

BÀI GIẢNG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

CHƯƠNG 2 GIỚI THIỆU VỀ C SHARP (C#)

TRẦN THỊ THU THẢO

BỘ MÔN TIN HỌC QUẢN LÝ, KHOA THỐNG KÊ – TIN HỌC
TRƯỜNG ĐẠI HỌC KINH TẾ, ĐẠI HỌC ĐÀ NẴNG
THAOTRAN@DUE.EDU.VN

NỘI DUNG

1. Ngôn ngữ C#

2. Biến và biểu thức

3. Cấu trúc điều khiển

4. Mảng trong C#

5. Hàm

Ngôn ngữ C#

SƠ LƯỢC VỀ C#

- C# (C Sharp) là một ngôn ngữ lập trình thuần hướng đối tượng được phát triển bởi microsoft.
- C# ra đời năm 2000, được thiết kế chủ yếu bởi **Anders Hejlsberg** – kiến trúc sư phần mềm nổi tiếng với các sản phẩm Turbo Pascal, Delphi, . . .
- Được xây dựng dựa trên nền tảng của 2 ngôn ngữ lập trình mạnh nhất đó là C++ và Java. Do đó C# được miêu tả là ngôn ngữ có sự cân bằng giữa C++, Visual Basic, Delphi và Java.
- C# với sự hỗ trợ mạnh mẽ của .NET Framework giúp cho việc tạo một ứng dụng Windows Forms hay WPF (Windows Presentation Foundation), . . . trở nên rất dễ dàng.

CÀI ĐẶT C# TRONG VISUAL STUDIO CODE



1. Cài đặt .Net Core SDK

Để phát triển ứng dụng .NET trước tiên cần cài đặt **.NET Core SDK**, hay vào [Download .NET](#), tải về .NET Core SDK phù hợp với hệ điều hành của mình và tiến hành cài đặt.

2. Cài đặt Visual Studio Code

Truy cập [Visual Studio Code](#) để tải phần mềm

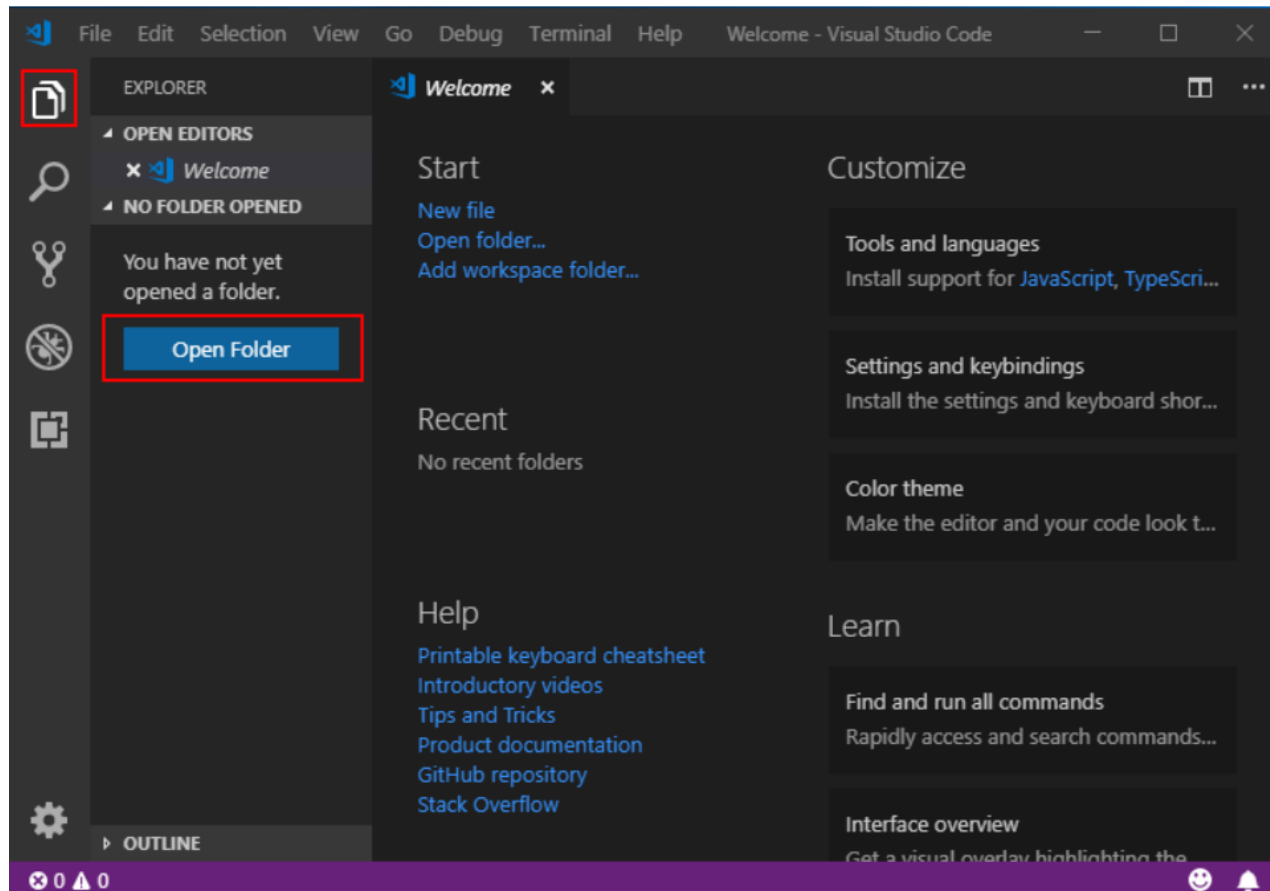
2. Tải các tiện ích (Extension trong VSCode)

- [OmniSharp](#)
- [C# for Visual Studio Code \(powered by OmniSharp\)](#)
- [Code Runner](#)
- [C# Extensions](#)
- [Paste JSON as Code](#)

Khởi tạo Project C# trong Visual Studio Code

- Mở Visual Studio Code.
- Click vào icon Explorer ở menu bên trái và click Open Folder.
- Chọn Folder muốn lưu trữ dự án C#. Trong trường hợp này, nên tạo và chọn một folder mới.

Khởi tạo Project C# trong Visual Studio Code



Khởi tạo Project C# trong Visual Studio Code

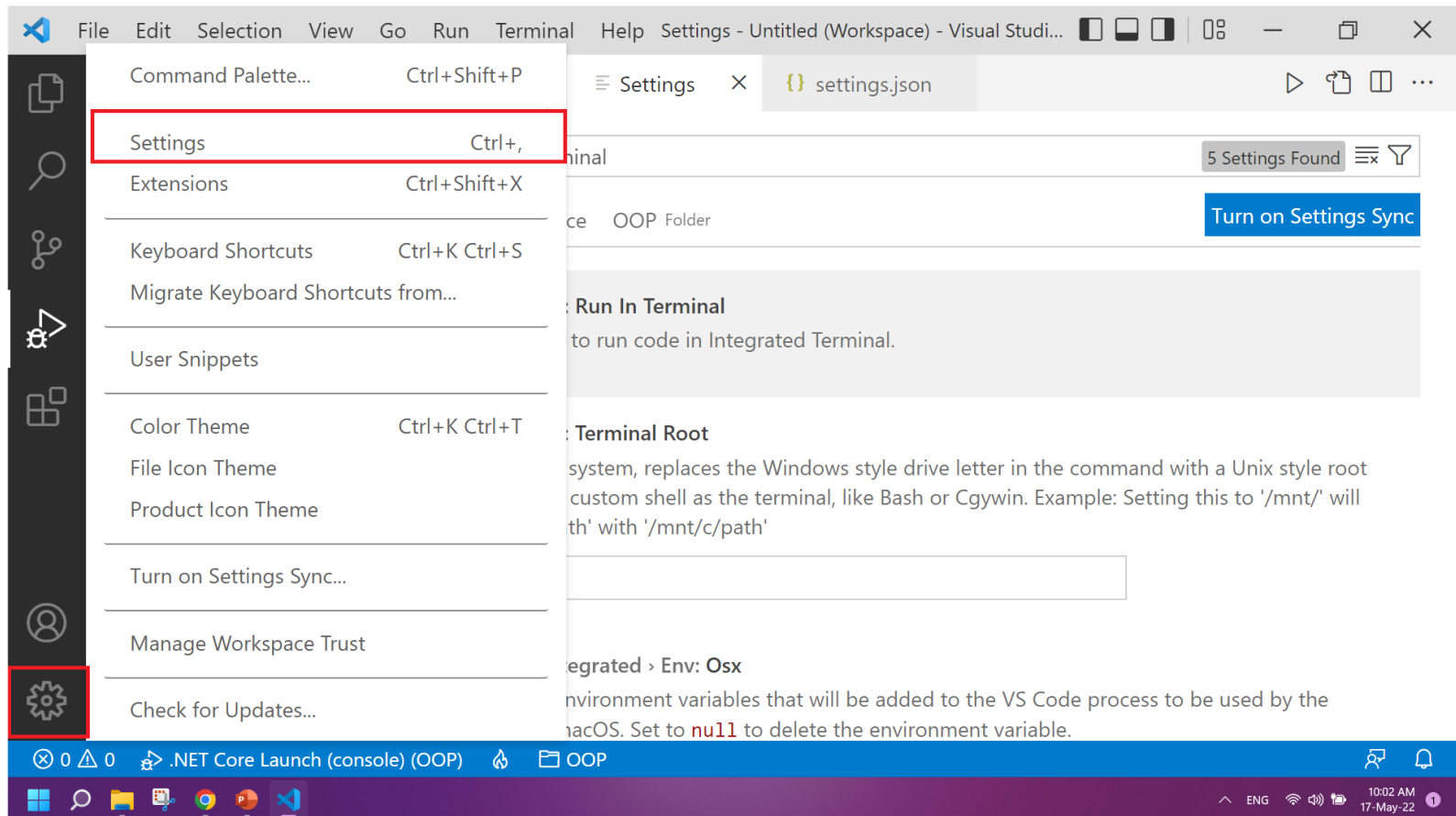
- Mở terminal tích hợp trong Visual Studio Code bằng cách chọn **View → Integrated Terminal** từ menu chính.

- Trong cửa sổ Terminal hiện ra, nhập câu lệnh

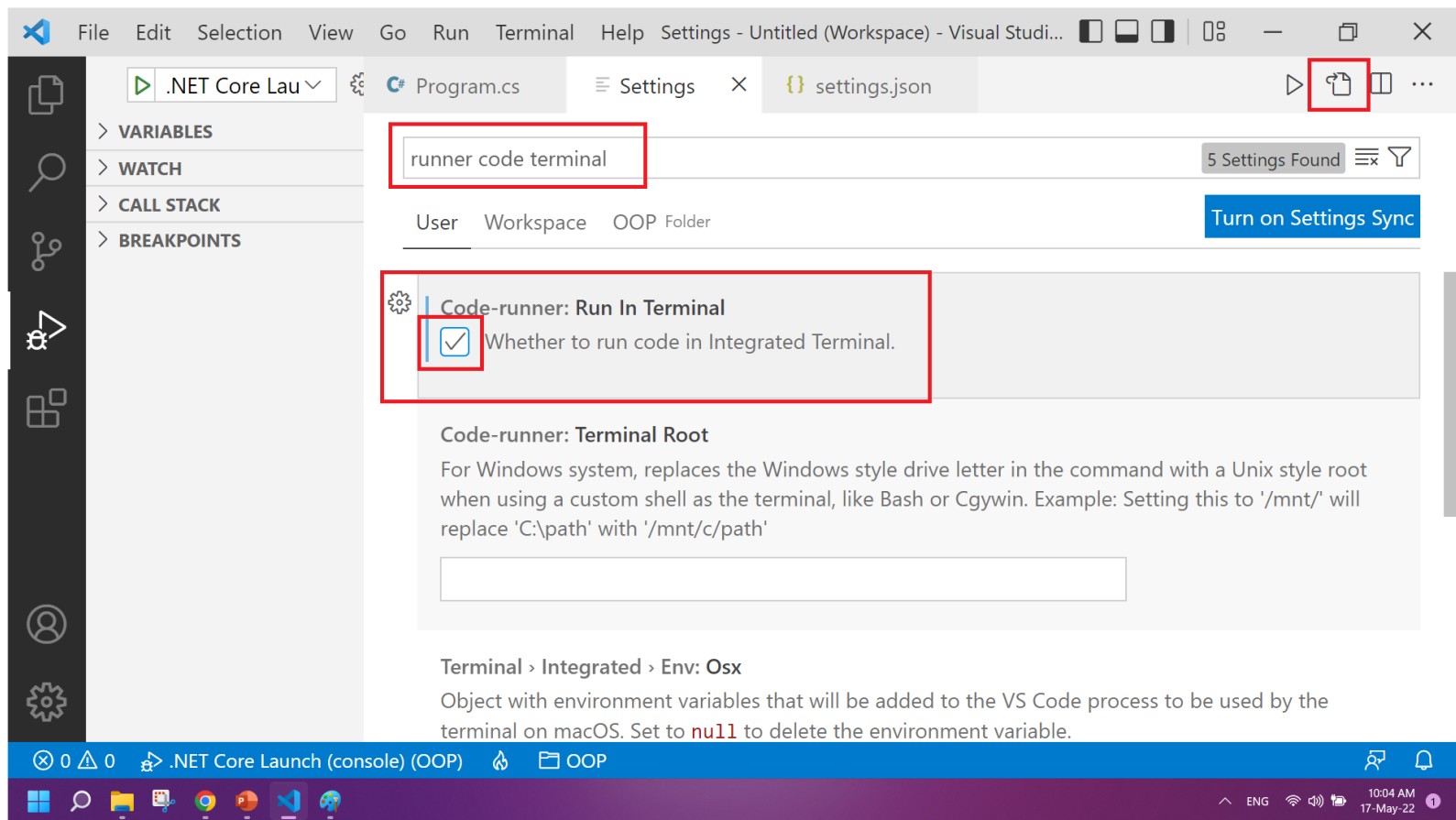
dotnet new console -o <tên dự án>

- Câu lệnh này sẽ tạo một file Program.cs vào folder theo tên thư mục đã chọn, trong file đó chứa chương trình “<tên dự án>”; đồng thời với đó là một file dự án C# tên là <tên dự án>.csproj.

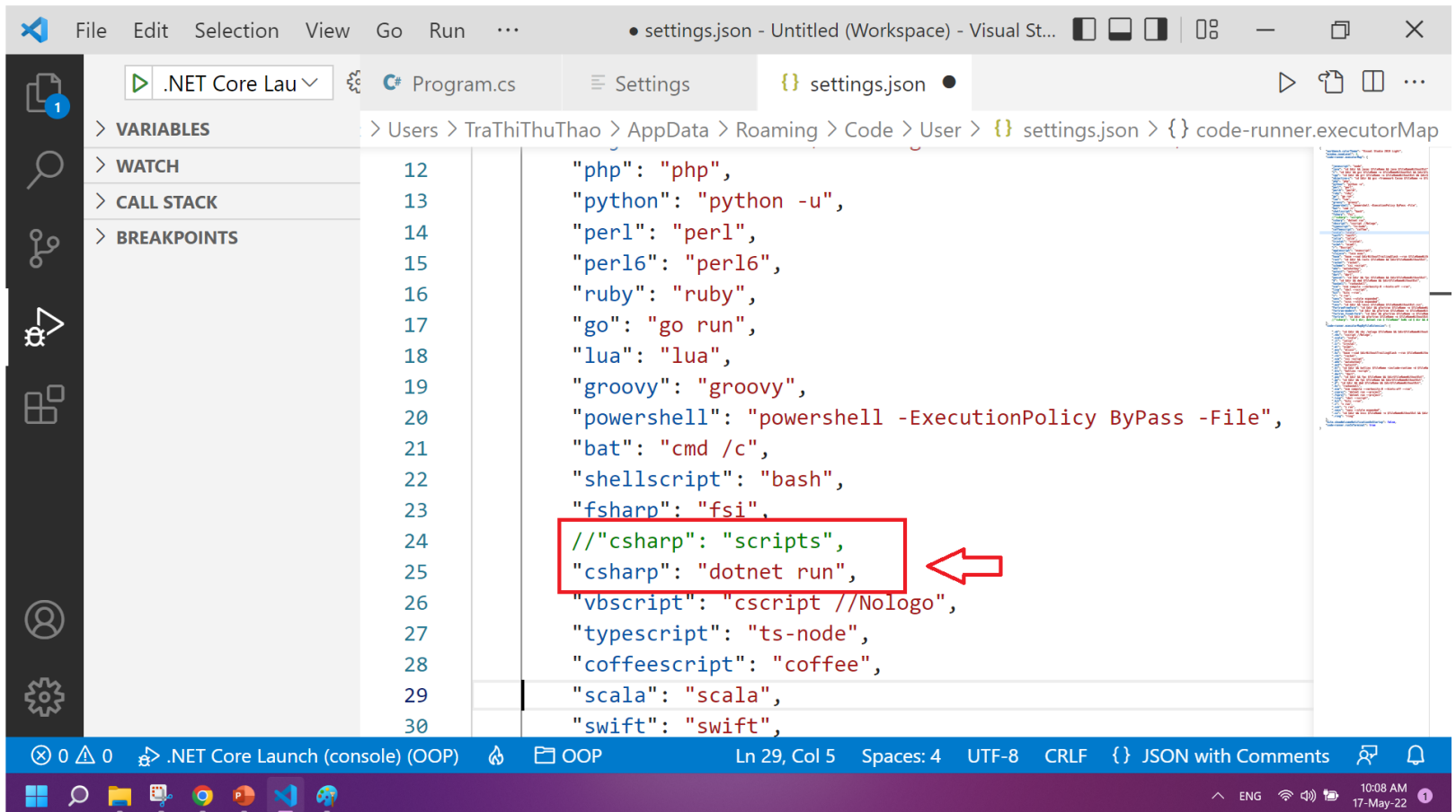
Debug and Run Project C# trong Visual Studio Code



Debug and Run Project C# trong Visual Studio Code



Debug and Run Project C# trong Visual Studio Code



The screenshot shows the Visual Studio Code interface with the `settings.json` file open. The file is located at `Users > TraThiThuThao > AppData > Roaming > Code > User > settings.json`. The `code-runner.executorMap` section is visible, and the `csharp` entry is highlighted with a red box and an arrow pointing to it.

```
{
  "php": "php",
  "python": "python -u",
  "perl": "perl",
  "perl6": "perl6",
  "ruby": "ruby",
  "go": "go run",
  "lua": "lua",
  "groovy": "groovy",
  "powershell": "powershell -ExecutionPolicy ByPass -File",
  "bat": "cmd /c",
  "shellscript": "bash",
  "fsharp": "fsi",
  // "csharp": "scripts",
  "csharp": "dotnet run",
  "vbscript": "cscript //NoLogo",
  "typescript": "ts-node",
  "coffeescript": "coffee",
  "scala": "scala",
  "swift": "swift",
}
```

Debug and Run Project C# trong Visual Studio Code

Hãy mở file `./vscode/launch.json`, tìm đến dòng:

`"console": "internalTerminal"`

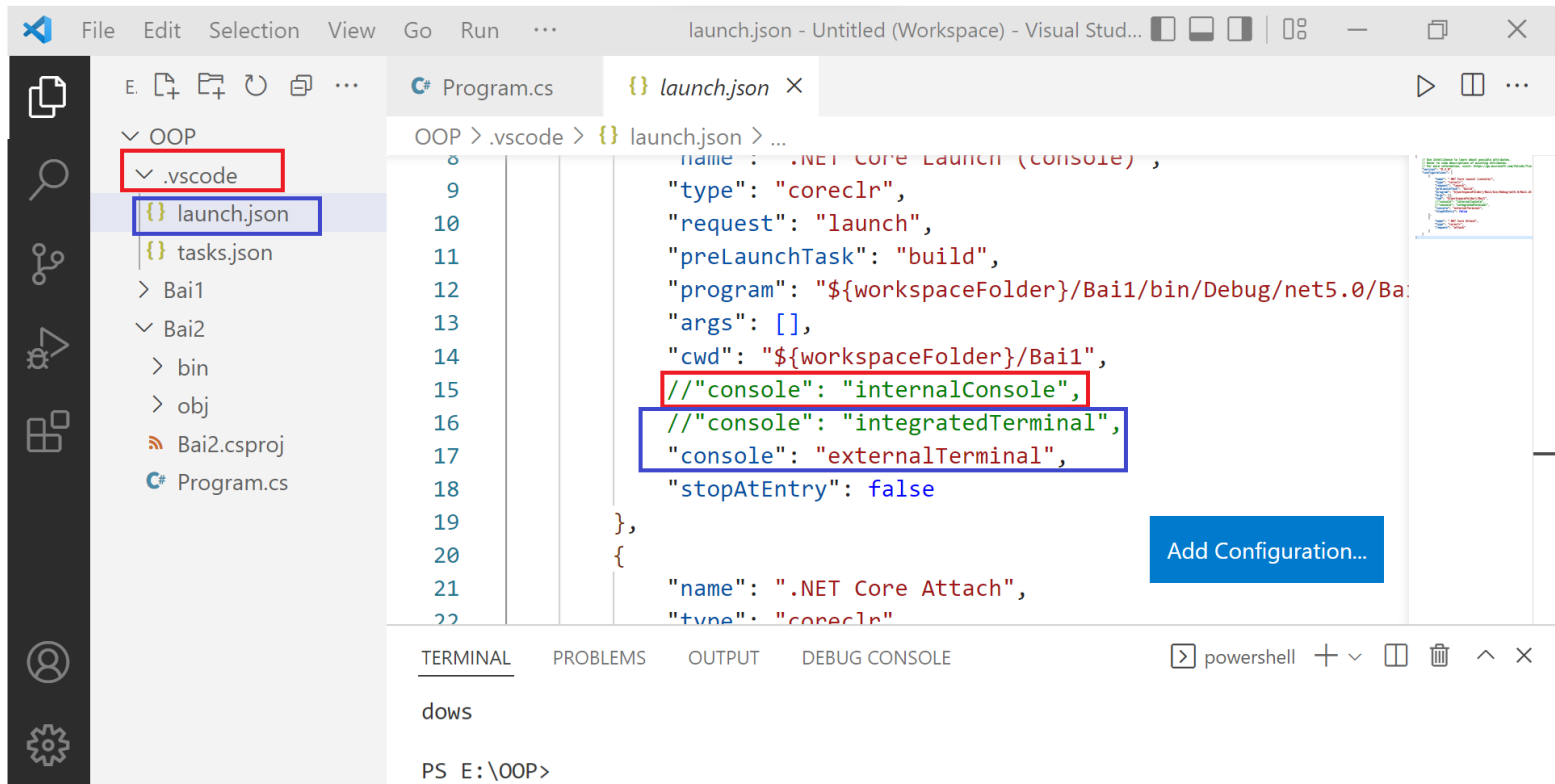
hãy thay bằng

`"console": "integratedTerminal"` (chạy trên Terminal)

hoặc

`"console": "externalTerminal"` (mở cửa sổ Terminal mới khi chạy)

Debug and Run Project C# trong Visual Studio Code



Debug and Run Project C# trong Visual Studio Code

Biên dịch, chạy chương trình

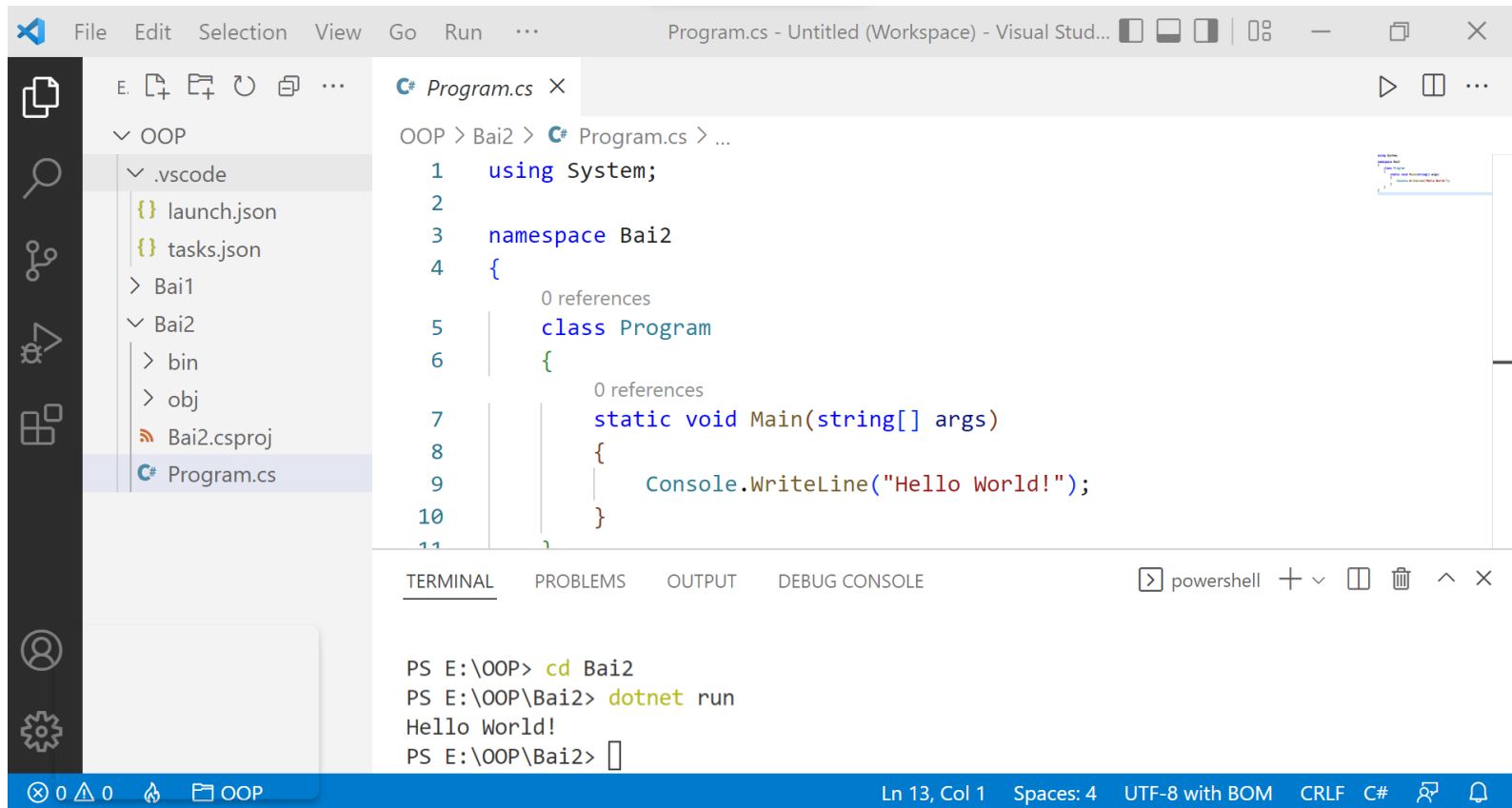
Để chạy chương trình (mở cửa sổ Terminal trong VSC:
CTRL + ~, đang ở thư mục <tên thư mục>) gõ:

dotnet run

Trường hợp không ở tại thư mục chứa project thì gõ:

cd <tên đường dẫn chứa project>

Debug and Run Project C# trong Visual Studio Code



The screenshot displays the Visual Studio Code interface. On the left, the Explorer sidebar shows a project structure with folders `.vscode`, `Bai1`, and `Bai2`. The `Bai2` folder is expanded, revealing subfolders `bin` and `obj`, and files `Bai2.csproj` and `Program.cs`. The `Program.cs` file is selected and its content is shown in the main editor. The code is a simple C# program that prints "Hello World!".

```
1 using System;
2
3 namespace Bai2
4 {
5     0 references
6     class Program
7     {
8         0 references
9         static void Main(string[] args)
10         {
11             Console.WriteLine("Hello World!");
12         }
13 }
```

At the bottom, the TERMINAL panel is active, showing the execution of the program:

```
PS E:\OOP> cd Bai2
PS E:\OOP\Bai2> dotnet run
Hello World!
PS E:\OOP\Bai2>
```

The status bar at the bottom indicates the current position is Line 13, Column 1, with 4 spaces, using UTF-8 with BOM encoding, CRLF line endings, and the C# language.

BIẾN VÀ BIỂU THỨC

- ❑ Kiểu dữ liệu (Data Types)
- ❑ Hằng (Constants)
- ❑ Biến (Variable)
- ❑ Toán tử
- ❑ Các hàm nhập xuất dữ liệu

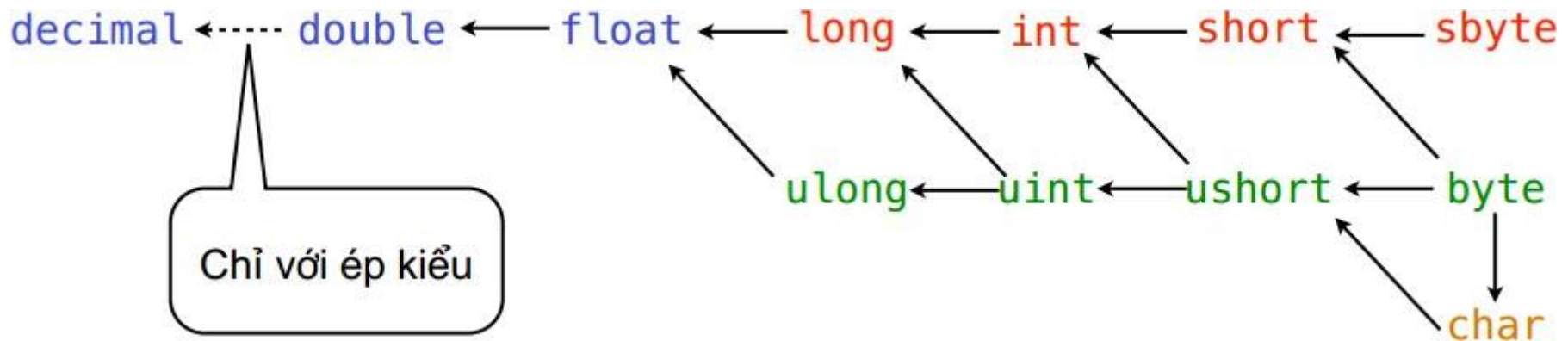
KIỂU DỮ LIỆU

- ❑ **Các kiểu dữ liệu cơ bản:** đã được xây dựng sẵn trong ngôn ngữ lập trình C#

Type	Size in Bytes	Range	Meaning
bool	1/2	True or False	Represents true/false values
byte	1	0 to 255	8-bit unsigned integer
char	2	Any Unicode Character	Character
decimal	12	$(-7.9 \times 10^{28} \text{ to } 7.9 \times 10^{28}) / 100 \text{ to } 28$	Numeric type for financial calculations
double	8	$(+/-)5.0 \times 10^{-324} \text{ to } (+/-)1.7 \times 10^{308}$	Double-precision floating point
float	4	$-3.4 \times 10^{38} \text{ to } +3.4 \times 10^{38}$	Single-precision floating point
int	4	-2,147,483,648 to 2,147,483,647	Integer
long	8	923,372,036,854,775,808 to 9,223,372,036,854,775,807	Long integer
sbyte	1	-128 to 127	8-bit signed integer
short	2	-32,768 to 32,767	Short integer
uint	4	0 to 4,294,967,295	Unsigned integer

KIỂU DỮ LIỆU

❑ Sự tương thích giữa các kiểu



- Dùng khi gán dữ liệu giữa các biến có kiểu khác nhau;
- Chuyển đổi ngầm định theo chiều mũi tên

```
int DiemThi = 5;           //→ DiemThi = 5  
float Mon1 = DiemThi;     //→ Mon1 = 5.0
```

❑ Sự tương thích giữa các kiểu

- Chuyển đổi theo chiều ngược lại có thể xảy ra mất dữ liệu hoặc có kết quả không mong muốn

```
int i = 3.1416;
```

```
//→lỗi vì vế trái có kiểu thấp hơn vế phải
```

```
// Được viết lại như sau:
```

```
int i = (int)3.1416;
```

```
// tuy nhiên sẽ bị mất dữ liệu
```

```
// → i có giá trị là: 3
```

HẲNG (CONSTANT)

□ Khái niệm

- Hằng là một đại lượng mang giá trị không thay đổi trong quá trình xử lý của chương trình

□ Một số trường hợp sử dụng Hằng

- Giá trị của số PI để tính diện tích hình tròn ($PI=3.14159265358979$)
- Tính tiền BHXH, biết rằng: $BHXH=30\%$ Tiền lương; Trong đó 30% chính là đại lượng Hằng
- Số SV tối đa để xét học bổng ở mỗi Khoa là 10 SV;

HẲNG (CONSTANT)

❑ Cú pháp định nghĩa Hằng

```
const <Tên_Kiểu> <Tên_Hằng> = <Giá_trị>;
```

Từ khóa khai báo

Giá trị của Hằng cần định nghĩa

■ Ví dụ:

```
const double PI = 3.14159265358979;  
const float TyLeBHXH = 0.3;  
const int Max = 10;
```

❑ Vị trí khai báo HẰNG

```
3 namespace ViDu1
4 {
5     class Program
6     {
7         //Vị trí khai báo HẰNG --> TOÀN CỤC
8         //...
9         static void Main(string[] args)
10        {
11            //Vị trí khai báo HẰNG --> CỤC BỘ
12            //...
13        }
14    }
15 }
```

❑ Lưu ý khi sử dụng Hằng

- Hằng cần được khai báo và khởi gán giá trị trước khi sử dụng;
- Khai báo Hằng trong phạm vi một lớp và bên ngoài một Hàm;
- Khởi gán cho Hằng là một giá trị được xác định hoặc một biểu thức, không gán cho Hằng giá trị từ một Biến;

//Gán giá trị cho Hằng

```
const int MAX = 1000;
```

```
const char newline = '\\n';
```

```
const float Pi = 3.1416;
```

//Gán biểu thức cho Hằng

```
const float SquarePi = Pi * Pi;
```

❑ KHAI BÁO BIẾN:

- Một biến được sử dụng trong chương trình C# cần phải được khai báo trước;
- **Cú pháp khai báo một biến:**

```
<Tên_Kiểu> <Tên_Biến> ;
```

Ví dụ:

```
float HeSoLuong; //Biến kiểu số thực  
int Tuoi_NV1; //Biến kiểu số nguyên  
string HoTen_NV1; //Biến kiểu chuỗi  
boolean GioiTinh; //Biến kiểu logic
```

- **Cú pháp gán giá trị cho biến:**

```
<Tên_Biến> = <Giá_Trị> ;
```

Ví dụ:

```
float HeSoLuong;  
  
HeSoLuong = 1.92;
```

❑ KHAI BÁO BIẾN:

- Vừa khai báo vừa khởi tạo giá trị cho biến:

```
<Tên_Kiểu> <Tên_Biến> = <Giá_Trị> ;
```

Ví dụ:

```
float HeSoLuong=1.92;  
int Tuoì_NV1=23;  
string MonHoc="CSLT";  
char XepLoai='A'; //Biến kiểu ký tự  
Boolean GioiTinh=True;
```

- Khai báo đồng thời nhiều biến có cùng kiểu:

```
<Tên_Kiểu> <Tên_Biến1>, <Tên_Biến2>, ...;
```

Ví dụ:

```
float HSL1, HSL2, HSL3;  
int TuoìNV1, TuoìNV2, TuoìNV3;
```


❑ CÁC LOẠI BIẾN

■ Biến cục bộ

- Là biến được khai báo trong một HÀM và chỉ ảnh hưởng trong phạm vi của HÀM đó;
- Khi ra khỏi HÀM, biến sẽ không còn giá trị sử dụng;

■ Biến toàn cục

- Biến toàn cục được khai báo ngay sau dòng lệnh khai báo lớp class;
- Biến toàn cục có giá trị sử dụng trong toàn bộ chương trình;

❑ CÁC LOẠI BIẾN

```
class Program
```

```
{
```

```
    const float PI = 3.1416F;
```

```
    static float ChuVi;
```

```
    0 references
```

```
    static void Main(string[] args)
```

```
    {
```

```
        int BanKinh=5;
```

```
        ChuVi = TinhChuVi(BanKinh);
```

```
        Console.Write(ChuVi);
```

```
        Console.Read();
```

```
    }
```

```
    1 reference
```

```
    static float TinhChuVi(int bk)
```

```
    {
```

```
        float ChuVi;
```

```
        ChuVi = bk * PI;
```

```
        return ChuVi;
```

```
    }
```

```
}
```

Phạm vi TOÀN CỤC

Phạm vi CỤC BỘ

Vị trí khai báo **BIẾN TOÀN CỤC**
và **HẲNG**

Vị trí khai báo **BIẾN CỤC BỘ**
trong hàm **Main()**

Vị trí khai báo **BIẾN CỤC BỘ**
trong chương trình con

❑ Toán tử toán học

- Gán $a=10$; $b=5$;

Toán tử	Mô tả	Biểu thức	Kết quả
+	Phép cộng	$a + b$	15
-	Phép trừ	$a - b$	5
*	Phép nhân	$a * b$	50
/	Phép chia hết	a / b	2
%	Phép chia lấy dư	$a \% b$	0

❑ Toán tử quan hệ

Toán tử	Mô tả	Biểu thức	Kết quả
==	Phép so sánh bằng	$a == b$	False
!=	Phép so sánh khác	$a != b$	True
>	Phép so sánh lớn hơn	$a > b$	True
<	Phép so sánh bé hơn	$a < b$	False
>=	Phép so sánh lớn hơn hoặc bằng	$a >= b$	True
<=	Phép so sánh bé hơn hoặc bằng	$a <= b$	False

❑ Toán tử logic

- Gán $a=10$; $b=5$

Toán tử	Mô tả	Biểu thức	Kết quả
&	Phép và (AND)	$(a==10) \& (b>10)$	False
	Phép hoặc (OR)	$(a==10) (b>10)$	True
	Phép hoặc (OR)	$(a==10) (b>10)$	True
^	Phép duy nhất một, trả về TRUE nếu tồn tại đúng một biểu thức có giá trị TRUE	$(a==10) ^ (b>10)$	False

❑ Toán tử tăng và giảm

- Gán $a=10$;

Toán tử	Mô tả	Biểu thức	Kết quả
++a	Phép tăng trước	$b = ++a$	$b = 11$; $a = 11$
a++	Phép tăng sau	$b = a++$	$b = 10$; $a = 11$
--a	Phép giảm trước	$b = --a$	$b = 9$; $a = 9$
a--	Phép giảm sau	$b = a--$	$b = 10$; $a = 9$

❑ Toán tử gán

- Gán $a=10$;

Biểu thức	Biểu thức tương đương	Kết quả
$a += 5;$	$a = a + 5;$	$a = 15$
$a -= 5;$	$a = a - 5;$	$a = 5$
$a *= 5;$	$a = a * 5;$	$a = 25$
$a /= 5;$	$a = a / 5;$	$a = 2$
$a \%= 5;$	$a = a \% 5;$	$a = 0$

CÁC HÀM NHẬP XUẤT DỮ LIỆU

- ❑ Hàm in dữ liệu lên màn hình
 - Cú pháp

```
Console.Write("<Chuỗi ký tự>");  
Console.Write("<Chuỗi ký tự> + <Biến>");
```

- In nội dung rồi xuống dòng:

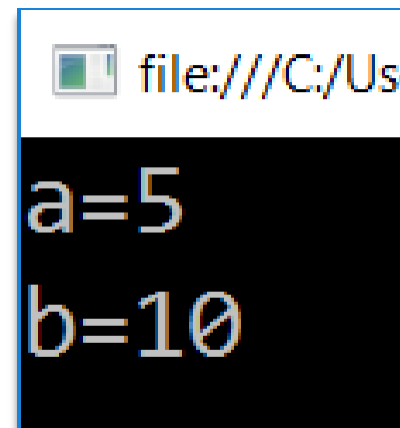
```
Console.WriteLine("<Chuỗi ký tự>");  
Console.WriteLine("<Chuỗi ký tự> + <Biến>");
```

CÁC HÀM NHẬP XUẤT DỮ LIỆU

❑ Hàm in dữ liệu lên màn hình

- Ví dụ: sử dụng mã `\n` để xuống dòng

```
static void Main(string[] args)
{
    int a = 5, b = 10;
    Console.Write("a=" + a);
    Console.Write("\nb=" + b);
    Console.ReadKey();
}
```

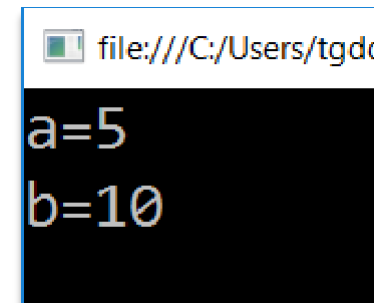


CÁC HÀM NHẬP XUẤT DỮ LIỆU

❑ Hàm in dữ liệu lên màn hình

- Ví dụ: in giá trị của **biến** lên màn hình

```
static void Main(string[] args)
{
    int a = 5, b = 10;
    Console.WriteLine("a=" + a);
    Console.Write("b=" + b);
    Console.ReadKey();
}
```

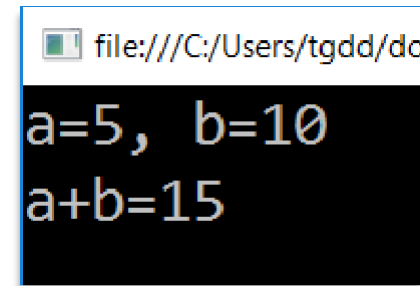


CÁC HÀM NHẬP XUẤT DỮ LIỆU

❑ Hàm in dữ liệu lên màn hình

- Ví dụ: in giá trị của **biểu thức** lên màn hình

```
static void Main(string[] args)
{
    int a = 5, b = 10;
    Console.WriteLine("a=" + a + ", b=" + b);
    Console.Write("a+b=" + (a+b));
    Console.ReadKey();
}
```

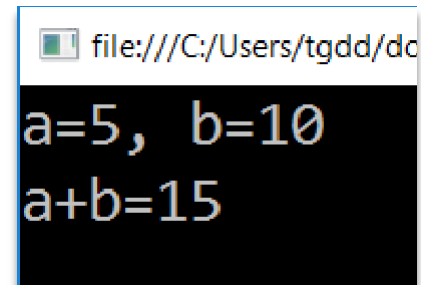


CÁC HÀM NHẬP XUẤT DỮ LIỆU

❑ Hàm in dữ liệu lên màn hình

- Ví dụ: sử dụng tham số để in dữ liệu

```
static void Main(string[] args)
{
    int a = 5, b = 10;
    Console.WriteLine("a={0}, b={1}",a,b);
    Console.Write("a+b={0}",(a+b));
    Console.ReadKey();
}
```



CÁC HÀM NHẬP XUẤT DỮ LIỆU

❑ Hàm nhập dữ liệu từ bàn phím

- Nhập một ký tự: (ít sử dụng)

```
<Biến> = Console.Read();
```

- Nhập một ký tự:

```
Console.ReadKey();
```

- Nhập một ký tự hoặc chuỗi ký tự:

```
<Biến> = Console.ReadLine();
```

- Nhập một số:

```
<Biến>=<Ép kiểu>Console.ReadLine();
```

CÁC HÀM NHẬP XUẤT DỮ LIỆU

❑ Hàm nhập dữ liệu từ bàn phím

■ Ví dụ:

```
static void Main(string[] args)
{
    int Tuoi; string HoTen;
    Console.Write("Ho va ten: ");
    HoTen = Console.ReadLine(); //Nhap mot XAU KY TU

    Console.Write("Tuoi: ");
    Tuoi = Convert.ToInt32(Console.ReadLine()); //Nhap mot SO

    Console.WriteLine("\nXin chao ban: {0}, Chuc mung sinh nhat lan thu: {1}",HoTen,Tuai);
    Console.ReadKey();
}
```

Ép sang kiểu số nguyên
(được sử dụng trong trường
hợp nhập SỐ)

file:///C:/Users/tgdd/documents/visual studio 2010/Projects/ChuongTrinhDauTien/ChuongTrinhDauTien/bin/Debug/ChuongTrinh

Ho va ten: Bao Nam

Tuai: 10

Xin chao ban: Bao Nam, Chuc mung sinh nhat lan thu: 10

CÁC HÀM NHẬP XUẤT DỮ LIỆU

□ **Ví dụ:** Một số hàm ép kiểu thông dụng

```
float x;  
int y;  
char ch;  
double z;
```

```
x = float.Parse(Console.ReadLine());  
ch = char.Parse(Console.ReadLine());
```

```
y = Convert.ToInt32(Console.ReadLine());  
//Hoặc  
y = int.Parse(Console.ReadLine());
```

```
z = Convert.ToDouble(Console.ReadLine());  
//Hoặc  
z = double.Parse(Console.ReadLine());
```

CÁC HÀM NHẬP XUẤT DỮ LIỆU

❑ Hàm nhập dữ liệu từ bàn phím

```
string HoTen; char GioiTinh; double HSL; int Tuoi;  
Console.Write("Ho va ten: ");  
HoTen= Console.ReadLine(); //Nhập một xâu KÝ TỰ --> Không ép kiểu  
  
Console.Write("Gioi tinh (M/F): ");  
GioiTinh = Convert.ToChar(Console.ReadLine()); //Nhập một KÝ TỰ --> Ép kiểu  
  
Console.Write("Tuoi: ");  
Tuoi = Convert.ToInt32(Console.ReadLine()); //Nhập một SỐ NGUYÊN --> Ép kiểu  
  
Console.Write("He so luong: ");  
HSL = Convert.ToDouble(Console.ReadLine()); //Nhập một SỐ THỰC --> Ép kiểu  
  
Console.Write("--> {0},{1},{2},{3}",HoTen,GioiTinh,Tuoi,HSL);
```



```
file:///C:/Users/tgdd/documents/visual studio 201  
Ho va ten: Bao Nam  
Gioi tinh (M/F): M  
Tuoi: 20  
He so luong: 2.34  
--> Bao Nam,M,20,2.34
```

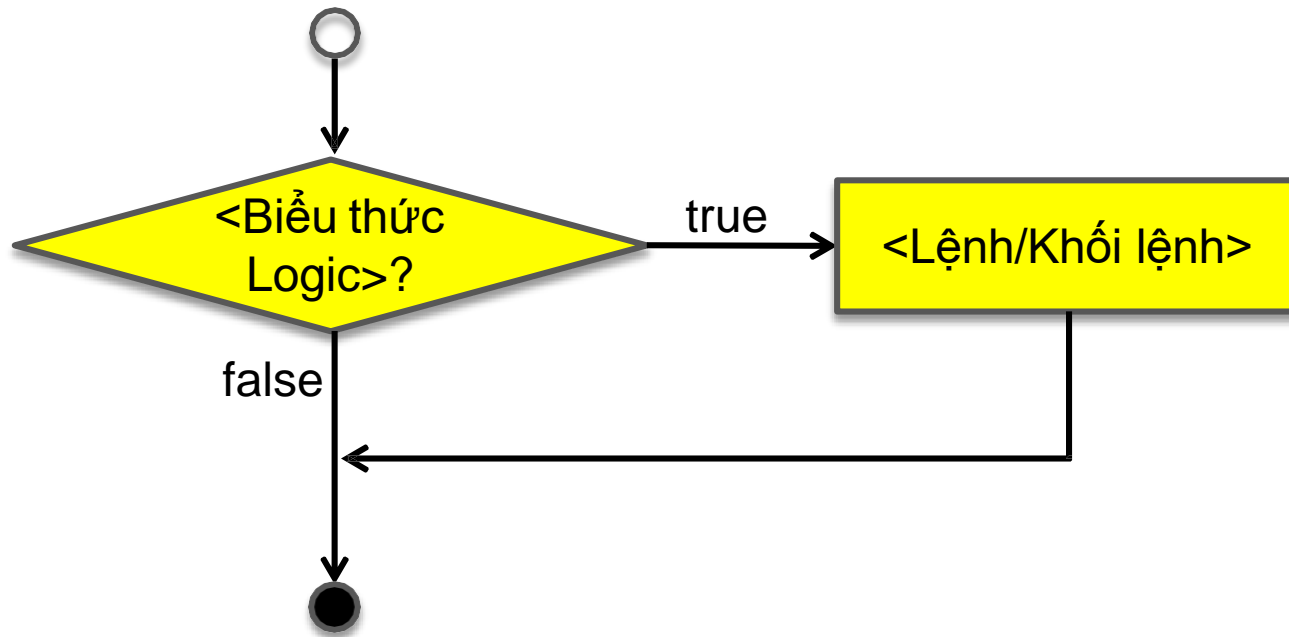
CẤU TRÚC ĐIỀU KHIỂN

- ❑ Cấu trúc điều kiện if
- ❑ Cấu trúc rẽ nhánh switch ... case
- ❑ Cấu trúc lặp while
- ❑ Cấu trúc lặp do ... while
- ❑ Cấu trúc lặp for ... loop
- ❑ Câu lệnh nhảy break, continue, return

CẤU TRÚC ĐIỀU KIỆN IF – DẠNG 1

□ Cú pháp

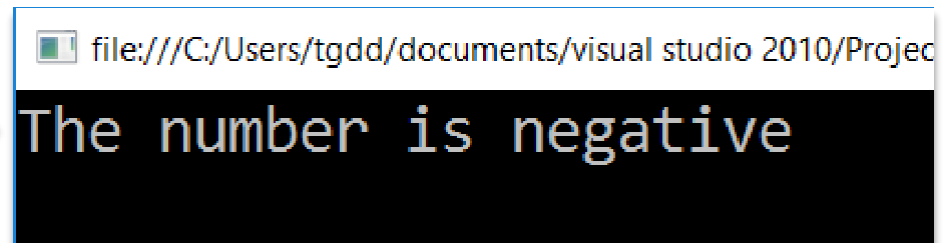
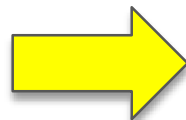
```
if (<Biểu thức Logic>
{
    //Lệnh hoặc Khối lệnh
}
```



CẤU TRÚC ĐIỀU KIỆN IF – DẠNG 1

- ❑ **Ví dụ:** viết chương trình giải bài toán theo sơ đồ khối sau

```
int num = -4;  
if (num < 0)  
{  
    Console.WriteLine("The number is negative");  
}
```



CẤU TRÚC ĐIỀU KIỆN IF – DẠNG 2

❑ Cú pháp

```
if (<Biểu thức Logic>)
```

```
{
```

```
    //Lệnh hoặc Khối lệnh 1
```

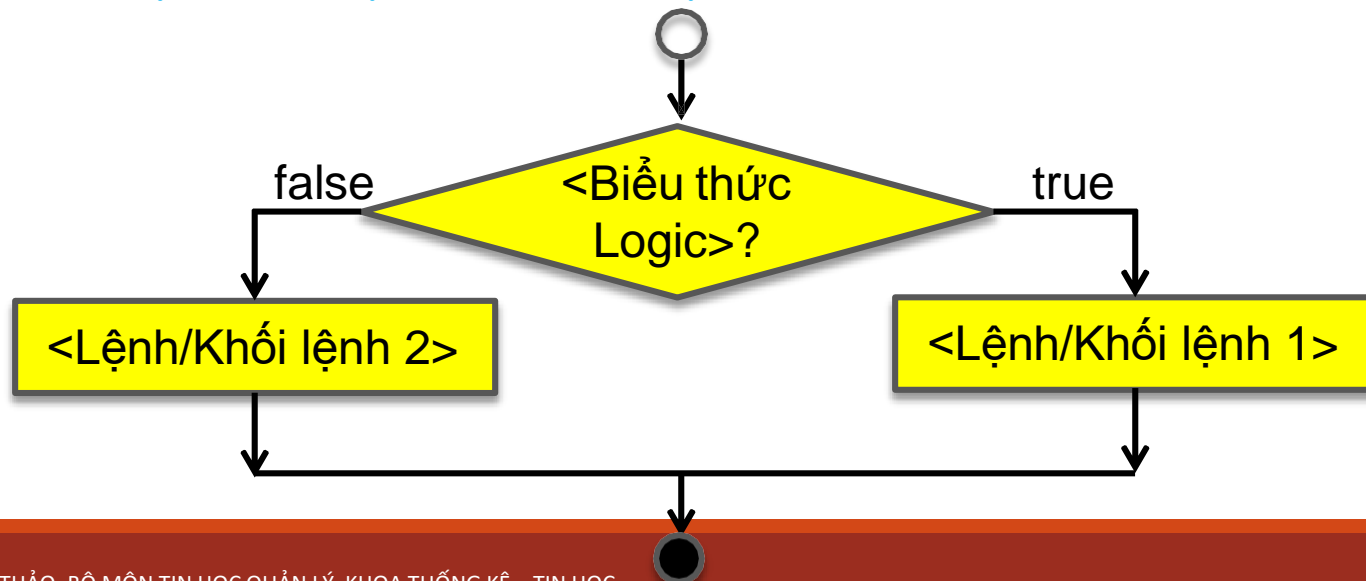
```
}
```

```
else
```

```
{
```

```
    //Lệnh hoặc Khối lệnh 2
```

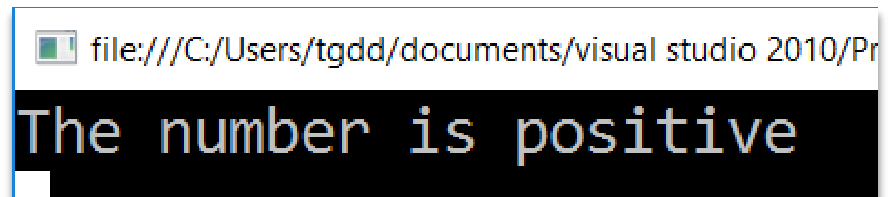
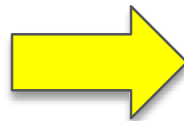
```
}
```



CẤU TRÚC ĐIỀU KIỆN IF – DẠNG 2

- ❑ **Ví dụ:** viết chương trình giải bài toán theo sơ đồ khối sau

```
int num = 10;  
if (num < 0)  
{  
    Console.WriteLine("The number is negative");  
}  
else  
{  
    Console.WriteLine("The number is positive");  
}
```



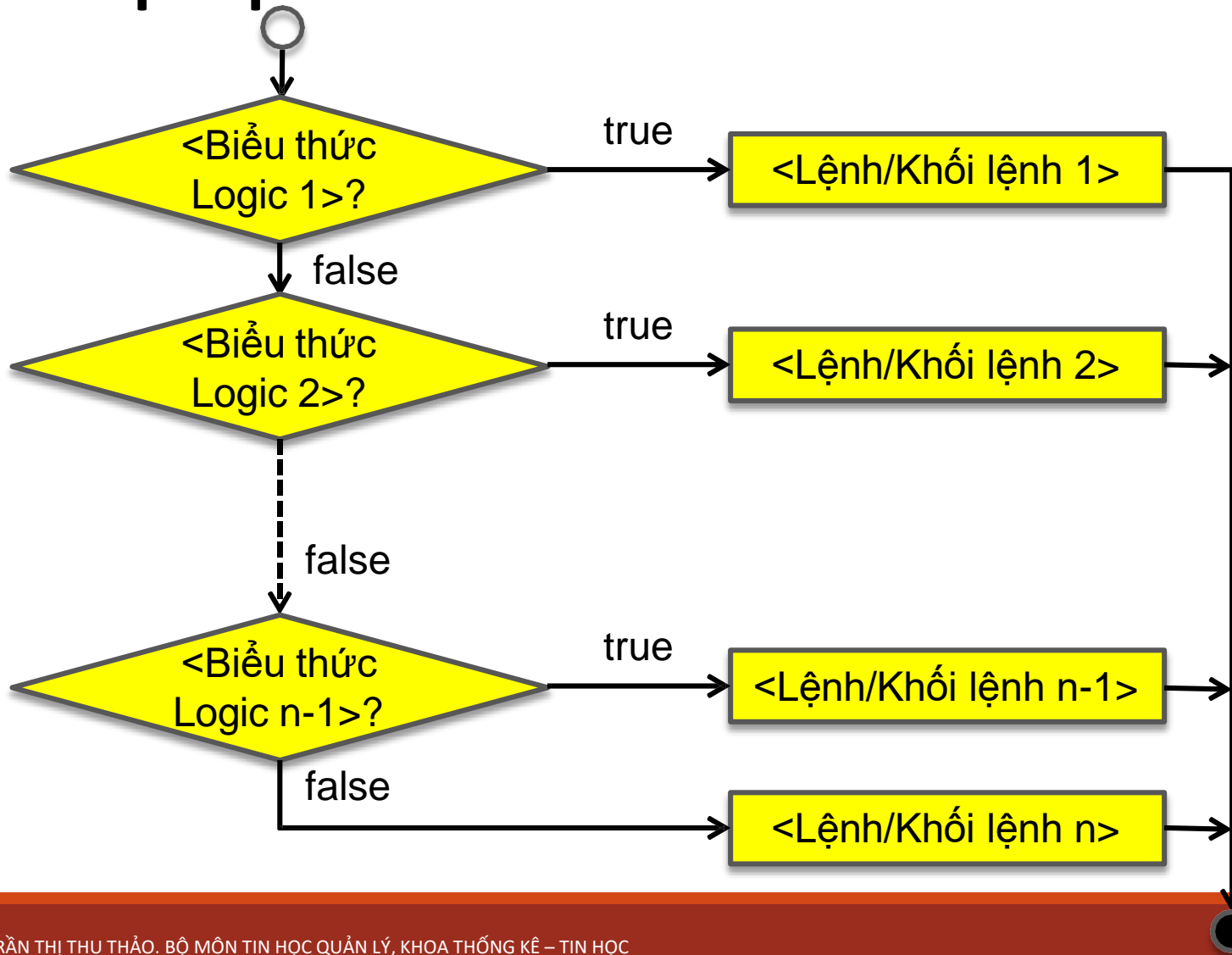
CẤU TRÚC ĐIỀU KIỆN IF – DẠNG 3

❑ Cú pháp

```
if (<Biểu thức Logic 1>
{
    //Lệnh/ khối lệnh 1
}
else if (<Biểu thức Logic 2>)
{
    //Lệnh/ khối lệnh 2
}
...
else if (<Biểu thức Logic n-1>)
{
    //Lệnh/ khối lệnh n-1
}
[ else
{
    //Lệnh/ khối lệnh n
} ]
```

CẤU TRÚC ĐIỀU KIỆN IF – DẠNG 3

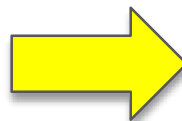
□ Cú pháp



CẤU TRÚC ĐIỀU KIỆN IF – DẠNG 3

- ❑ **Ví dụ:** viết chương trình giải bài toán theo sơ đồ khối sau

```
int num = 13;  
if (num < 0)  
{  
    Console.WriteLine("The number is negative");  
}  
else if ((num % 2) == 0)  
{  
    Console.WriteLine("The number is even");  
}  
else  
    Console.WriteLine("The number is odd");
```



file:///C:/Users/tgdd/documents/visual st

The number is odd

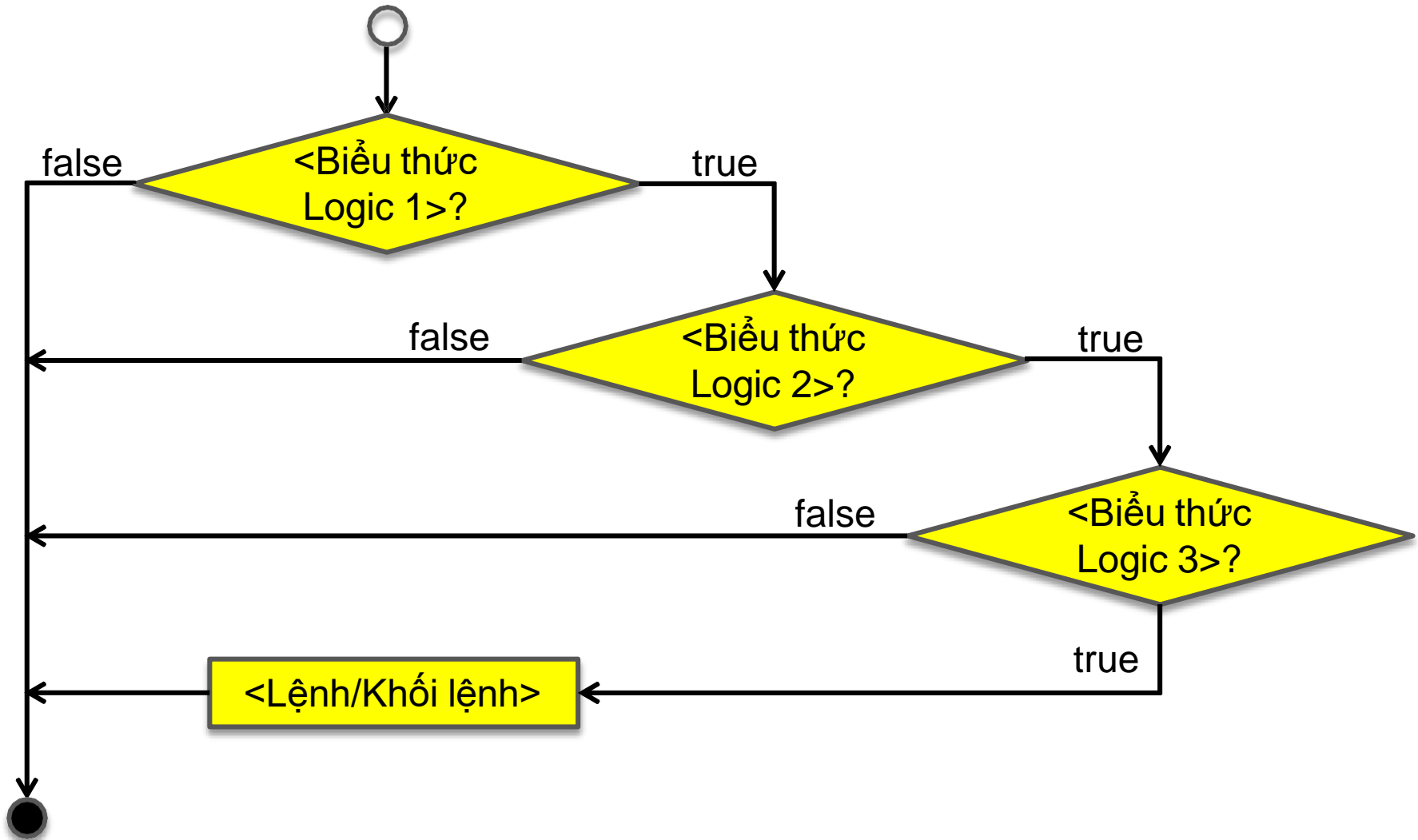
SỰ LỒNG NHAU CỦA CÁC CẤU TRÚC ĐIỀU KIỆN

❑ Cú pháp

```
if (<Biểu thức Logic 1>
{
    ...
    if (<Biểu thức Logic 2>)
    {
        ...
        if (<Biểu thức Logic 3>)
        {
            ...
        }
        ...
    }
    ...
}
```

SỰ LỒNG NHAU CỦA CÁC CẤU TRÚC ĐIỀU KIỆN

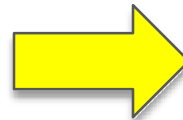
□ Cú pháp



SỰ LÒNG NHAU CỦA CÁC CẤU TRÚC ĐIỀU KIỆN

❑ **Ví dụ:** viết chương trình giải bài toán theo sơ đồ khối sau

```
int yrsOfService= 3;
double salary = 1500;
int bonus = 0;
if (yrsOfService< 5)
{
    if (salary < 500)
        bonus = 100;
    else
        bonus = 200;
}
else
    bonus = 700;
```



file:///C:/Users/tgdd/documents/visual stl
Bonus amount: 200

```
Console.WriteLine("Bonus amount: " + bonus);
```

CẤU TRÚC RẺ NHÁNH SWITCH ... CASE

❑ Cú pháp

switch (<Biểu thức nguyên>)

{

case < n_1 >:

<Lệnh/Khối lệnh 1>; [**break**];

case < n_2 >:

<Lệnh/Khối lệnh 2>; [**break**];

...

case < n_k >:

<Lệnh/Khối lệnh k>; [**break**];

[**default** :

<Lệnh/Khối lệnh k+1>; **break**;]

}

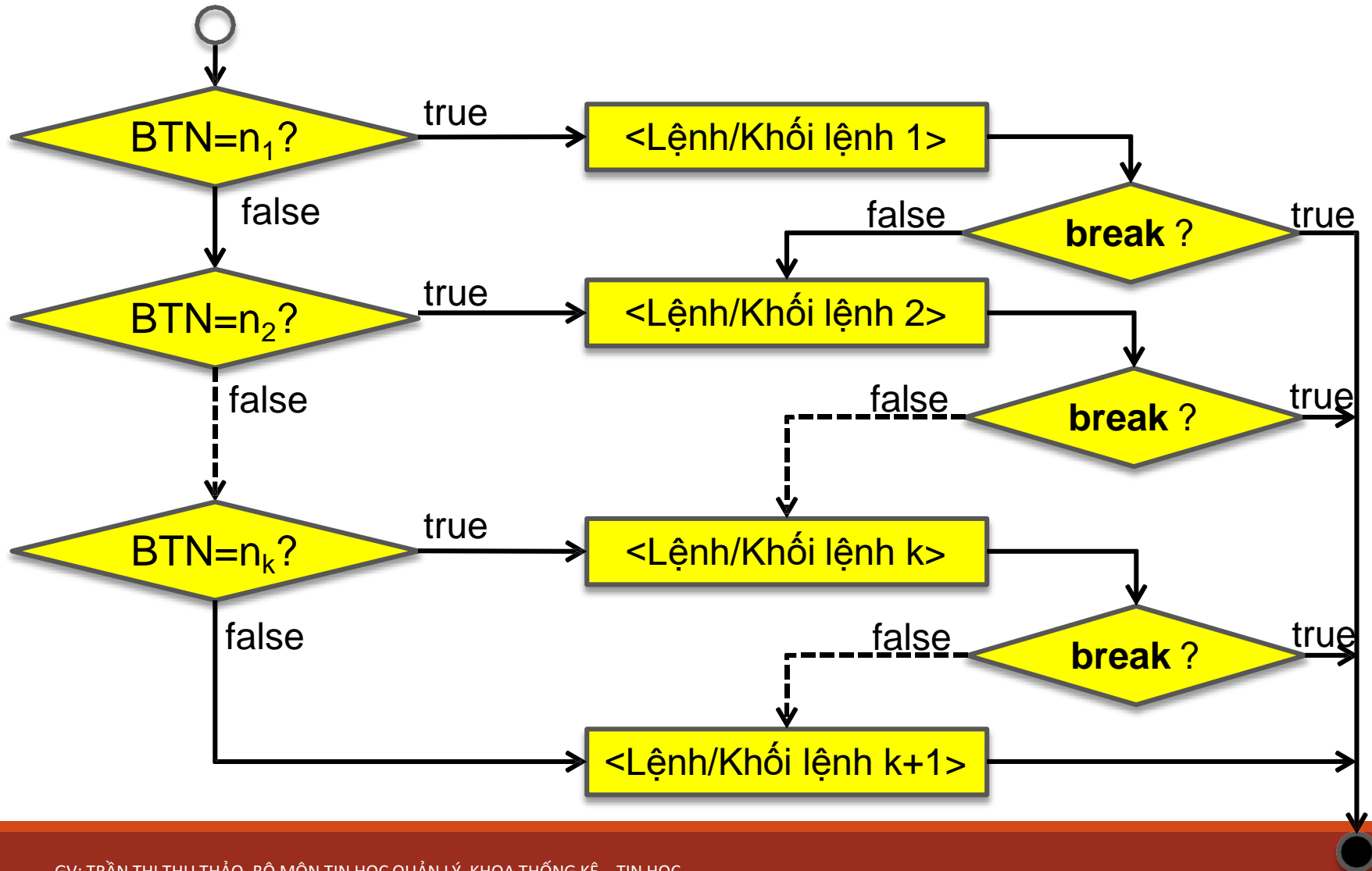
CẤU TRÚC RẺ NHÁNH SWITCH ... CASE

■ Trong đó:

- **<Biểu thức nguyên>**: là một biến hoặc biểu thức có giá trị nguyên;
- **<n₁>, <n₂>, ... <n_k>**: là một giá trị hằng số, có kiểu số nguyên, ký tự hoặc kiểu logic (true/false);
- Từ khóa **break**: có thể được sử dụng hoặc không; được sử dụng để thoát khỏi thân **switch**;
- Từ khóa **default**: có thể được sử dụng hoặc không; có chức năng tương tự **else** trong cấu trúc **if**;
- Những bài toán được giải bằng **if ... else if ... else** thì có thể giải được bằng cấu trúc **switch ... case**;

CẤU TRÚC RẺ NHÁNH SWITCH ... CASE

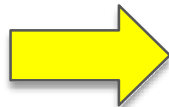
□ Cú pháp



CẤU TRÚC Rẽ NHÁNH SWITCH ... CASE

- ❑ **Ví dụ 1:** nhập từ bàn phím một số nguyên từ 1 → 7, in lên màn hình tên của ngày trong tuần. Nhập sai thì thông báo lỗi.

```
int day;  
Console.WriteLine("Nhap so nguyen (1-->7): ");  
day=Convert.ToInt32(Console.ReadLine());  
switch (day)  
{  
    case 1:  
        Console.WriteLine("Sunday"); break;  
    case 2:  
        Console.WriteLine("Monday"); break;  
    case 3:  
        Console.WriteLine("Tuesday"); break;  
    case 4:  
        Console.WriteLine("Wednesday"); break;  
    case 5:  
        Console.WriteLine("Thursday"); break;  
    case 6:  
        Console.WriteLine("Friday"); break;  
    case 7:  
        Console.WriteLine("Saturday"); break;  
    default:  
        Console.WriteLine("Vui long nhap so tu 1-->7 !!!"); break;  
}
```



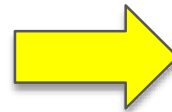
file:///C:/Users/tgdd/documents/visual studio 2010/Pi
Nhap so nguyen (1-->7):
1
Sunday

Luôn có ít nhất một từ khóa **break** trong khối lệnh sau **default**

CẤU TRÚC RẺ NHÁNH SWITCH ... CASE

- ❑ **Ví dụ 2:** Nhập điểm thi từ 0 → 10, in lên màn hình xếp loại kết quả học tập. Nhập sai thì bỏ qua.

```
int diem; Console.WriteLine("Nhap diem thi (0-->10): ");
diem=Convert.ToInt32(Console.ReadLine());
switch (diem)
{
    case 0:
    case 1:
    case 2:
    case 3: Console.WriteLine("Kem\n");break;
    case 4: Console.WriteLine("Yeu\n");break;
    case 5:
    case 6: Console.WriteLine("Trung binh\n");break;
    case 7:
    case 8: Console.WriteLine("Kha\n");break;
    case 9:
    case 10: Console.WriteLine("Gioi\n");break;
}
```

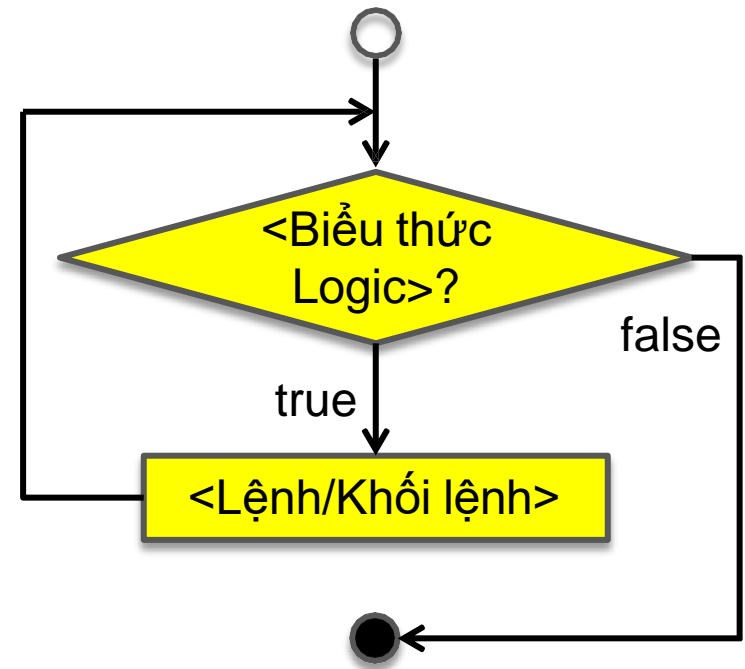


```
file:///C:/Users/tgdd/documents/visual studio 2010/
Nhap diem thi (0-->10):
9
Gioi
```

CẤU TRÚC WHILE

□ Cú pháp

while (<Biểu thức Logic>)
 <Lệnh/Khối lệnh>



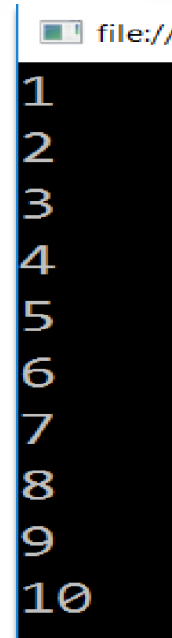
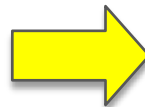
□ Hoạt động của toán tử while

- **Bước 1:** tính giá trị của <Biểu thức Logic>
- **Bước 2:**
 - Nếu có giá trị **True** thì thực hiện <Lệnh/Khối lệnh> rồi quay lại **Bước 1**;
 - Nếu có giá trị **False** thì kết thúc;

CẤU TRÚC WHILE

- ❑ **Ví dụ 1:** in các số từ 1 đến 10, mỗi số nằm trên một dòng.
Yêu cầu sử dụng cấu trúc **while**.

```
int i=1;  
while (i <= 10) //i<=10 : là biểu thức Logic  
{  
    Console.WriteLine(i); //Công việc cần lặp lại  
    i++; //i: được gọi là biến chạy, xác định số lần lặp lại  
}
```



```
file://  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

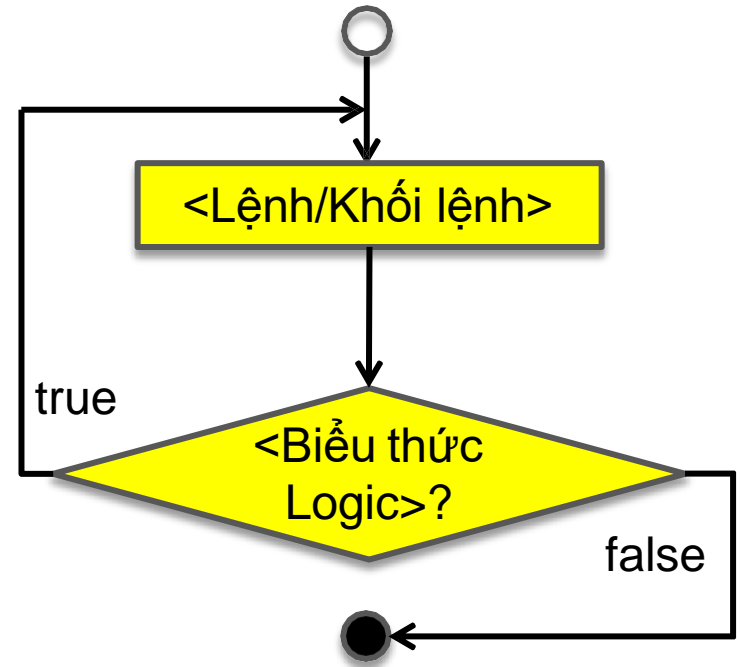

CẤU TRÚC DO - WHILE

□ Cú pháp

do

<Lệnh/Khối lệnh>

while (<Biểu thức Logic>);



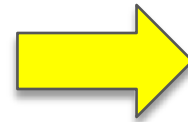
□ Hoạt động của toán tử do-while

- **Bước 1:** thực hiện <Lệnh/Khối lệnh>
- **Bước 2:** tính giá trị của <Biểu thức Logic>
 - Nếu có giá trị **đúng** thì quay lại **Bước 1**;
 - Nếu có giá trị **sai** thì kết thúc;

CẤU TRÚC DO - WHILE

- ❑ **Ví dụ 1:** in các số từ 1 đến 10, mỗi số trên một dòng. Yêu cầu sử dụng cấu trúc **do-while**.

```
int i = 1;
do
{
    Console.WriteLine(i);
    i++;
} while (i <= 10);
```

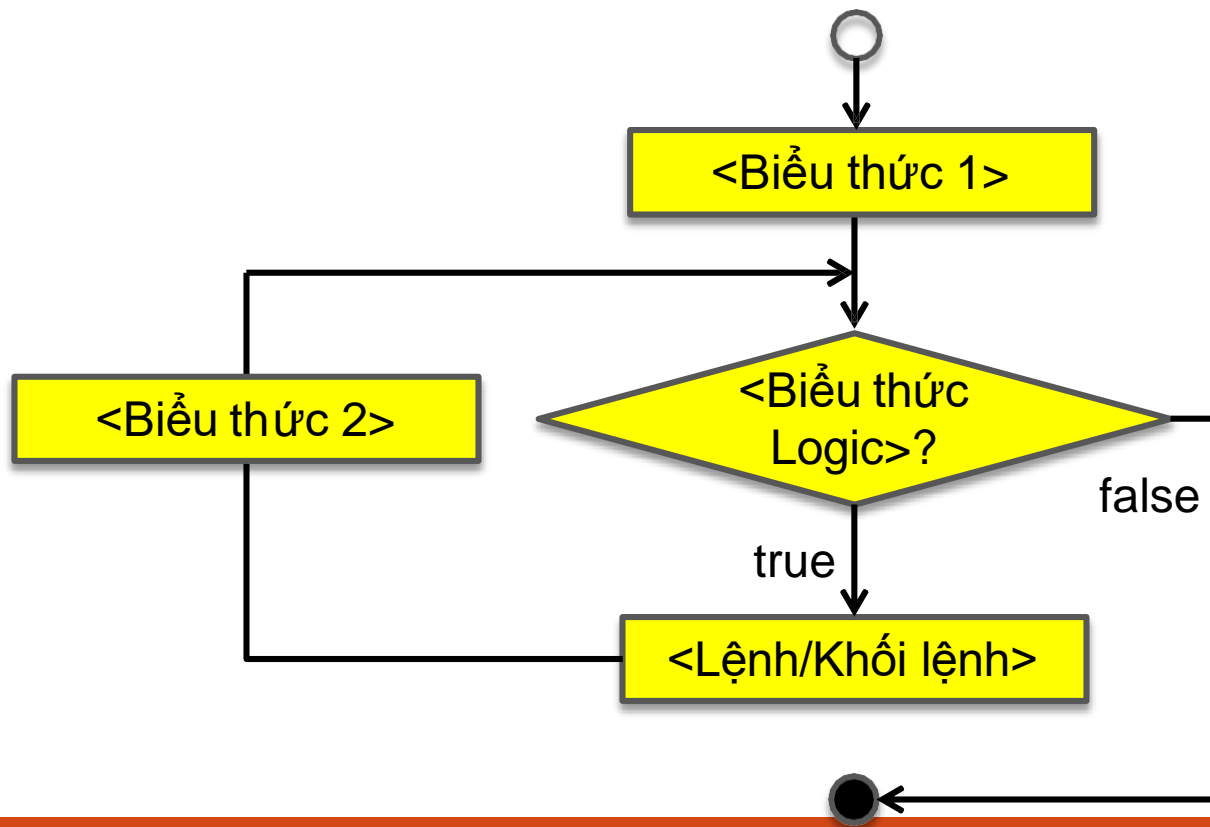
A screenshot of a console window with a black background and white text. The window title bar shows a file icon and the text 'file://'. The output consists of the numbers 1 through 10, each on a new line.

1
2
3
4
5
6
7
8
9
10

CẤU TRÚC FOR

❑ Cú pháp

for([<B.Thức 1>];<B.Thức Logic>;[< B.Thức 2 >])
 <Lệnh/Khối lệnh>



CẤU TRÚC FOR

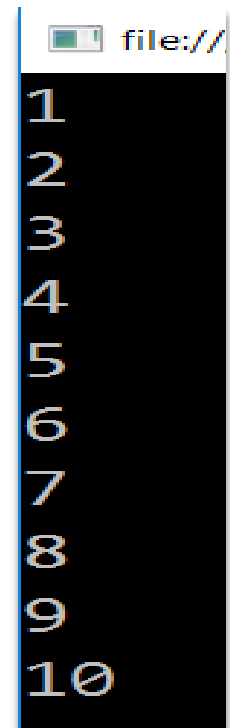
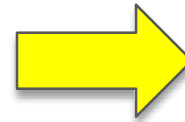
❑ Hoạt động của toán tử for

- **Bước 1:** Thực hiện <Biểu thức 1>
- **Bước 2:** Tính giá trị của <Biểu thức Logic>
 - Nếu có giá trị **False** thì kết thúc;
 - Nếu có giá trị **True** thì thực hiện <Lệnh/Khối lệnh> rồi xuống **Bước 3**;
- **Bước 3:**
 - Thực hiện <Biểu thức 2>
 - Quay về **Bước 2**

CẤU TRÚC FOR

- ❑ **Ví dụ 1:** in các số từ 1 đến 10, mỗi số nằm trên một dòng. Yêu cầu sử dụng cấu trúc **for**.

```
int i; //i: biến chạy  
for (i = 1; i <= 10; i++)  
    Console.WriteLine(i);
```



```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

CÂU LỆNH NHẢY BREAK, CONTINUE

❑ Lệnh break

- Được sử dụng để thoát khỏi cấu trúc lặp gần nhất

while (<Điều kiện lặp>)

{

...

if (<Biểu thức logic>)

break;

...

}

...

Thoát khỏi vòng lặp



CÂU LỆNH NHẢY BREAK, CONTINUE

❑ Lệnh break

do

{

...

if (<Biểu thức logic>)

break;

...

}

while (<Điều kiện lặp>);

...

Thoát khỏi vòng lặp

CÂU LỆNH NHẢY BREAK, CONTINUE

❑ Lệnh break

for (<Lệnh 1>; <Điều kiện lặp>; <Lệnh 2>)

{

...

if (<Biểu thức logic>)

break;

...

}

...

Thoát khỏi vòng lặp

CÂU LỆNH NHẢY BREAK, CONTINUE

❑ Lệnh continue

- Được sử dụng để bỏ qua các lệnh còn lại của vòng lặp và bắt đầu chu trình lặp tiếp theo.

```
while (<Điều kiện lặp>)  
{  
    ...  
    if (<Biểu thức logic>)  
        continue;  
    ...  
}  
...
```

Bắt đầu chu kỳ lặp mới

CÂU LỆNH NHẢY BREAK, CONTINUE

❑ Lệnh continue

- Được sử dụng để bỏ qua các lệnh còn lại của vòng lặp và bắt đầu chu trình lặp tiếp theo.

do

{

...

if (<Biểu thức logic>

continue;

...

}

while (<Điều kiện lặp>;

...

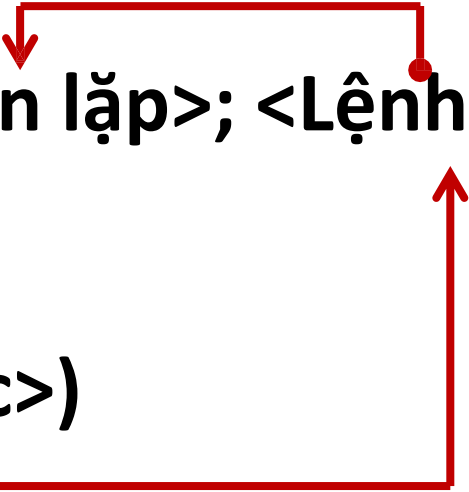
Bắt đầu chu kỳ lặp mới

CÂU LỆNH NHẢY BREAK, CONTINUE

❑ Lệnh continue

- Được sử dụng để bỏ qua các lệnh còn lại của vòng lặp và bắt đầu chu trình lặp tiếp theo.

```
for (<Lệnh 1>; <Điều kiện lặp>; <Lệnh 2>)  
{  
    ...  
    if (<Biểu thức logic>)  
        continue;  
    ...  
}  
...
```

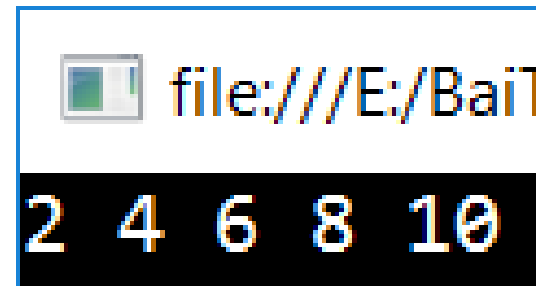
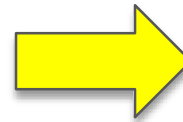


Bắt đầu chu kỳ lặp mới

CÂU LỆNH NHẢY BREAK, CONTINUE

- ❑ **Ví dụ 1:** In lên màn hình một dãy các số chẵn liên tục từ 1 đến 10.

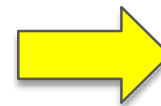
```
int i;  
for (i = 1; i <= 10; i++ )  
{  
    if (i % 2 != 0) continue;  
    Console.Write(i + " ");  
}  
Console.ReadKey();
```



CÂU LỆNH NHẢY BREAK, CONTINUE

- ❑ **Ví dụ 2:** Nhập từ bàn phím một số nguyên n và n số nguyên; Việc nhập sẽ dừng lại cho đến khi đủ n số nguyên hoặc số 0 được nhập vào. In lên màn hình tổng của các số nguyên dương đã được nhập.

```
int i,n,x,S=0;
Console.Write("Nhap n= ");
n=int.Parse(Console.ReadLine());
for (i = 1; i <= n; i++)
{
    Console.Write("So thu " + i + ":");
    x=int.Parse(Console.ReadLine());
    if (x<0) continue;
    if (x == 0)
        break;
    else S += x;
}
Console.Write("S=" + S);
```



file:///E:/BaiTai

```
Nhap n= 10
So thu 1:1
So thu 2:2
So thu 3:3
So thu 4:-4
So thu 5:5
So thu 6:0
S=11
```

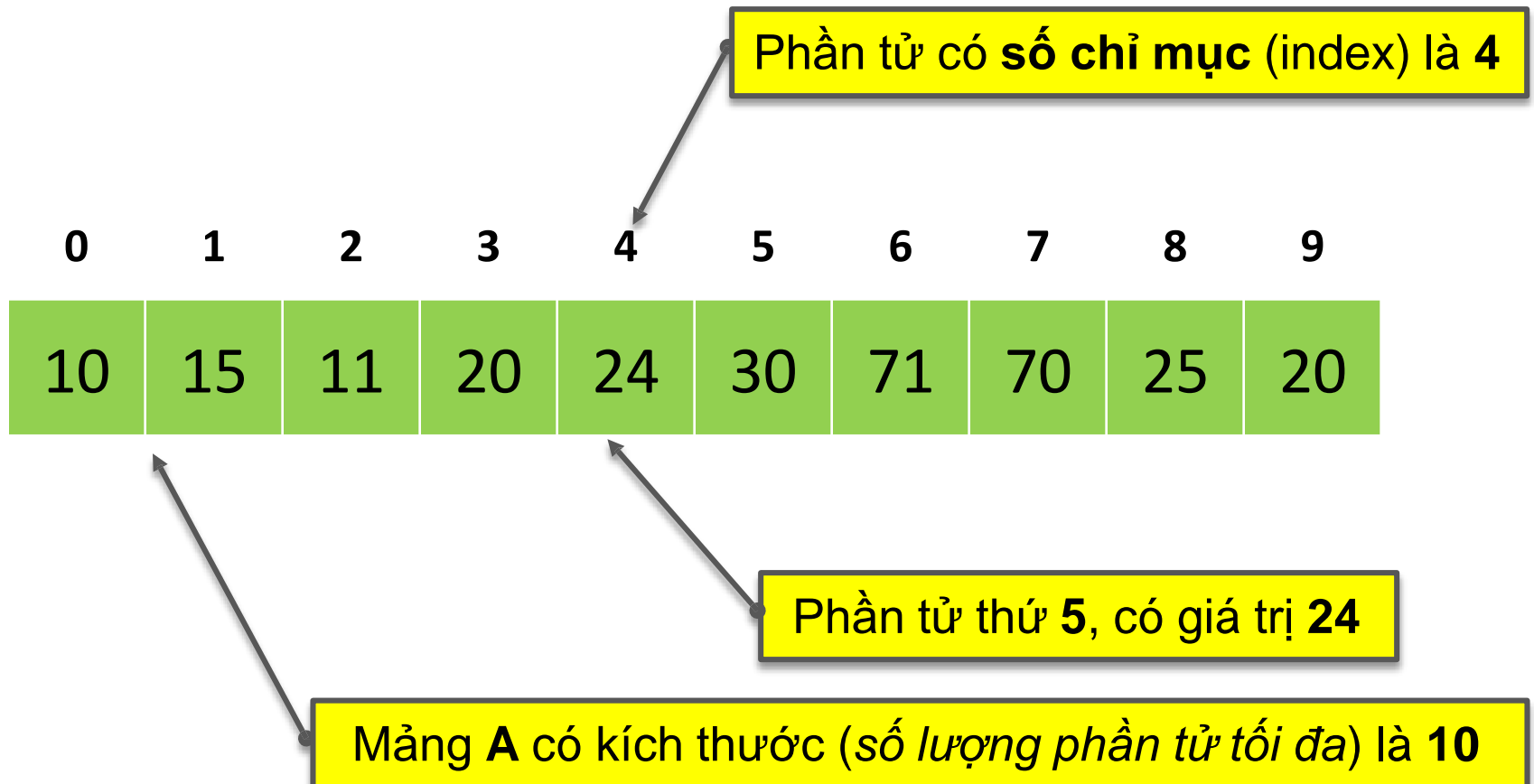
MẢNG

- ❑ **Mảng một chiều**
- ❑ **Mảng hai chiều**

MẢNG – MẢNG MỘT CHIỀU

❑ Cấu trúc mảng một chiều

- Ví dụ 4.2: mảng một chiều có tên là **A**, lưu trữ **10** số nguyên.



MẢNG – MẢNG MỘT CHIỀU

❑ Khai báo mảng:

■ Cách 1:

<Tên kiểu>[] <Tên mảng>;

<Tên mảng> = new <Tên kiểu> [size];

size là số lượng phần tử tối đa của mảng, hay còn gọi là kích thước của mảng.

Ví dụ 4.3a:

```
int[] A;
```

```
A = new int[100];
```

Khai báo cố định số phần tử

```
//Khai báo mảng
```

```
//Khởi tạo mảng
```


MẢNG – MẢNG MỘT CHIỀU

❑ Khai báo mảng:

- Cách 2: vừa khai báo – vừa khởi tạo mảng

<Tên kiểu>[] <Tên mảng> = new <Tên kiểu>[size];

Ví dụ 4.3b:

```
int[] A = new int[100];
```

Khai báo cố định số phần tử

- Cách 3: cấp phát động số phần tử cho mảng

Ví dụ 4.3c:

```
int n;  
int[] A; //Khai báo mảng
```

```
n = int.Parse(Console.ReadLine());  
A = new int[n]; //Cấp phát động số phần tử
```

MẢNG – MẢNG MỘT CHIỀU

❑ Khai báo mảng:

- Ví dụ 4.4: Khai báo một mảng có tên là **Diem**, dùng để lưu trữ điểm thi của tối đa **10** sinh viên:

- Cách 1:

```
float[] Diem;           //Khai báo mảng  
Diem = new float[10];   //Khởi tạo mảng
```

- Cách 2:

```
//Vừa khai báo vừa khởi tạo mảng  
float[] Diem = new float[10];
```

MẢNG – MẢNG MỘT CHIỀU

❑ Khai báo mảng:

- Ví dụ 4.4: Khai báo một mảng có tên là **Diem**, dùng để lưu trữ điểm thi của tối đa **10** sinh viên:
 - Cách 3:

```
int n;  
float[] Diem; //Khai báo mảng  
  
Console.Write("So luong SV: ");  
n = int.Parse(Console.ReadLine());  
Diem= new float[n]; //Cấp phát động số phần tử
```

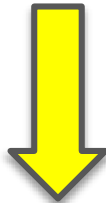
MẢNG – MẢNG MỘT CHIỀU

❑ Gán giá trị cho mảng:

■ Cách 1: Khởi gán giá trị cho mảng

<Tên kiểu>[] <Tên mảng> = {gtri1, gtri2, ..., gtriN};

```
int[] A = {10,15,11,20,24,30,71,70,25,20};
```



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
10	15	11	20	24	30	71	70	25	20

MẢNG – MẢNG MỘT CHIỀU

❑ Gán giá trị cho mảng:

- Cách 2: Gán giá trị cho từng phần tử trong mảng

<Tên mảng>[<Số chỉ mục>] = <Giá trị>;

```
int[] A = new int[10];
```

```
A[0] = 10;
```

```
A[1] = 15;
```

```
A[2] = 11;
```

```
A[3] = 20;
```

```
A[4] = 24;
```

```
A[5] = 30;
```

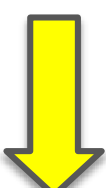
```
A[6] = 71;
```

```
A[7] = 70;
```

```
A[8] = 25;
```

```
A[9] = 20;
```

Phần tử **đầu tiên** có
số chỉ mục là **0**



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
10	15	11	20	24	30	71	70	25	20

MẢNG – MẢNG MỘT CHIỀU

- ❑ **Nhập giá trị từ bàn phím cho các phần tử trong mảng**
 - **Ví dụ 4.7:** Nhập từ bàn phím **một số nguyên n** và **điểm thi của n sinh viên**.

```
int n, i;  
float[] Diem; //Khai báo mảng  
  
Console.Write("So luong SV: ");  
n = int.Parse(Console.ReadLine());  
Diem= new float[n]; //Cấp phát động số phần tử  
  
Console.WriteLine("Nhap diem của {0} SV: ", n);  
for (i = 0; i < n; i++)  
{  
    Console.Write(" -SV {0}: ", i + 1);  
    Diem[i] = float.Parse(Console.ReadLine());  
}
```

Sử dụng pp khai báo động

Nhập giá trị cho pt có số chỉ mục i

MẢNG – MẢNG MỘT CHIỀU

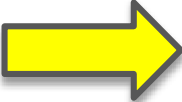
❑ Truy xuất mảng:

Truy xuất phần tử thứ i

Ví dụ 4.9: In lên màn hình điểm thi của 5 SV trong mảng **Diem**

```
int i = 0;
float[] Diem = {(float)5.5,8,(float)6.5,10,7};
while (i < Diem.Length)
{
    Console.WriteLine("Diem {0} = {1}",i+1,Diem[i]);
    i++;
}
```

<Tên mảng>.Length → cho biết chiều dài của mảng



```
Diem 1 = 5.5
Diem 2 = 8
Diem 3 = 6.5
Diem 4 = 10
Diem 5 = 7
```

MẢNG – MẢNG MỘT CHIỀU

❑ Cấu trúc lặp **foreach**

■ Cú pháp:

foreach (<Tên kiểu> <Tên biến> **in** <Tên mảng>)
 <Lệnh>/<Khối lệnh>

■ Chức năng: duyệt qua từng phần tử trong mảng

MẢNG – MẢNG MỘT CHIỀU

❑ Cấu trúc lặp **foreach**

■ Ví dụ 4.11:

```
int[] A = {1,2,3,4,5};  
//Cach 1:  
Console.WriteLine("Cach 1:");  
for (int i = 0; i < 5; i++)//A.Length hoặc A.GetLength(0)  
    Console.WriteLine(A[i]);  
  
//Cach 2:  
Console.WriteLine("Cach 2:");  
foreach (int So in A)  
    Console.WriteLine(So);
```



```
Cach 1:  
1  
2  
3  
4  
5  
Cach 2:  
1  
2  
3  
4  
5
```

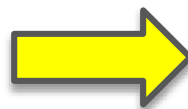
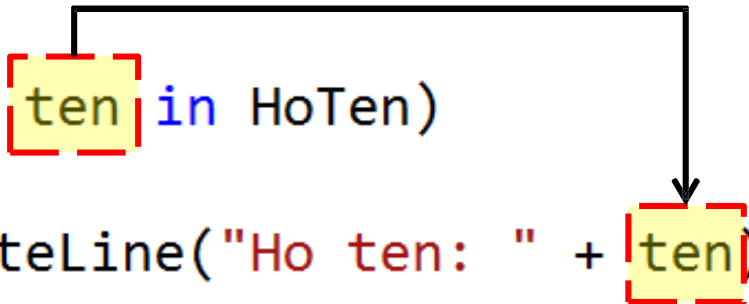
MẢNG – MẢNG MỘT CHIỀU

❑ Cấu trúc lặp **foreach**

■ Ví dụ 4.11:

```
string[] HoTen = {"Nguyen An", "Le Binh", "Tran Ngoc",  
                  "Pham Thanh", "Nguyen Hanh"};
```

```
foreach (string ten in HoTen)  
{  
    Console.WriteLine("Ho ten: " + ten);  
}
```



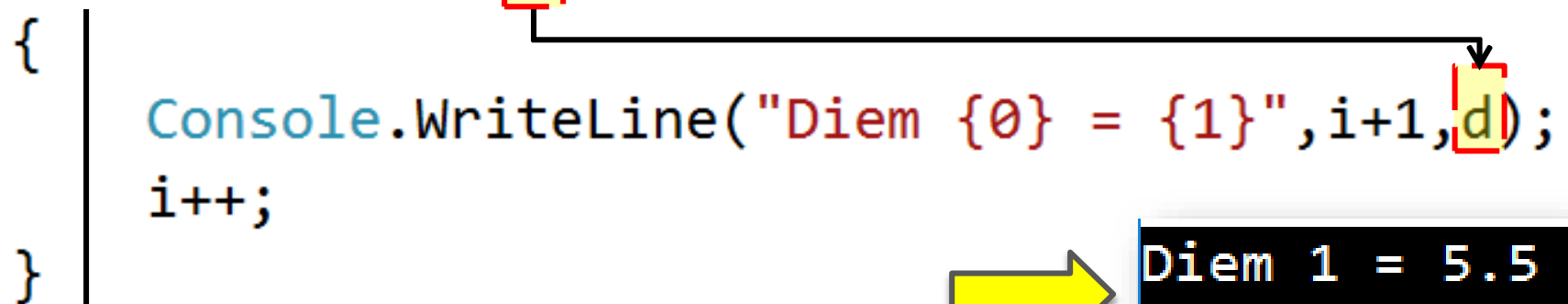
```
Ho ten: Nguyen An  
Ho ten: Le Binh  
Ho ten: Tran Ngoc  
Ho ten: Pham Thanh  
Ho ten: Nguyen Hanh
```

MẢNG – MẢNG MỘT CHIỀU

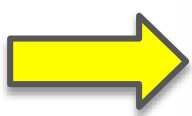
❑ Cấu trúc lặp **foreach**

■ Ví dụ 4.12:

```
int i = 0;  
float[] Diem = {(float)5.5,8,(float)6.5,10,7};  
foreach (float d in Diem)  
{  
    Console.WriteLine("Diem {0} = {1}",i+1,d);  
    i++;  
}
```



Biến **d** có kiểu **float**, sẽ được gán bằng giá trị của các phần tử trong mảng **Diem**



```
Diem 1 = 5.5  
Diem 2 = 8  
Diem 3 = 6.5  
Diem 4 = 10  
Diem 5 = 7
```

MẢNG – MẢNG MỘT CHIỀU

❑ Sao chép mảng

■ Hàm Array.Copy()

Array.Copy(<Mảng nguồn>, <Mảng đích>, <Số phần tử>)

- **<Mảng nguồn>**: mảng cần copy
- **<Mảng đích>**: mảng mới được tạo ra
- **<Số phần tử>**: số phần tử cần sao chép từ **<Mảng nguồn>** sang **<Mảng đích>**, lớn nhất là bằng kích thước của **<Mảng nguồn>**

MẢNG – MẢNG MỘT CHIỀU

❑ Sao chép mảng

■ Cách 1: Sử dụng hàm `Array.Copy()`

- Ví dụ 4.13:

```
int[] A = { 5, 10, 15, 20, 25 };  
int[] B = new int[5];
```

```
Array.Copy(A, B, 5); //Sao chép toàn bộ Mảng A --> B
```

```
Console.Write("Mang A: ");  
foreach (int x in A)  
    Console.Write(x + " ");
```

```
Console.Write("Mang B: ");  
foreach (int x in B)  
    Console.Write(x + " ");
```



```
Mang A: 5 10 15 20 25  
Mang B: 5 10 15 20 25
```

MẢNG – MẢNG MỘT CHIỀU

❑ Sao chép mảng

■ Cách 2: Sao chép từng phần tử trong mảng


- Ví dụ 4.14:

```
int[] A = { 5, 10, 15, 20, 25 };
```

```
int[] B = new int[5];
```

```
for (int i = 0; i < A.Length; i++)
```

```
    B[i] = A[i]; //Sao chép từng phần tử A --> B
```

	[0]	[1]	[2]	[3]	[4]
A	5	10	15	20	25
					
B	5	10	15	20	25

MẢNG – MẢNG MỘT CHIỀU

❑ Sao chép mảng

■ Phép tham chiếu mảng bằng phép gán

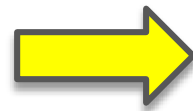
- Ví dụ 4.14:

```
int[] A = { 5, 10, 15, 20, 25 };
```

```
int[] B;
```

```
B = A;
```

```
foreach (int x in B)  
    Console.Write(x + " ");
```



5 10 15 20 25

Lưu ý:

- Tham chiếu là tạo ra một biến mảng thứ 2, nhưng cùng trỏ đến cùng không gian lưu trữ của mảng ban đầu;
- Mọi tác động trên mảng B, đều ảnh hưởng đến mảng A.

MẢNG – MẢNG MỘT CHIỀU

❑ Sao chép mảng

■ Phép tham chiếu mảng bằng phép gán

- Ví dụ 4.15:

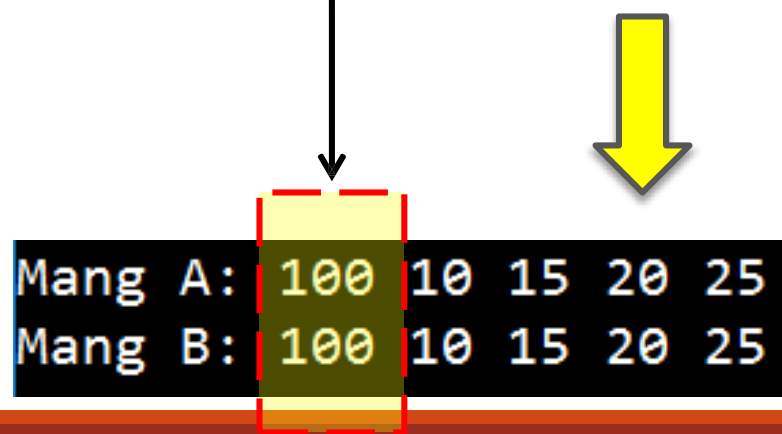
```
int[] A = { 5, 10, 15, 20, 25 };  
int[] B = new int[5];
```

```
B = A;
```

```
→ B[0] = 100;
```

```
Console.Write("Mang A: ");  
foreach (int x in A)  
    Console.Write(x + " ");
```

```
Console.WriteLine();  
Console.Write("Mang B: ");  
foreach (int x in B)  
    Console.Write(x + " ");
```



Mang A:	100	10	15	20	25
Mang B:	100	10	15	20	25

MẢNG – MẢNG HAI CHIỀU

❑ Cấu trúc mảng hai chiều

The diagram illustrates a 2D array structure. It features a grid of 5 rows and 7 columns. The rows are indexed from 0 to 4, labeled 'i dòng' on the left with a downward arrow. The columns are indexed from 0 to 6, labeled 'j cột' on top with a rightward arrow. To the right of the grid, the notation $A[i, j]$ is shown with an arrow pointing to the cell at row 2, column 6. The grid cells are light green and contain the following values:

	j cột	0	1	2	3	4	5	6
i dòng	0	5	7	2	1	10	4	3
1	1	1	2	3	5	9	2	6
2	2	6	4	5	2	3	7	5
3	3	1	9	10	7	8	8	2
4	4	2	6	5	3	2	1	10

MẢNG – MẢNG HAI CHIỀU

❑ Khai báo mảng hai chiều

■ Cách 1:

<Tên kiểu>[,] <Tên mảng>;

<Tên mảng> = new <Tên kiểu> [size1, size2];

■ Cách 2:

<Tên kiểu>[,] <Tên mảng>=new<Tên kiểu>[size1, size2];

size1: số dòng

size2: số cột

MẢNG – MẢNG HAI CHIỀU

❑ Khai báo mảng hai chiều

■ Cách 3: Cấp phát động kích thước mảng

```
int size1, size2;  
int[,] A;//Khai báo mảng  
  
size1=int.Parse(Console.ReadLine()); //Chiều 1 - Dòng  
size2=int.Parse(Console.ReadLine()); //Chiều 2 - Cột  
A = new int[size1, size2];
```

MẢNG – MẢNG HAI CHIỀU

❑ Khởi gán giá trị cho mảng hai chiều

■ Ví dụ 4.16:

```
int[,] A = { { 1, 2, 3, 4 }  
            , { 4, 3, 2, 1 }  
            , { 1, 2, 3, 4 } };
```

<Tên mảng>.GetLength(<Số chiều>) →
Trả về số phần tử ở Chiều tương ứng

```
for (int i = 0; i < A.GetLength(0); i++) //Duyệt qua từng Dòng  
{  
    for (int j = 0; j < A.GetLength(1); j++) //Duyệt qua từng Cột  
        Console.Write(A[i, j] + " ");  
    Console.WriteLine();  
}
```



1	2	3	4
4	3	2	1
1	2	3	4

MẢNG – MẢNG HAI CHIỀU


❑ Nhập giá trị cho từng phần tử trong mảng

- Ví dụ 4.17: nhập từ bàn phím giá trị của các phần tử trong ma trận **3x4**

```
int[,] A = new int[3, 4];
```

```
for (int i = 0; i < 3; i++) //Duyệt qua từng Dòng
    for (int j = 0; j < 4; j++) //Duyệt qua từng Cột
    {
        Console.WriteLine("Nhập A[{0},{1}] = ", i, j);
        A[i, j] = int.Parse(Console.ReadLine());
    }
```

Nhập giá trị cho phần tử **A[i , j]**



Nhap	A[0,0]	=	1
Nhap	A[0,1]	=	2
Nhap	A[0,2]	=	3
Nhap	A[0,3]	=	4
Nhap	A[1,0]	=	4
Nhap	A[1,1]	=	3
Nhap	A[1,2]	=	2
Nhap	A[1,3]	=	1
Nhap	A[2,0]	=	1
Nhap	A[2,1]	=	2
Nhap	A[2,2]	=	3
Nhap	A[2,3]	=	4

MẢNG – MẢNG HAI CHIỀU

❑ Các hàm thuộc class: **System.Array**

<code>Clear()</code>	Hàm tĩnh này gán giá trị mặc định (theo kiểu dữ liệu của mảng) cho tất cả các thành phần của mảng
<code>Copy()</code>	Hàm cho phép sao chép một phần của mảng này lên mảng khác
<code>CopyTo()</code>	Hàm này được sử dụng để sao chép các thành phần của mảng nguồn sang mảng đích
<code>GetLength()</code>	Hàm này trả về số lượng thành phần trên một chiều chỉ định của mảng
<code>IndexOf()</code>	Hàm này trả về chỉ số của giá trị xuất hiện đầu tiên trong mảng một chiều
<code>LastIndexOf()</code>	Hàm trả về chỉ số của giá trị xuất hiện cuối cùng trên mảng một chiều
<code>Length</code>	Thuộc tính này trả ra số lượng thành phần của mảng
<code>Rank</code>	Thuộc tính này trả ra số chiều của mảng
<code>Reverse()</code>	Hàm tĩnh này đảo ngược thứ tự các thành phần của một mảng một chiều
<code>Sort()</code>	Hàm tĩnh này sắp xếp một mảng một chiều kiểu cơ bản

HÀM

- ☐ Khai báo Hàm
- ☐ Lời gọi Hàm
- ☐ Truyền giá trị từ ngoài vào trong Hàm
- ☐ Truyền giá trị từ trong ra ngoài Hàm

KHAI BÁO HÀM

❑ Cú pháp khai báo hàm:

```
[<Phạm_vi>][static]<Kiểu><Tên_Hàm>([<DS/Tham số>])  
{  
    Lệnh/Khối lệnh  
}
```

■ Trong đó:

- <Phạm_vi>: Cho biết phạm vi có thể tiếp cận đến hàm (*xem thêm trang sau*)
- <Kiểu>: Kiểu giá trị của kết quả trả về qua tên hàm
- <Tên_Hàm>: Do người lập trình tự đặt
- <DS/Tham số>: Các tham số sẽ được truyền qua hàm (*xem thêm ở trang sau*)

KHAI BÁO HÀM

❑ Các loại <Phạm_vi> : (Access Modifier)

Phạm vi tăng dần
↓

- **private**: Truy cập bị giới hạn trong phạm vi của class chứa nó. Nếu để trống thì mặc định là **private**.
- **protected**: Truy cập bị giới hạn trong phạm vi của class chứa nó và bất kỳ class con **thừa kế** từ class này.
- **internal**: Truy cập bị giới hạn trong phạm vi của **Solution** hiện tại.
- **protected internal**: Truy cập bị giới hạn trong phạm vi của Solution hiện tại và trong class định nghĩa hoặc các class con.
- **public**: Không có bất kỳ giới hạn nào khi truy cập vào các thành viên công khai (public).

KHAI BÁO HÀM

□ **<DS/Tham số>** (*danh sách tham số*)

■ **Cú pháp:**

[ref/out]<Kiểu> <Biến>, [ref/out] <Kiểu> <Biến>,...

Ví dụ:

ref int b, out float c, int a

■ **Ý nghĩa:**

- Là các biến: mang giá trị từ bên ngoài **VÀO** bên trong hàm (input), hoặc mang giá trị từ bên trong **RA** bên ngoài hàm (output).
- **ref**: mang dữ liệu **VÀO** và **RA**
- **out**: chỉ mang dữ liệu **RA**
- **Để trống**: chỉ mang dữ liệu **VÀO**

KHAI BÁO HÀM

❑ Ví dụ 6.1a: Khai báo hàm **Nhap(n)**

- Khai báo phần **header** của hàm như sau:

```
private static void Nhap(out int n)
```

out: trả về số nguyên được nhập vào

void: kiểu không trả về giá trị

static: bắt buộc sử dụng vì hàm được khai báo cùng class với hàm **main()**

private: giới hạn phạm vi sử dụng trong class hiện hành

KHAI BÁO HÀM

❑ Ví dụ 6.1a: Khai báo hàm **Nhap(n)**

- Hàm được viết đầy đủ như sau:

```
private static void Nhap(out int n)
{
    //Nhập liệu
    Console.Write("Nhap n= ");
    n = int.Parse(Console.ReadLine());
}
```

Phần **thân** (body) của hàm

Phần **đầu** (header) của hàm

KHAI BÁO HÀM

❑ Ví dụ 6.1c: Khai báo hàm InKQ(S)

```
private static void InKQ(int S)
{
    //In kết quả
    Console.Write("Tong S= " + S);
    Console.ReadKey();
}
```

S chỉ truyền tham số vào nên không sử dụng **ref** hoặc **out**

KHAI BÁO HÀM

❑ Một số lưu ý quan trọng

- Muốn đưa vào/ lấy ra giá trị từ hàm thì: sử dụng tham số hoặc qua tên hàm (lấy ra/output);
- Sử dụng **ref** khi: tham số đồng thời **input** và **output** giá trị từ hàm;
- Sử dụng **out** khi: tham số chỉ **output** giá trị từ hàm;
- Để trống: tham số chỉ **input** giá trị từ hàm;
- Một hàm có thể không có tham số nào;
- Vị trí đặt code khai báo hàm:
 - Bên ngoài và dưới hàm main()
 - Hoặc trong một file class khác

KHAI BÁO HÀM

□ Một số lưu ý quan trọng

- Kiểu của hàm phụ thuộc vào Output của hàm:
 - Hàm có kiểu **void**:
 - Kết quả được trả về qua tham số
 - Kết quả được in lên màn hình
 - Kết quả được ghi ra tệp tin
 - Hàm có kiểu **KHÁC void** (int, float, char,...):
 - Kết quả được trả về qua **tên hàm**

LỜI GỌI HÀM

□ Ý nghĩa:

- Khi cần sử dụng một hàm đã định nghĩa, ta cần thực hiện lời gọi hàm;
- Vị trí đặt code để gọi hàm là bên trong hàm main() hoặc trong một hàm khác (hàm gọi hàm);
- Hàm được gọi thông qua tên hàm, theo sau là danh sách tham số (nếu có);
- Sử dụng cú pháp hai ngôi (có phép gán =) khi hàm trả kết quả qua tên hàm (khác kiểu void).

LỜI GỌI HÀM

- Ví dụ 6.2: gọi hàm trong hàm Main()

```
static void Main(string[] args)
{
    int n, S;
    Nhap(out n);
    S=Tinh(n);
    InKQ(S);
}
```

~~private static void~~ **Nhap(out int n)**

~~private static int~~ **Tinh(int n)**

~~private static void~~ **InKQ(int S)**

LỜI GỌI HÀM

- Ví dụ 6.3: gọi hàm trong một hàm khác hàm

```
static void Main(string[] args)
{
    BaiToanSo1();
}
private static void BaiToanSo1()
{
    int n, S;
    Nhap(out n);
    S = Tinh(n);
    InKQ(S);
}
```

Cần phải gọi hàm **BaiToanSo1()** trong hàm **Main()** để kích hoạt chương trình

Các hàm **Nhap()**, **Tinh()**, **InKQ()** được gọi trong hàm **BaiToanSo1()**

TRUYỀN GIÁ TRỊ CHO HÀM

❑ Phương pháp 1: Sử dụng tham số của hàm

- Bổ sung từ khóa khi khai báo hàm
 - **out**: không nhận vào, chỉ lấy ra
 - **ref**: nhận vào và lấy ra
 - **Để trống**: chỉ nhận vào, không lấy ra

`private static void Nhap(out int n)` << khai báo

n: không nhận vào, chỉ lấy ra

`Nhap(out n);` << gọi hàm

TRUYỀN GIÁ TRỊ CHO HÀM

❑ Phương pháp 1: Sử dụng tham số của hàm

`private static void InKQ(int S)` << khai báo

S: chỉ nhận vào, không lấy ra

`InKQ(S);` << gọi hàm

TRUYỀN GIÁ TRỊ CHO HÀM

❑ Phương pháp 1: Sử dụng tham số của hàm

- Bài toán hoán vị, sử dụng **ref**

```
static void Main(string[] args)
{
    int x = 5, y = 10;
    HoanVi(ref x, ref y);
    InKQ(x, y);
}

static void HoanVi(ref int x, ref int y)
{
    int z;
    z = x;
    x = y;
    y = z;
}
```



x=10, y=5

TRUYỀN GIÁ TRỊ CHO HÀM

❑ Phương pháp 2: Trả giá trị qua tên hàm

- Hàm kiểu khác **void**
- Trong thân hàm có ít nhất 1 từ khóa **return** để trả kết quả

```
private static int Tinh(int n)
{
    //Tính tổng
    int S = 0;
    for (int i = 1; i <= n; i++)
        S += i;
    return S;
}
```

Kiểu khác **void**, kết quả sẽ được trả về qua tên hàm

return <x>; kết thúc hàm và trả giá trị <x> về qua tên hàm

TRUYỀN GIÁ TRỊ CHO HÀM

□ Phương pháp 2: Trả giá trị qua tên hàm

- Lời gọi hàm phải ở dạng biểu thức hai ngôi
- Vế phải: là tên hàm cần gọi, kèm tham số (nếu có)
- Vế trái: là 1 biến, sẽ lưu trữ giá trị là kết quả của hàm ở vế phải;

`S = Tinh(n);`

TRUYỀN GIÁ TRỊ CHO HÀM

❑ Phương pháp 3: Sử dụng biến toàn cục

```
static int x = 5, y = 10;  
static void Main(string[] args)  
{  
    HoanVi();  
    InKQ(x, y);  
}  
static void HoanVi()  
{  
    int z;  
    z = x;  
    x = y;  
    y = z;  
}
```

Khai báo x, y là biến toàn cục



x=10, y=5

TRUYỀN GIÁ TRỊ CHO HÀM

□ Phạm vi của biến trong và ngoài hàm:

- .NET cấp phát cho mỗi hàm một vùng nhớ có kích thước cố định;
- Các biến được tạo ra trong hàm nào có phạm vi hoạt động trong hàm đó;
- Khi hết phạm vi của hàm thì biến đó bị huỷ (*không sử dụng được*);
- Hai hàm khác nhau, không thể sử dụng biến lẫn nhau;

TRUYỀN GIÁ TRỊ CHO HÀM

- ❑ Hàm chồng (Operator overloading) là những hàm **trùng tên nhau**, nhưng **khác nhau về tham số** (số lượng, kiểu,...)

```
static void Nhap(out int a)
{
    Console.Write("Nhap so nguyen: ");
    a = int.Parse(Console.ReadLine());
}
```

```
static void Nhap(out float a)
{
    Console.Write("Nhap so thuc: ");
    a = float.Parse(Console.ReadLine());
}
```

```
static void Nhap(out int a, out float b)
{
    Console.Write("Nhap so nguyen: ");
    a = int.Parse(Console.ReadLine());
    Console.Write("Nhap so thuc: ");
    b = float.Parse(Console.ReadLine());
}
```

BÀI TẬP CHƯƠNG 1

Bài 1. Cài đặt Visual Studio Code

Bài 2. Cài đặt trình C# trong VSCode

Bài 3. Viết chương trình thực hiện các yêu cầu sau:

- Nhập vào từ bàn phím hai số nguyên **a** và **b**
- Tính tổng bình phương của **a** và **b**

a. Hàm **TBP(a,b)**: kết quả được in lên màn hình

b. Hàm **TBP(a,b,kq)**: tham số kq sẽ mang kết quả

c. Hàm **TBP(a,b)**: kết quả được trả về qua tên hàm.