pháp:

***do <lênh> while <biểu_thức>*.**

Vòng lặp **do …while** thực hiện ít nhất 1 lần.

Ví dụ II.2.5: Kết quả của đoạn lệnh sau là gì?

```
using System;
public class Tester{
     public static int Main( )      {
          int i = 11;
          do    {
               Console.WriteLine("i: {0}",i);
               i++;
          } while (i < 10);
          return 0;
     }
}
```

II.2.6. Lệnh for

Cú pháp:

for (khởi tạo; điều kiện dừng; lặp) lệnh;

Ví dụ II.2.6: Kiểm tra số nguyên tố

```
using System;
class NguyenTo{
      static void Main(string[] args)  {
            int N, i;
            Console.WriteLine("Nhap so nguyen duong (>1): ");
            N= int.Parse(Console.ReadLine());
            if (N <2)  {
                  Console.WriteLine("So khong hop le ");
                  return;
            }
            bool KetQua;
            KetQua = true;
            for ( i = 2; i<= Math.Sqrt(N); i++){
                  if (N%i==0) {
                        KetQua = false;
                        break;
                  }
            }
            if (KetQua)
                  Console.WriteLine("{0} la so nguyen to",N);
            else
                  Console.WriteLine("{0} khong la so nguyen
            to",N);

            Console.ReadLine();
      }
}
```

II.2.7. Lệnh foreach

Vòng lặp **foreach** cho phép tạo vòng lặp duyệt qua một tập hợp hay một mảng. Câu lệnh **foreach** có cú pháp chung như sau:

foreach (<kiểu thành phần> <tên truy cập thành phần > in < tên tập hợp>)
<Các câu lệnh thực hiện>

Ví dụ II.2.7: Xuất các kí tự trong chuỗi.

```
using System;
using System.Collections.Generic;
using System.Text;

public class UsingForeach {
    public static int Main()     {
        string S = "He no";
        string[] MonAn;
```

```
            MonAn = new string[3] {"Ga", "Vit","Ngan"};

            foreach (char item in S)       {
                Console.Write("{0} ", item);
            }
            Console.WriteLine();
            foreach (string Si in MonAn)
                Console.Write("{0} \n", Si);

            System.Console.Read();
            return 0;
        }
}
```

II.2.8. Lệnh continue và break

Lệnh ***continue*** được dùng trong vòng lặp (for, while, do… while), cho phép bỏ qua các lệnh phía sau nó và quay lại đầu vòng lặp **.**

Lệnh *break* cho phép thoát ra khỏi vòng lặp gần nó nhất*;*

Ví dụ II.2.8: Chương trình cho phép liên tục nhập các số nguyên dương để kiểm tra chúng có phải là số nguyên tố hay không cho đến khi nhập số 0 hoặc số 1. Nếu người dùng nhập số âm thì cho phép nhập lại.

```
public class Break_Continue    {
      public static bool laSoNguyenTo(int M)   {
            if (M < 2) return false;
            for (int i = 2; i <= Math.Sqrt(M); i++)
                if (M % i == 0) return false;
            return true;
        }

       public static void Main() {
            while (true)  {
                Console.WriteLine("Nhap 1 so nguyên duong de
kiem tra tinh chat nguyen to");
                Console.WriteLine("Nhap so 0 hoac 1 de dung
CT!");
                Console.WriteLine("Nhap lai neu nhap so am");

                int X = Int16.Parse(Console.ReadLine());
                if (X < 0)     {
                    Console.WriteLine("Nhap lai so nguyen
duong!\n");
                    continue;
                }
                else if (X == 0 || X == 1)       {
                    Console.WriteLine("Ket thuc CT\n");
                    break;
                }
```

```
                //Thu hien viec huy bo du lieu ghi khong thanh
cong
                bool kq = laSoNguyenTo(X);
                if (kq) Console.WriteLine("{0}  la so nguyen
to!\n", X);
                else Console.WriteLine("{0}  ko la so nguyen
to!\n", X);
            }
        }
    }
```

II.3. Mảng

Mảng thuộc loại dữ liệu tham chiếu (dữ liệu thực sự được cấp phát trong Heap).

II.3.1. Mảng một chiều

- Cú pháp khai báo mảng 1 chiều:

Kiểu [] Ten_bien; //Lúc này Ten_bien == null

- Cú pháp cấp phát cho mảng bằng từ khóa new:

Ten_bien = new Kiểu [Kích Thước];

Ví dụ II.3.1: Nhập một mảng số nguyên, sắp xếp và xuất ra màn hình.

```
using System;
class Array
{
     public static void NhapMang(int[] a, uint n) {
          for ( int i = 0; I < n; i++){
               Console.WriteLine("Nhap phan tu thu
{0}",i);
               a[i] = Int32.Parse(Console.ReadLine());
          }
     }
     public static void XuatMang(int[] a, uint n) {
          for ( int i = 0; i < n; i++)
               Console.Write("{0} ",a[i]);
     }
     public static void SapXep(int[] a, uint n){
          int i, j, temp;
          for ( i = 0; i < n-1; i++)          {
               for ( j = i+1; j < n; j++)     {
                    if (a[i]>a[j])       {
                         temp=a[i];
                         a[i]=a[j];
```

```
                        a[j]=temp;
                    }
                }
            }
        }

        public static void Main()      {
            Console.WriteLine("Nhap kich thuoc mang: ");
            uint n = uint.Parse(Console.ReadLine());
            int[] A;
            A = new int[n];
            NhapMang(A,n);

            Console.WriteLine("Mang vua nhap");
            XuatMang(A,n);

            SapXep(A,n);
            Console.WriteLine("Mang sau khi sap xep");
            XuatMang(A,n);
            Console.ReadLine();
        }
}
```

II.3.2. Mảng nhiều chiều

- Cú pháp khai báo mảng 2 chiều:

  Kiểu [][] Ten_bien;

Vì mảng 2 chiều là mảng mà mỗi phần tử lại là một mảng con 1 chiều nên để cấp phát đủ vùng nhớ chứa các phần tử (đủ số dòng, số cột) ta cần lần lượt thực hiện các bước sau:

1. Cấp phát mảng trỏ đến các dòng
2. Lần lượt cấp phát cho các mảng con cho từng dòng.

Cú pháp của hai bước trên như sau:

- Cấp pháp một mảng các mảng (Cấp phát mảng trỏ đến các dòng):

  Ten_bien = new Kiểu [Số dòng][];

- Cấp phát cho từng mảng con thứ i (cấp phát cho các mảng con cho từng dòng):

  Ten_bien [i] = new Kiểu [kích thước của dòng thứ i];

Ví dụ II.3.2: Nhập, xuất ma trận số thực.

```
using System;
class Matrix {
     public static void NhapMaTran(float[][]a,int n,uint
m)    {
          for ( int i = 0; i < n; i++)  {
               for (int  j = 0; j < m; j++)        {
                    Console.Write("Nhap phan tu thu
[{0},{1}]",i,j);
                    a[i][j] =
float.Parse(Console.ReadLine());           }
          }
     }

     public static void XuatMaTran(float[][]a, int n, int
m)    {
          for (int  i = 0; i < n; i++)  {
               for ( j = 0; j < m; j++)
                    Console.Write("\t{0}",   a[i][j]);
               Console.WriteLine();
          }
     }

     public static void Main()      {
          uint n, m;
          Console.WriteLine("Nhap so dong cua ma tran:
");
          n = uint.Parse(Console.ReadLine());
          Console.WriteLine("Nhap so cot cua ma tran: ");
          m = uint.Parse(Console.ReadLine());

          float[][] A;   //Khai báo ma trận
          //Cấp phát số dòng (n)
          A = new float[n][];

          //Cấp phát số cột (m)
          for(int i = 0; i < n; i++) A[i] = new float[m];

          NhapMaTran(A,n, m);

          Console.WriteLine("Ma tran vua nhap");
          XuatMaTran(A,n,m);
          Console.ReadLine();
     }
}
```

II.3.3. Một số ví dụ khác về mảng nhiều chiều

Sau đây là một số ví dụ về mảng nhiều chiều:

- Khai báo mảng 3 chiều kiểu số nguyên với kích thước mỗi chiều là 4, 2 và 3:

```
int[,,] myArray = new int [4,2,3];
```

- Khai báo mảng 2 chiều, cấp phát và khởi gán giá trị cho mảng thành ma trận 4 dòng 2 cột:

```
int[,] myArray = new int[,] { {1,2}, {3,4}, {5,6},
                  {7,8} };
```

hoặc

```
int[,] myArray = {{1,2}, {3,4}, {5,6}, {7,8}};
```

- Gán giá trị cho một phần tử trong mảng 2 chiều:

```
myArray[2,1] = 25;
```

II.4. Không gian tên (namespace)

Có thể hiểu một không gian tên như là một thư viện. Sử dụng không gian tên giúp ta tổ chức mã chương trình tốt hơn, tránh trường hợp hai lớp trùng tên khi sử dụng các thư viện khác nhau. Ngoài ra, không gian tên được xem như là tập hợp các lớp đối tượng, và cung cấp duy nhất các định danh cho các kiểu dữ liệu và được đặt trong một cấu trúc phân cấp. Việc sử dụng không gian tên trong lập trình là một thói quen tốt, bởi vì công việc này chính là cách lưu các mã nguồn để sử dụng về sau. Ngoài thư viện (namespace) do MS.NET và các hãng thứ ba cung cấp, ta có thể tạo riêng cho mình các không gian tên.

C# đưa ra từ khóa **using** để khai báo sử dụng không gian tên trong chương trình:

using < Tên namespace >

Trong một không gian tên ta có thể định nghĩa nhiều lớp và không gian tên .

Để tạo một không gian tên ta dùng cú pháp sau:

```
namespace <Tên namespace>
{
     < Định nghĩa lớp A>
     < Định nghĩa lớp B >
     .....
}
```

Ví dụ II.4.1 : Định nghĩa lớp Tester trong namespace `Programming_C_Sharp`.

```
namespace Programming_C_Sharp
```

```
{
      using System;
      public class Tester
      {
            public static int Main( )
            {
                  for (int i=0;i<10;i++)
                  {
                        Console.WriteLine("i: {0}",i);
                  }
                  return 0;
            }
      }
}
```

Ví dụ II.4.2: Khai báo không gian tên lồng nhau:

```
namespace Programming_C_Sharp {
      namespace Programming_C_Sharp_Test1     {
            using System;
            public class Tester      {
                  public static void Main( )    {
                        for (int i = 0;i < 10;i++)
                              Console.WriteLine("i: {0}",i);
                  }
            }
      }
      namespace Programming_C_Sharp_Test2 {

            public class Tester {
                  public static void function2( )          {
                        for (int i=0;i<10;i++)        {
                              Console.WriteLine("i: {0}",i);
                  }
            }
      }
}
```

Phụ lục B - BIỆT LỆ (NGOẠI LỆ)

- Là các dạng lỗi gặp phải khi chạy chương trình (lúc biên dịch chương trình không phát hiện được). Thường là do người dùng gây ra lúc chạy chương trình.
- Kết thúc bởi từ khoá Exception.

I. Ném ra biệt lệ

Để báo động sự bất thường của chương trình.

Cú pháp:

throw [biểu thức tạo biệt lệ];

Sau khi ném ra một ngoại lệ, các đoạn lệnh sau lệnh **throw** sẽ bị bỏ qua. Chương trình thực hiện việc bắt ngoại lệ hoặc dừng.

II. Bắt ngoại lệ

- Việc bắt và xử lý biệt lệ nhằm giúp chương trình có tính dung thứ lỗi cao hơn, cho phép vẫn chạy chương trình đối với những lỗi không quá quan trọng. Chẳng hạn khi nhập một giá trị nguyên, người dùng vô tình nhập một ký tự, khi đó không nhất thiết phải dừng chương trình mà chỉ thông báo lỗi và cho phép người dùng nhập lại.
- Để chương trình thân thiện hơn đối với người sử dụng. Thông báo lỗi cụ thể, thay vì dạng thông báo lỗi mang tính kỹ thuật khó hiểu của hệ thống.

Việc bắt ngoại lệ được thực hiện thông qua khối ***try { } catch { }*** như sau:

```
try {
     Các câu lệnh có thể gây ra biệt lệ.
}
catch (khai báo biệt lệ 1 ) {
     Các câu lệnh xử lý     biệt lệ 1
}
…
catch (khai báo biệt lệ n ) {
     Các câu lệnh xử lý biệt lệ n
}
```

- Nếu không có khai báo biệt lệ nào trong khối ***catch*** thì khi đó ta bắt tất cả các dạng ngoại lệ do khối ***try*** gây ra.

Ví dụ: Xét đoạn chương trình

```
using System;
```

```
class Class1
{
      static void Main(string[] args)
      {
            int x=0;
            Console.WriteLine("Nhap mot so nguyen");
            x = int.Parse(Console.ReadLine());
            Console.WriteLine("So nguyen vua nhap {0}",x);
            Console.ReadLine();
      }
}
```

- Khi chạy chương trình trên, nếu ta (vô tình) nhập một dữ liệu không phải là số nguyên (chẳng hạn nhập ký tự 'r'), chương trình sẽ dừng và báo lỗi **runtime** sau:

  An unhandled exception of type 'System.FormatException' occurred in mscorlib.dll

  Additional information: Input string was not in a correct format.

Vì vậy, để chương trình có tính dung thứ lỗi (vì đây có thể là lỗi vô tình của người sử dụng) ta cần viết lại như sau để cho người dùng nhập lại:

```
using System;
class Class1{
      static void Main(string[] args)  {
            int x=0;
            Console.WriteLine("Nhap mot so nguyen");
            NHAPLAI:
            try   {
                  x = int.Parse(Console.ReadLine());
            }
            //catch(System.Exception e)
            catch(System.FormatException e)  {
                  Console.WriteLine("Loi : " + e.ToString());
                 Console.WriteLine("Hay nhap lai 1 so nguyen!");
                  goto NHAPLAI;
            }
            Console.WriteLine("So nguyen vua nhap {0}",x);
            Console.ReadLine();
      }
}
```

Vì đoạn mã

```
x = int.Parse(Console.ReadLine());
```

có thể gây ra biệt lệ

```
System.FormatException
```

nên ta đặt nó trong khối ***try*** và khối ***catch*** bắt biệt lệ này. Sau đó xuất thông báo lỗi, nhưng không dừng chương trình mà cho phép nhập lại bằng lệnh nhảy tới nhãn ***NHAPLAI*:**

```
        goto NHAPLAI;
```

Vì mọi loại biệt lệ đều dẫn xuất từ ***System.Exception*** nên ta bất cứ biệt lệ cũng là một biệt lệ dạng ***System.Exception***. Do vậy, nếu ta không biết loại biệt lệ là gì ta thay lệnh

```
    catch(FormatException e)
```

bằng lệnh:

```
    catch(Exception e)
```

Nếu muốn bắt mọi ngoại lệ nhưng không