**ใบงานที่ 5**

**การควบคุมลำดับการทำงานของโปรแกรม**

**การเปลี่ยนทิศทางการทำงานของโปรแกรม**

**1.1 การเปลี่ยนทิศทางแบบไม่มีเงื่อนไข**

การเปลี่ยนทิศทางแบบไม่มีเงื่อนไขในภาษา C# มีประโยชน์ตรงที่สามารถเปลี่ยนทิศทางได้ทันต่อความต้องการ เนื่องจากในโปรแกรมบางประเภทต้องมีการตอบสนองแบบทันทีทันใด เช่นในกรณีที่เกิด exception ต่างๆ หรือต้องการออกนอก loop ต่างๆ โดยไม่สามารถใช้วิธีการตามปกติได้ การเปลี่ยนทิศทางการทำงานของโปรแกรมแบบไม่มีเงื่อนไข มีหลายคำสั่ง เช่น goto, try…catch, throw, break, continue และ return

**1.1.1 คำสั่ง goto**

คำสั่ง goto ใช้เพื่อกระโดดไปทำงานยังตำแหน่งที่ต้องการในทันที โดยไม่มีเงื่อนไขใดๆ ทั้งนั้น คำสั่ง goto มักจะใช้ร่วมกับคำสั่ง switch-case เนื่องจากมี label สำหรับแต่ละ case กำหนดไว้เรียบร้อยแล้ว นอกจากนี้คำสั่ง goto ยังสามารถใช้ในการกระโดดออกจาก loop ที่ลึกหรือซับซ้อนได้อีกด้วย

การใช้คำสั่ง goto ในการกระโดดแบบไม่มีเงื่อนไข

using System;

public class GotoTest

{

static void Main(string[] args)

{

Console.WriteLine("Line 1");

Console.WriteLine("Line 2");

Console.WriteLine("Line 3");

line4:

Console.WriteLine("Line 4");

Console.WriteLine("Line 5");

Console.WriteLine("Line 6");

goto line10;

Console.WriteLine("Line 7");

Console.WriteLine("Line 8");

Console.WriteLine("Line 9");

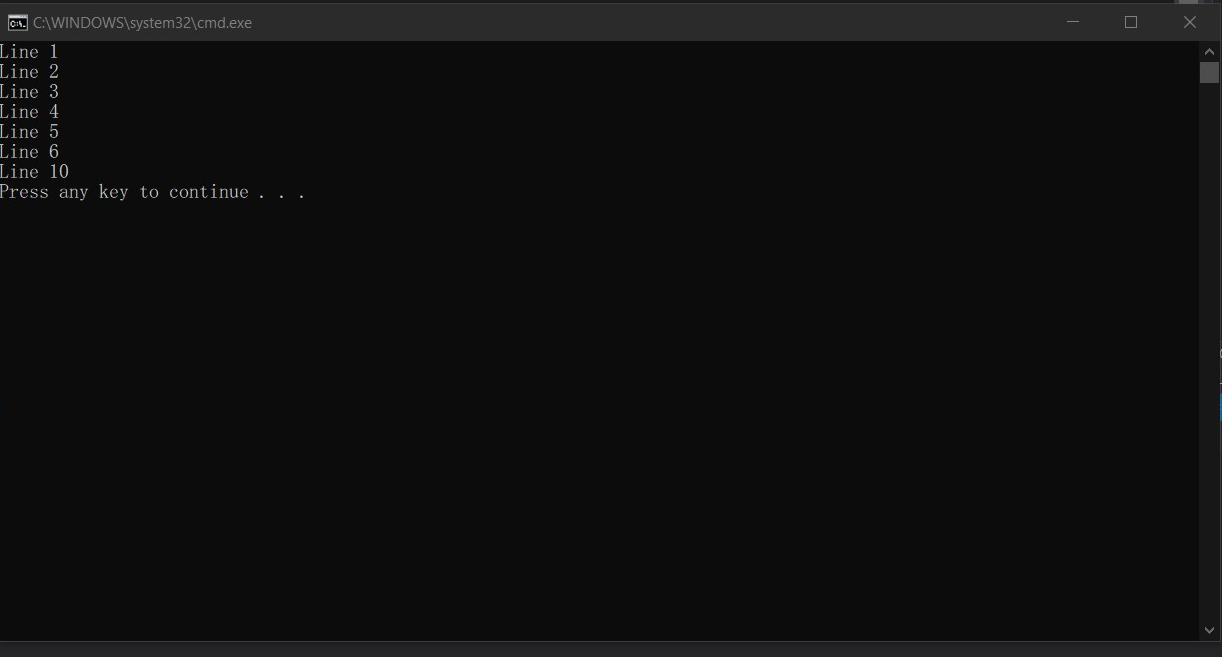
line10:

Console.WriteLine("Line 10");

}

}

➢ รันโปรแกรมและบันทึกผล



👷 การทดลอง ให้นักศึกษา แก้ไขดัดแปลงโปรแกรม โดยใช้คำสั่ง goto และให้มีเอาต์พุตดังนี้

Line 1 Line 4 Line 5 Line 2 Line 9

การใช้คำสั่ง goto ร่วมกับคำสั่ง switch-case

using System;

namespace ConsoleApp2

{

class CoffeeShop

{

static void Main()

{

Console.WriteLine("Coffee sizes: 1=Small 2=Medium 3=Large");

Console.Write("Please enter your selection: ");

string s = Console.ReadLine();

int n = int.Parse(s);

int cost = 0;

switch (n)

{

case 1:

cost += 25;

break;

case 2:

cost += 25;

goto case 1;

case 3:

cost += 50;

goto case 1;

default:

Console.WriteLine("Invalid selection.");

break;

}

if (cost != 0)

{

Console.WriteLine("Please insert {0} Bath.", cost);

}

Console.WriteLine("Thank you for your business.");

// Keep the console open in debug mode.

Console.WriteLine("Press any key to exit.");

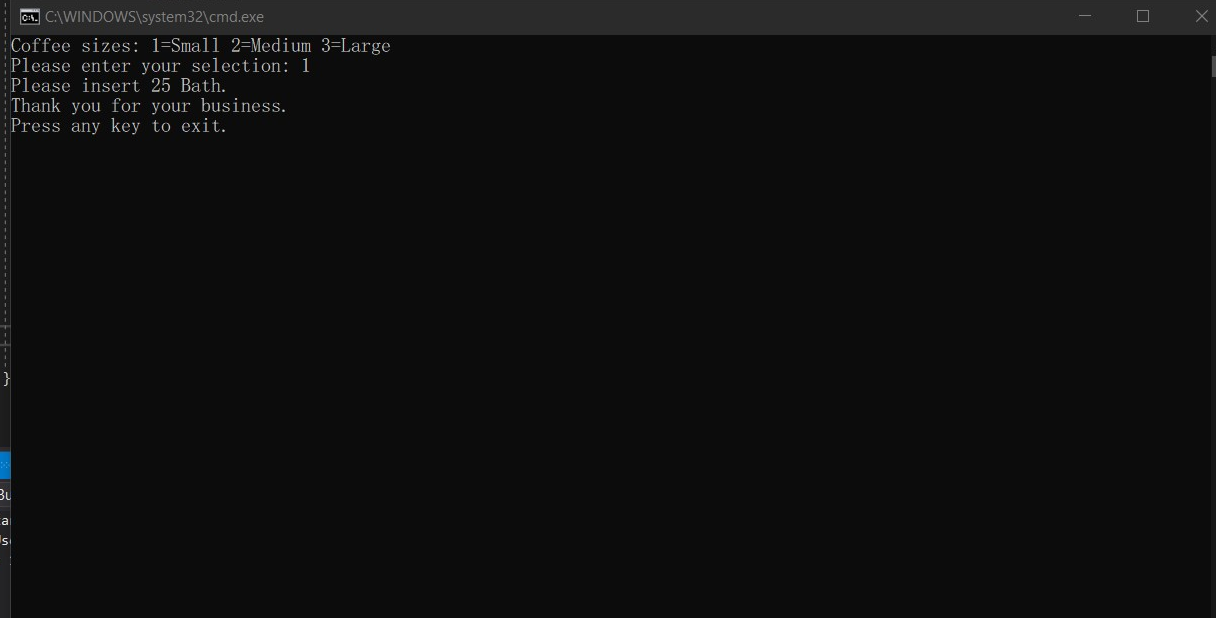
Console.ReadKey();

}

}

}

➢ รันโปรแกรมและบันทึกผล



การใช้คำสั่ง goto ร่วมกับคำสั่ง loop ที่ซับซ้อน

using System;

namespace ConsoleApp2

{

public class GotoWithLoop

{

static void Main()

{

int x = 200, y = 4;

int count = 0;

string[,] array = new string[x, y];

// Initialize the array:

for (int i = 0; i < x; i++)

for (int j = 0; j < y; j++)

array[i, j] = (++count).ToString();

// Read input:

Console.Write("Enter the number to search for: ");

// Input a string:

string myNumber = Console.ReadLine();

// Search:

for (int i = 0; i < x; i++)

{

for (int j = 0; j < y; j++)

{

if (array[i, j].Equals(myNumber))

{

goto Found;

}

}

}

Console.WriteLine("The number {0} was not found.", myNumber);

goto Finish;

Found:

Console.WriteLine("The number {0} is found.", myNumber);

Finish:

Console.WriteLine("End of search.");

// Keep the console open in debug mode.

Console.WriteLine("Press any key to exit.");

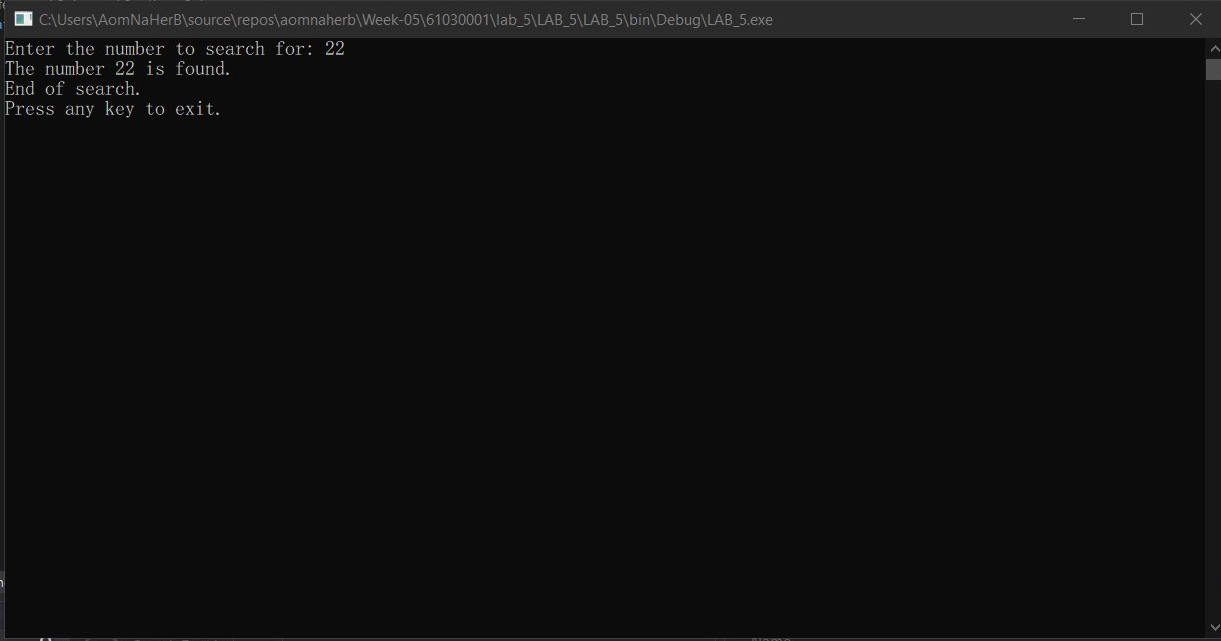
Console.ReadKey();

}

}

}

➢ รันโปรแกรมและบันทึกผล



**1.1.2. try…catch…finally**

ประโยค try…catch…finally ใช้สำหรับการดักจับและจัดการข้อผิดพลาดของโปรแกรม ทั้งขณะทำงาน (Run Time Process) หรือในขณะเริ่มต้นทำงาน (Init Process) โดยเราจะวางคำสั่งที่คาดการว่าจะเกิดข้อผิดพลาดขึ้นไว้ในบล็อกของ Try และวางส่วนจัดการข้อผิดพลาดไว้ในบล็อกของ catch และถ้ามีการดำเนินการใดๆ ที่ต้องทำทั้งในกรณีที่มีและไม่มีข้อผิดพลาด ก็จะใส่ไว้ในบล็อกของ Finally ในคำสั่งนี้สามารถเขียนบล็อกของ catch ได้หลายบล็อก คำสั่งนี้มีประโยชน์มากในการทำงานกับระบบที่เต็มไปด้วยการทำงานที่ไม่แน่นอน เช่น ระบอินเตอร์เน็ต หรือการใช้งานอุปกรณ์เชื่อมต่อ เช่น Printer หรือ External drive ในกรณีที่การเชื่อมต่อไม่เสถียร หรือไม่สามารถเขียน-อ่านไฟล์ได้ คำสั่งนี้จะช่วยป้องกันการค้างของโปรแกรมของเราขณะเรียกข้อมูลจาก network printer หรือ external drive ที่ถูกถอดออกไปจากระบบได้

โปรแกรมที่ไม่ได้ใช้คำสั่ง try…catch…finally

using System;

public class TryCatch

{

static void Main(string[] args)

{

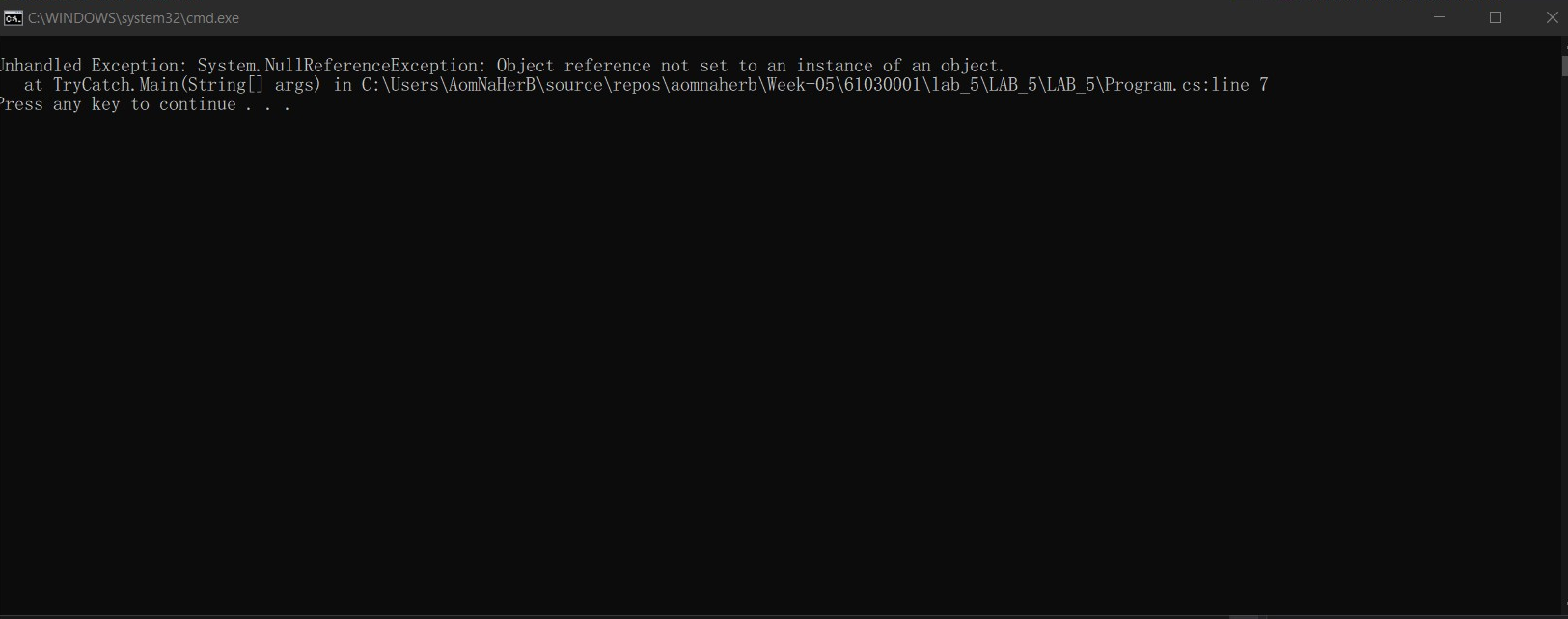
object o2 = null;

int i2 = (int)o2; // Error

}

}

➢ รันโปรแกรมและบันทึกผล



เนื่องจากมีการส่งค่าที่เป็น null ให้กับตัวแปร i2 จึงเกิด error ดังกล่าวขึ้นในขณะ run time ซึ่งจะเห็นว่า เราสามารถรันโปรแกรมนี้ได้ เนื่องจาก compiler ตรวจไม่พบข้อผิดพลาดในขณะคอมไพล์ วิธีการดักจับและแก้ไขข้อผิดพลาด ทำได้โดยการใช้คำสั่ง try…catch กับส่วนของโปรแกรมที่คาดว่าอาจจะเกิดข้อผิดพลาด ซึ่งจากโปรแกรมข้างบนนั้น compiler ฟ้องว่ามีข้อผิดพลาดเกิดขึ้นในบรรทัดที่ 7 ดังนั้นสามารถแก้ไขโปรแกรมได้เป็นดังนี้

โปรแกรมที่ใช้คำสั่ง try…catch

using System;

public class TryCatch

{

static void Main(string[] args)

{

object o2 = null;

try

{

int i2 = (int)o2;

Console.WriteLine("i2 = {0}", i2);

}

catch

{

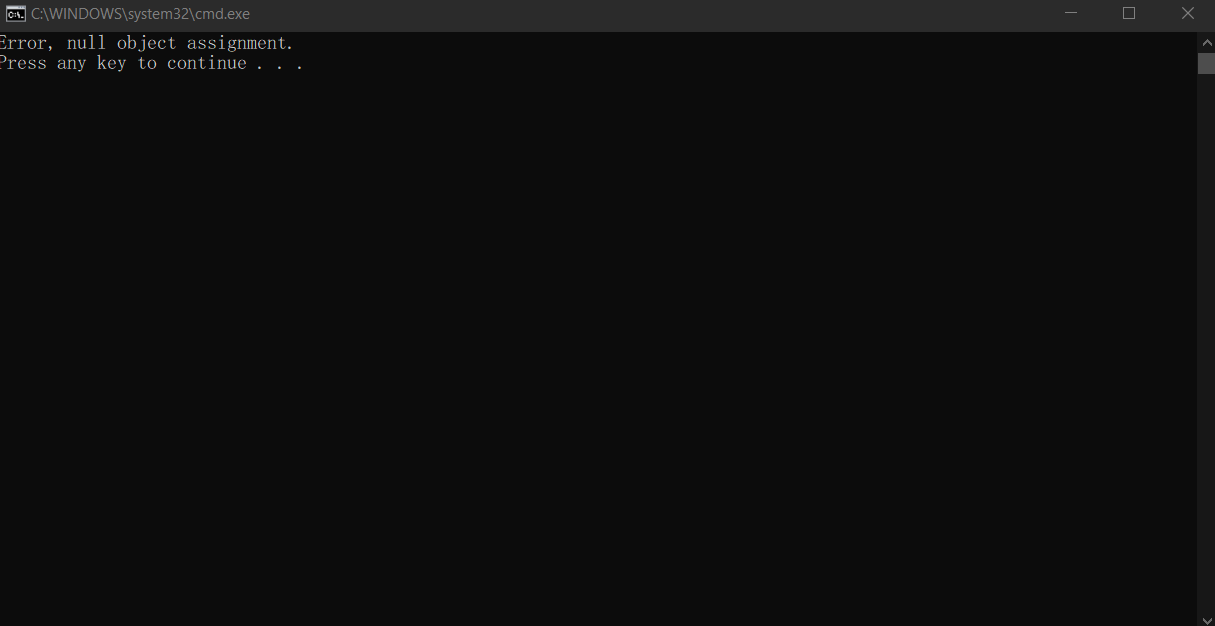
Console.WriteLine("Error, null object assignment.");

}

}

}

➢ รันโปรแกรมและบันทึกผล



ในคำสั่ง catch นั้น เราสามารถใส่ parameter ซึ่งเป็นประเภทของข้อผิดพลาดได้ด้วย เช่น NullReferenceException เพื่อดักจับการส่งค่า 1 ให้ตัวแปร หรือ DivideByZeroException ไว้คอยดักจับ ในกรณีที่มีการหารด้วยค่าศูนย์ เป็นต้น โดยมีรูปแบบการใช้งานดังตัวอย่าง ตัวอย่าง การดักจับข้อผิดพลาดหลายๆ รูปแบบ

การดักจับข้อผิดพลาดหลาย ๆ รูปแบบ

using System;

public class TryCatch

{

static void Main(string[] args)

{

int a = 0;

try

{

Console.WriteLine(100/a);

}

catch(NullReferenceException e)

{

Console.WriteLine(e.Message);

}

catch (DivideByZeroException e)

{

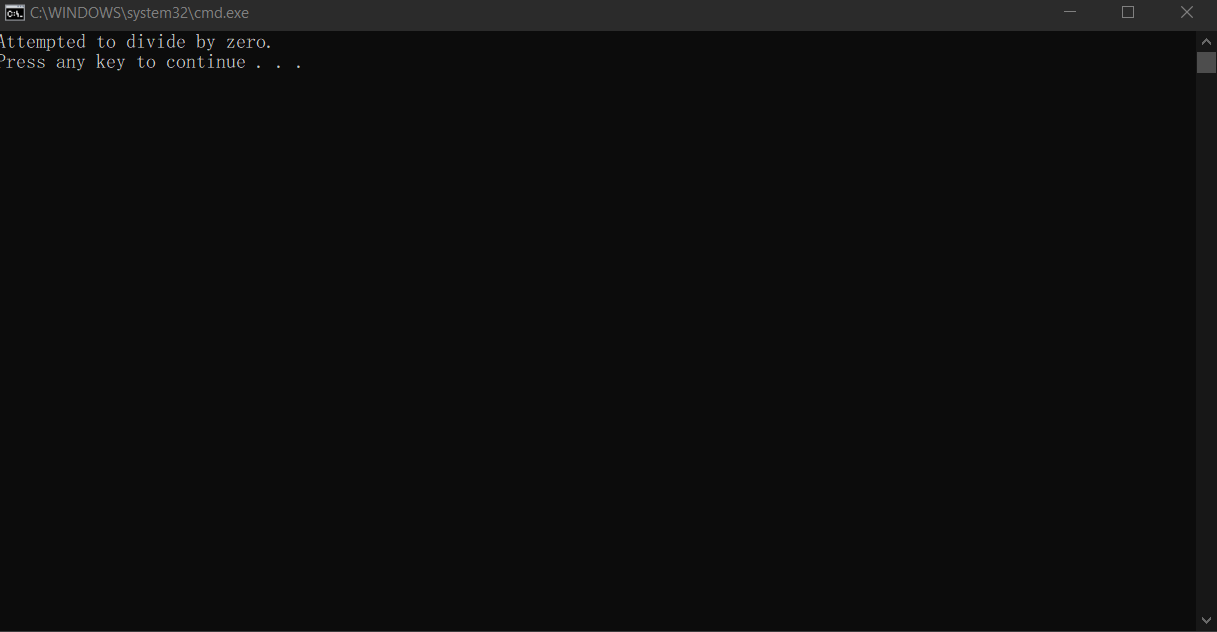
Console.WriteLine(e.Message);

}

}

}

➢ รันโปรแกรมและบันทึกผล



**การทดลอง การดักจับข้อผิดพลาดในรูปแบบต่างๆ**

**คำสั่ง**

ให้นักศึกษาเขียนโปรแกรมเพื่อทดสอบว่าโปรแกรมต่อไปนี้มีความผิดพลาดในการทำงานหรือไม่ ถ้ามี ให้นักศึกษาเขียนคำสั่ง try…catch เพิ่มเข้าไป เพื่อให้โปรแกรมรันได้โดยไม่ค้าง (เลือกประเภทของ exception จาก reference ท้ายใบงาน)

**ข้อ 1**

using System;

public class TryCatch

{

static void Main(string[] args)

{

int a = int.MaxValue;

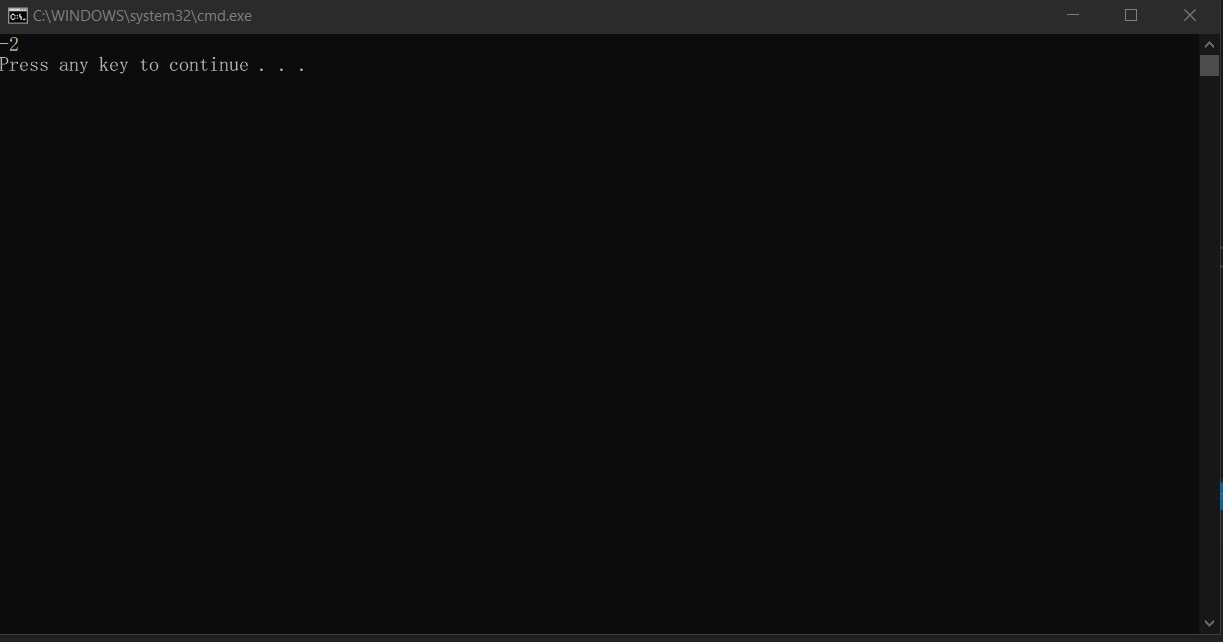
a \*= 2;

Console.WriteLine(a);

}

}

➢ รันโปรแกรมและบันทึกผล



**ข้อ 2**

using System;

public class TryCatch

{

static void Main(string[] args)

{

int a = 0;

int b = 10;

b /= a;

Console.WriteLine(a);

}

}

➢ รันโปรแกรมและบันทึกผล



**ข้อ 3**

using System;

public class TryCatch

{

static void Main(string[] args)

{

int value = 800000000;

checked // check for overflow

{

int square = value \* value;

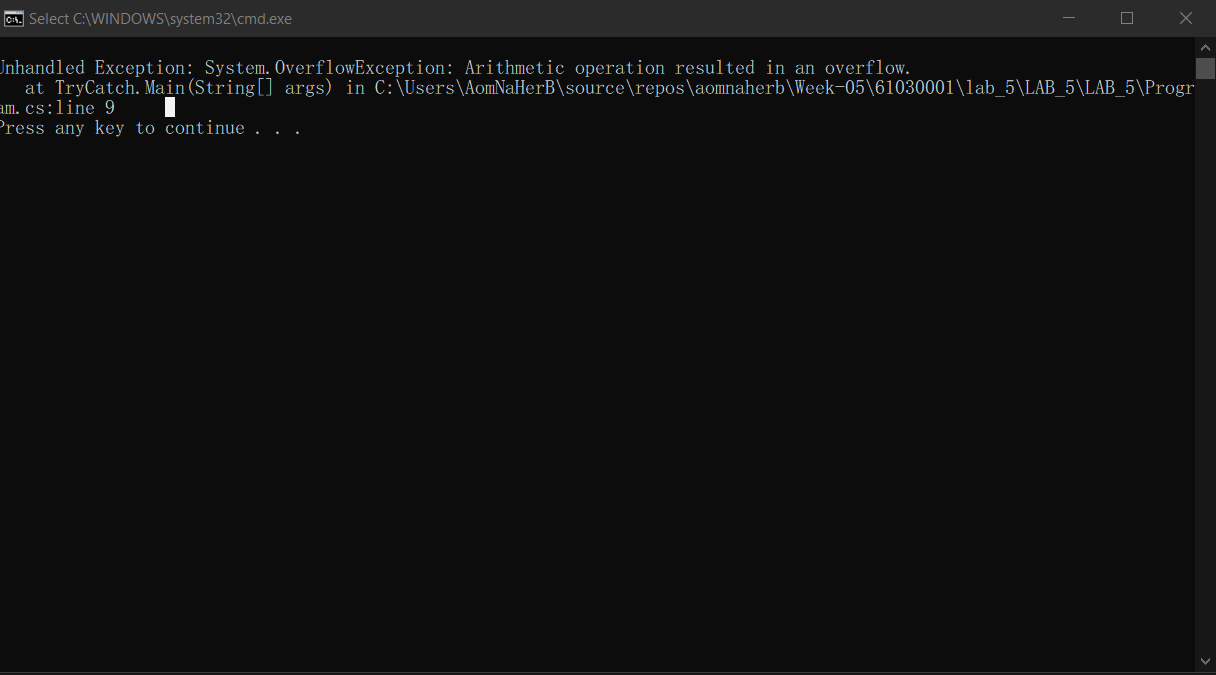
Console.WriteLine("{0} ^ 2 = {1}", value, square);

}

}

}

➢ รันโปรแกรมและบันทึกผล



**1.1.3. คำสั่ง throw**

คำสั่ง throw ใช้เพื่อเปลี่ยนเส้นทางการทำงานของโปรแกรมโดยเจาะจง exception เป้าหมาย ตัวอย่าง การดักจับข้อผิดพลาดหลายๆ รูปแบบ

**การทดลอง การใช้คำสั่ง throw**

using System;

using System.IO;

public class ExceptionLearning

{

public static void Main()

{

int a = 10;

int b = 20;

int c = add(a, b);

}

private static int add(int a, int b)

{

throw new NotImplementedException();

}

}

➢ รันโปรแกรมและบันทึกผล



**👷 การทดลอง ชนิดของ exception**

ให้เปลี่ยนชนิดของการ throw exception ในบรรทัดที่ 34 เป็น exception ดังต่อไปนี้ แล้วอธิบายผลที่ได้

1. DivideByZeroException
2. NullReferenceException
3. FileNotFoundException
4. FormatException

using System;

using System.IO;

public class ExceptionLearning

{

public static void Main()

{

int a = 10;

int b = 20;

int c ;

try

{

c = div(a, b);

}

catch (DivideByZeroException e)

{

Console.WriteLine("DivideByZeroException");

Console.WriteLine(e.Message);

}

catch (NullReferenceException e)

{

Console.WriteLine("NullReferenceException");

Console.WriteLine(e.Message);

}

catch (Exception e)

{

Console.WriteLine("Exception");

Console.WriteLine(e.Message);

}

}

private static int div(int a, int b)

{

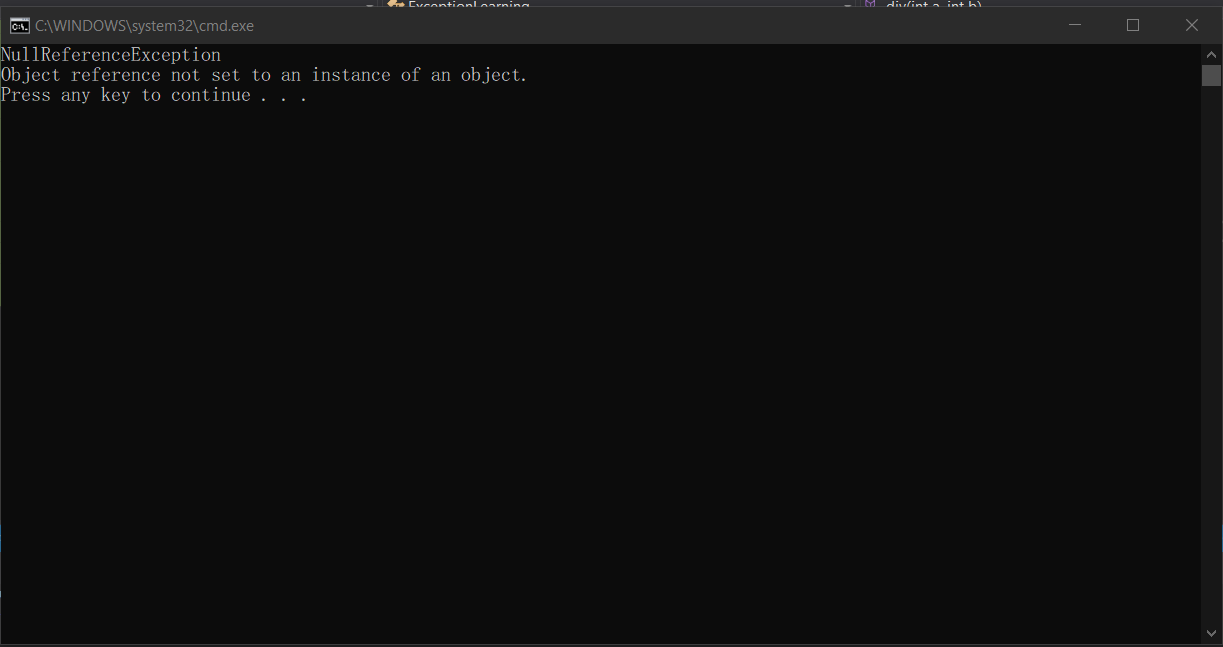
throw new <ชื่อเมธอดในข้อ 1 - 4 ครั้งละข้อ> ();

}

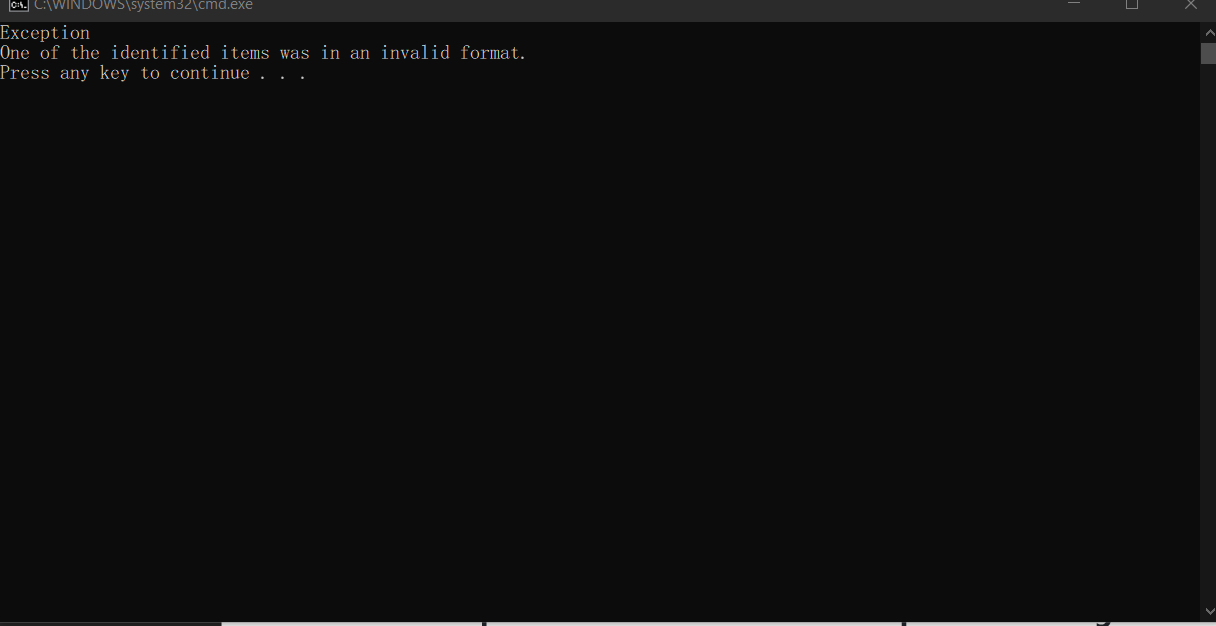
}

➢ รันโปรแกรมและบันทึกผล









**เรื่องของ exception นี้จะทำการทดลองเพิ่มเติมในเรื่อง Exception Handling**

**1.2 การเปลี่ยนทิศทางแบบมีเงื่อนไข (Conditional Branching)**

**1.2.1 คำสั่ง if**

คำสั่ง if เป็นคำสั่งที่ใช้เปลี่ยนทิศทางการทำงานของโปรแกรมตามเงื่อนไข หรืออาจจะเรียกได้ว่าเป็นคำสั่ง สำหรับเลือก เส้นทาง (Selection statements) โดยค่าที่นำมาเป็นเงื่อนไขในการตัดสินใจ จะต้องมีชนิดเป็น boolean (ซึ่งมีค่าเป็น true หรือ false) เท่านั้น

**รูปแบบของคำสั่ง if**

**1. แบบมีคำสั่งเดียว**

if ( condition )

statement ;

**2.แบบมีหลายคำสั่ง (เป็นบล็อก)**

if ( condition )

{ // begin of block

statement\_1 ;

statement\_2 ;

...

statement\_3 ;

} // end of block

**การทดลอง การใช้คำสั่ง if**

using System;

using System.IO;

public class IfLearning

{

public static void Main()

{

int a = 2;

if (a == 2)

{

Console.WriteLine("execute this line");

}

if (a < 2)

{

Console.WriteLine("execute this line too");

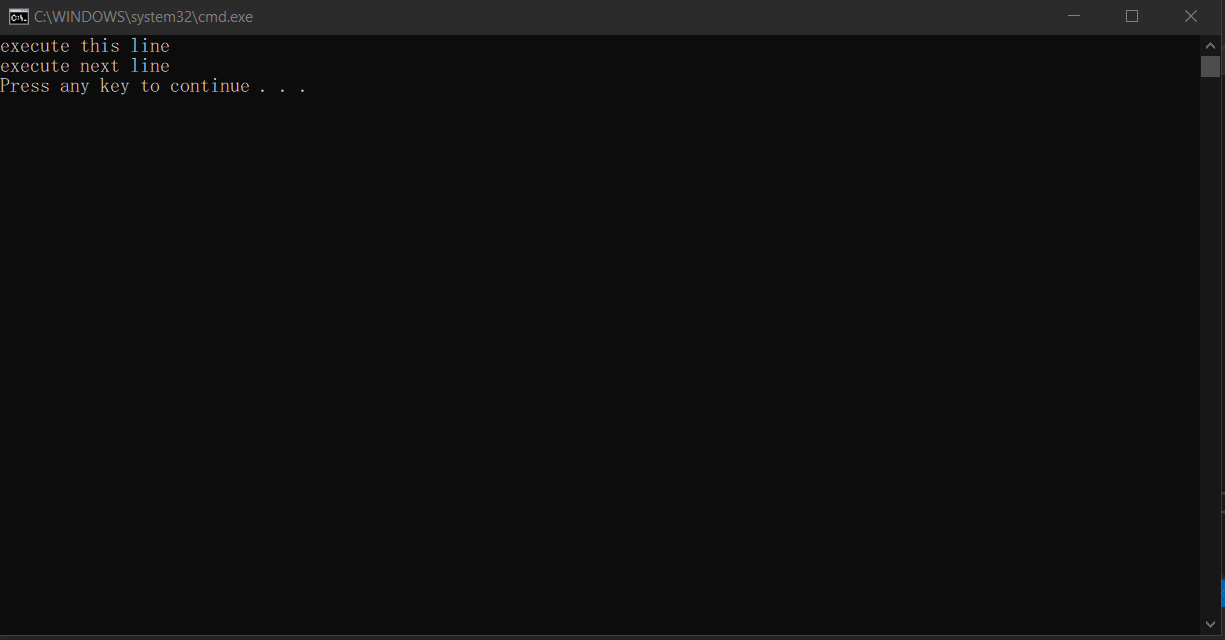
}

Console.WriteLine("execute next line");

}

}

➢ รันโปรแกรมและบันทึกผล



ในบล็อกของคำสั่ง if นั้น statements ที่ถูกเรียกทำงานคือ statements ที่มีเงื่อนไขเป็น true เท่านั้น statements ที่เงื่อนไขของ if มีค่าเป็น false จะไม่ถูกเรียกทำงาน โดยการตัดสินใจจะเป็นอิสะต่อกัน คือคำสั่ง if บล็อกหลังไม่ได้รับผลกระทบใดๆ จาก if ในบล็อกแรก เช่นเดียวกับบรรทัดที่ 16 ของโปรแกรม ซึ่งไม่มีคำสั่งใดๆ มาควบคุมลำดับการทำงาน มันจึงถูกเรียกทำงานตามปกติ

**1.2.2 คำสั่ง if…else**

เงื่อนไขที่เป็นไปได้ของคำสั่งในการตัดสินใจมีสองทางเสมอ (true และ false) ที่ผ่านมา เราจะเห็นว่า คำสั่ง if เป็นคำสั่งที่เลือกทำเพียงทางเดียว (เฉพาะในกรณีที่เงื่อนไขเป็น true เท่านั้น) หากต้องการให้โปรแกรมทำงานทั้งกรณีที่เงื่อนไขเป็น true และ false เราต้องใช้คำสั่ง if…else โดยมีรูปแบบดังนี้ รูปแบบของคำสั่ง if…else

if (condition)

{

statement; // execute when condition = true

}

else

{

statement; // execute when condition = false

}

**การทดลอง การใช้งานคำสั่ง if…else**

using System;

using System.IO;

public class IfLearning

{

public static void Main()

{

int a = 2;

if (a == 2)

{

Console.WriteLine("execute this line");

}

else

{

Console.WriteLine("execute another line too");

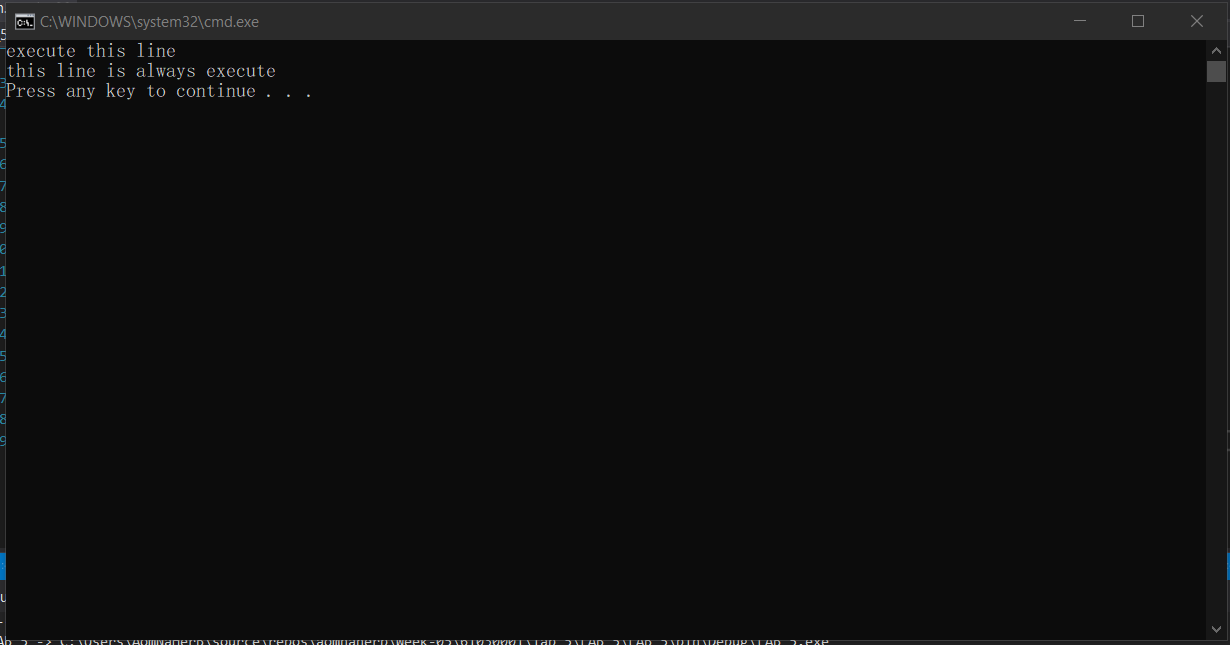
}

Console.WriteLine("this line is always execute");

}

}

➢ รันโปรแกรมและบันทึกผล



**1.2.3 คำสั่ง if ซ้อนกัน (nested if)**

คำสั่ง if สามารถเขียนซ้อนกันเป็นชั้นได้ เรียกว่า nested if มีรูปแบบดังนี้

if (condition)

{

if (condition) // nested if

{

...;

}

}

**การทดลอง การใช้งานคำสั่ง nested if**

using System;

using System.IO;

public class IfLearning

{

public static void Main()

{

int a = 10;

int b = 20;

if (a == 10)

{

if (b == 20)

{

Console.WriteLine("a = 10 and b = 20");

}

if (b != 20)

{

Console.WriteLine("a = 10 and b != 20");

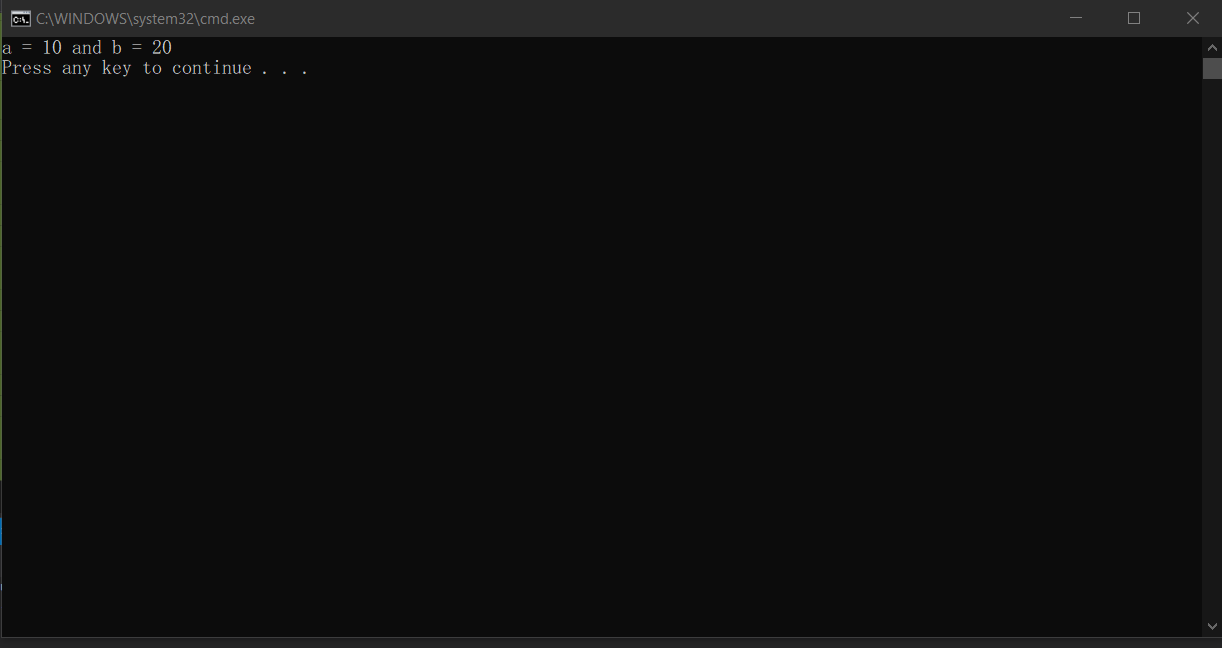
}

}

}

}

➢ รันโปรแกรมและบันทึกผล



ข้อสังเกตุ คำสั่ง nested if เปรียบเทียบได้กับการ AND กันของเงื่อนไขในระดับต่างๆ

**1.2.4 คำสั่ง if…else…if**

ในบางกรณีที่มีการตัดสินใจในหลายทางเลือก เราอาจใช้คำสั่ง if…else…if เรียงต่อกันไปเรื่อยๆ ตัวอย่างเช่นโปรแกรมการตัดเกรดที่มีหลายระดับ เป็นต้น

การทดลอง การใช้งานคำสั่ง if…else...if

using System;

using System.IO;

public class IfLearning

{

public static void Main()

{

int point = 68; // ทดลองเปลี่ยนเป็น Console.ReadLine() เพื่อรับค่าจากผู้ใช้

if (point < 50)

Console.WriteLine("Grade F");

else if (point < 60)

Console.WriteLine("Grade D");

else if (point < 70)

Console.WriteLine("Grade C");

else if (point < 80)

Console.WriteLine("Grade B");

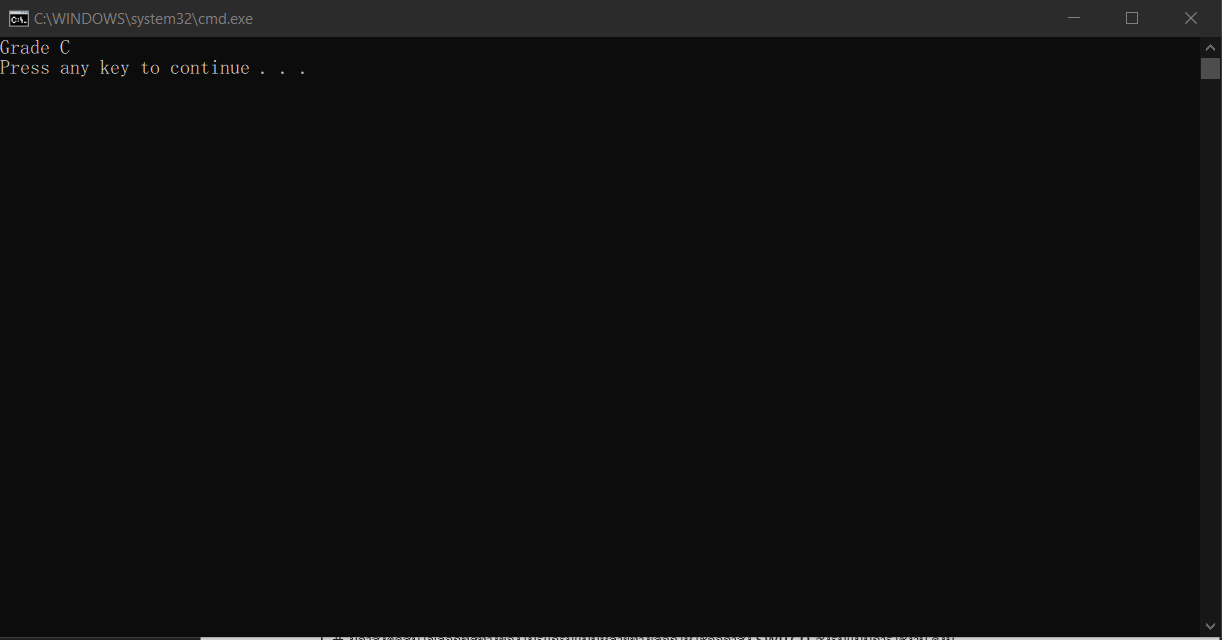
else

Console.WriteLine("Grade A");

}

}

➢ รันโปรแกรมและบันทึกผล



**1.2.5 คำสั่ง switch**

ในกรณีที่มีทางเลือกในการตัดสินใจเป็นจำนวนมาก ไม่เป็นการสะดวกที่จะเขียนเป็นโปรแกรมยาวๆ เช่นในกรณีของคำสั่ง if…else…if ภาษา C# มีคำสั่งตัดสินใจเลือกทิศทางของโปรแกรมแบบหลายทางเลือกให้ใช้คือคำสั่ง switch ซึ่งรูปแบบการใช้งาน ดังนี้

switch(<expression>) {

case <value> : <statement>

case <value> : <statement>

case <value> : <statement>

..........................

[default : <statement>]

}

ในภาษา C# นั้น ยอมให้นิพจน์ (constant-expression) เป็นแบบจํานวนเต็ม (integer) แบบอักขระ (char) และ แบบข้อความ (string)

การทดลอง การใช้งานคำสั่ง switch

using System;

using System.IO;

public class switchLearning

{

public static void Main()

{

Console.Write("Input your grade (A, B, C, D or F) : ");

string gradeString = Console.ReadLine();

string message;

switch (gradeString.ToUpper())

{

case "A":

message = "Excellent";

break;

case "B":

message = "Good";

break;

case "C":

message = "Cool";

break;

case "D":

message = "Try";

break;

case "F":

message = "Get out!!";

break;

default:

message = "Incorrect grade";

break;

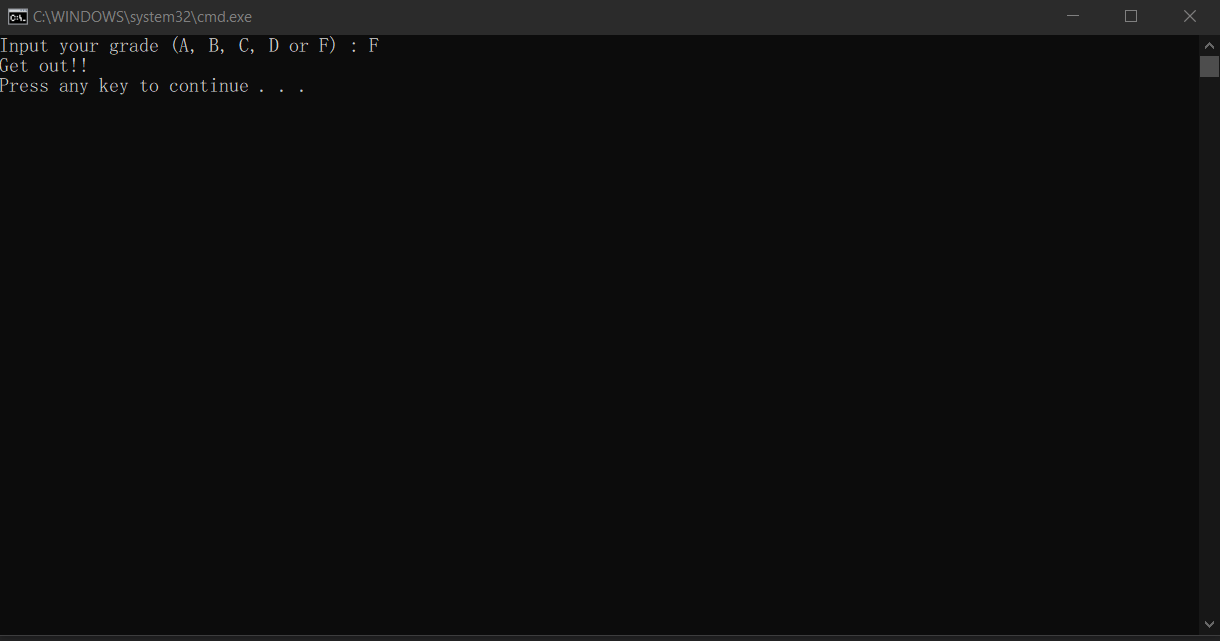
}

Console.WriteLine(message);

}

}

➢ รันโปรแกรมและบันทึกผล



**แบบฝึกหัด เรื่องคำสั่ง switch ให้เขียนโปรแกรมรับค่าชื่อของวัน แล้วพิมพ์ข้อความออกทางหน้าจอ ดังตัวอย่าง**

Input day name : sun

sun is Sunday, color Red

ตารางกำหนดชื่อและสีประจำวัน

| **Input ที่รับได้** | **ชื่อวัน** | **สี** |
| --- | --- | --- |
| sun | Sunday | Red |
| mon | Monday | Yellow |
| tue | Tuesday | Pink |
| wed | Wednesday | Green |
| thu | Thursday | Orange |
| fri | Friday | Blue |
| sat | Saturday | Purple |

**คำสั่งควบคุมการวนรอบ (Iteration statement)**

**คำสั่ง While**

คำสั่ง while จะวนรอบทำคำสั่งภายในลูป while จนกระทั่งเงื่อนไขภายในวงเล็บของคำสั่ง while มีค่าเป็น false และเนื่องจากคำสั่ง while จะมีการกระทำคำสั่งแรกหลังจากการตัดสินใจตามเงื่อนไขในวงเล็บ คำสั่ง while จึงอาจจะไม่ทำคำสั่งใดๆ เลยก็ได้ หากเงื่อนไขในวงเล็บเป็น false ตั้งแต่แรก

การทดลอง การใช้งานคำสั่ง while

using System;

namespace ConsoleApp2

{

class WhileTest

{

static void Main()

{

int n = 1;

while (n < 6)

{

Console.WriteLine("Current value of n is {0}", n);

n++;

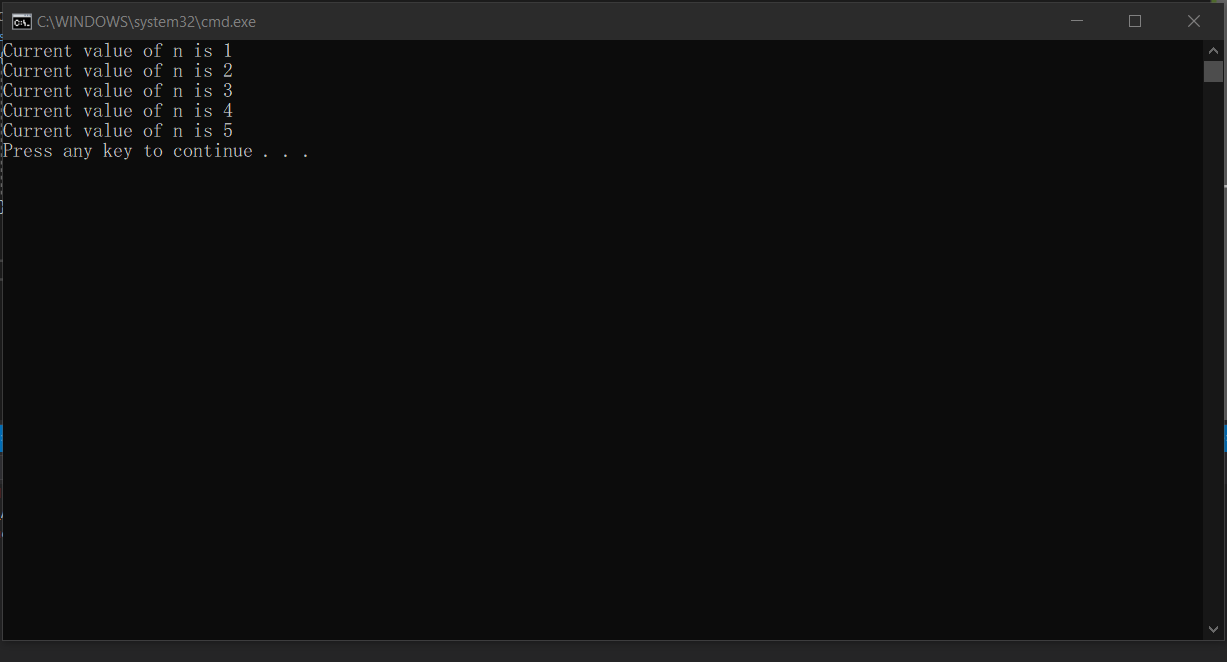
}

}

}

}

➢ รันโปรแกรมและบันทึกผล



**คำสั่ง do…while**

คำสั่ง do จะใช้งานร่วมกับคำสั่ง while รวมเป็นประโยค do...while ใช้เพื่อควบคุมให้มีการทำงานวนรอบจนกว่า เงื่อนไขในวงเล็บของคำสั่ง while จะเป็น false โดยทั่วไป เรามักจะพบเห็นบล็อกของคำสั่งในประโยค do...while ที่มีการล้อมรอบด้วยวงเล็บปีกกา { } แต่ในความเป็นจริง หากมีคำสั่งเดียว เราไม่จำเป็นต้องใส่วงเล็บปีกกาก็ได้

การทดลอง การใช้งานคำสั่ง do..while

using System;

namespace ConsoleApp2

{

class DoTest

{

static void Main()

{

int a = 0;

do a = 2;

while (false);

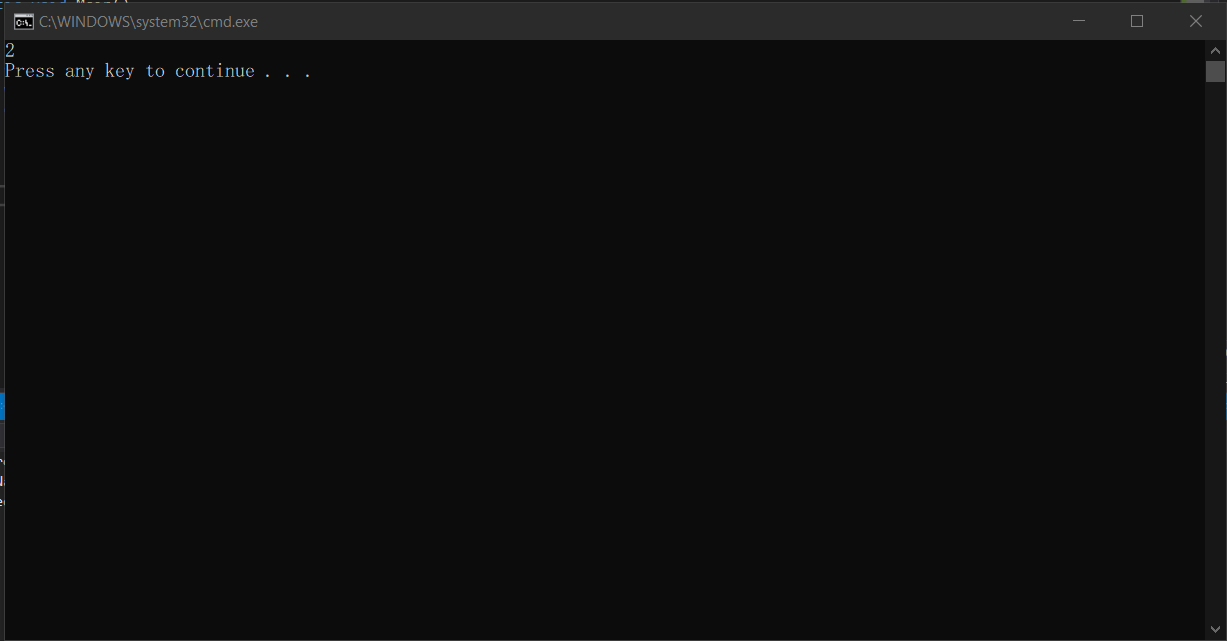
Console.WriteLine(a);

}

}

}

➢ รันโปรแกรมและบันทึกผล



คำสั่ง do จะต่างจากคำสั่ง while ตรงที่ คำสั่ง do จะมีการทำงานอย่างน้อย 1 คำสั่ง ก่อนที่จะมีการตัดสินใจว่าจะทำซ้ำหรือไม่ โดยการพิจารณาเงื่อนไนในประโยค while

การทดลอง การใช้งานคำสั่ง while เปรียบเทียบกับคำสั่ง do...while

using System;

namespace ConsoleApp2

{

class DoTest

{

static void Main()

{

Console.WriteLine("---- Begin of While statement.");

int n = 1;

while (n < 1)

{

Console.WriteLine(" Current value of n is {0}", n);

n++;

}

Console.WriteLine("---- End of While statement.");

Console.WriteLine("---- Begin of Do..While statement.");

n = 1;

do

{

Console.WriteLine(" Current value of n is {0}", n);

n++;

}

while (n < 1);

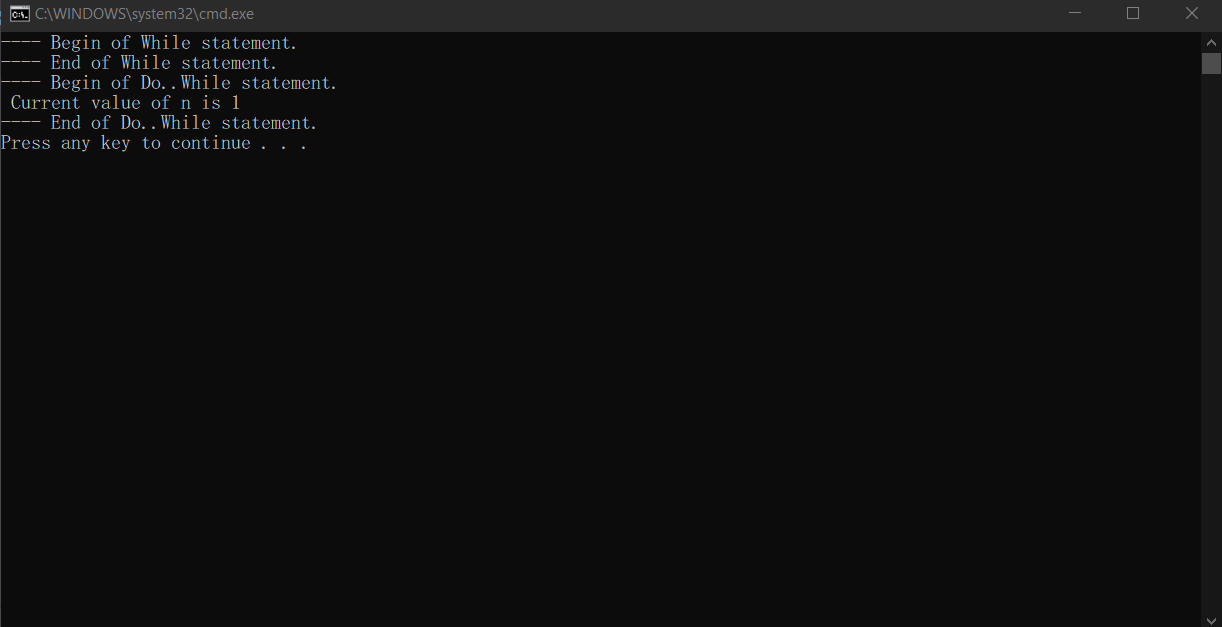
Console.WriteLine("---- End of Do..While statement.");

}

}

}

➢ รันโปรแกรมและบันทึกผล



**คำสั่ง for**

คำสั่ง for จะช่วยให้สามารถควบคุมการวนรอบของคำสั่งให้ทำงานซ้ำๆ ได้จนกว่าเงื่อนไขในการดำเนินการต่อไปจะเป็น false คำสั่งนี้มีประโยชน์ในการวนรอบทำงานกับข้อมูลใจอาเรย์ (iterating over arrays) หรือในงานที่ต้องการทราบจำนวนรอบ ที่ได้ทำไปแล้ว

รูปแบบของคำสั่ง for

for (initialization; conditional check ; update)

{

statements;

}

การทดลอง การใช้งานคำสั่ง for

using System;

namespace ConsoleApp2

{

class ForLoopTest

{

static void Main()

{

for (int i = 1; i <= 5; i++)

{

Console.WriteLine(i);

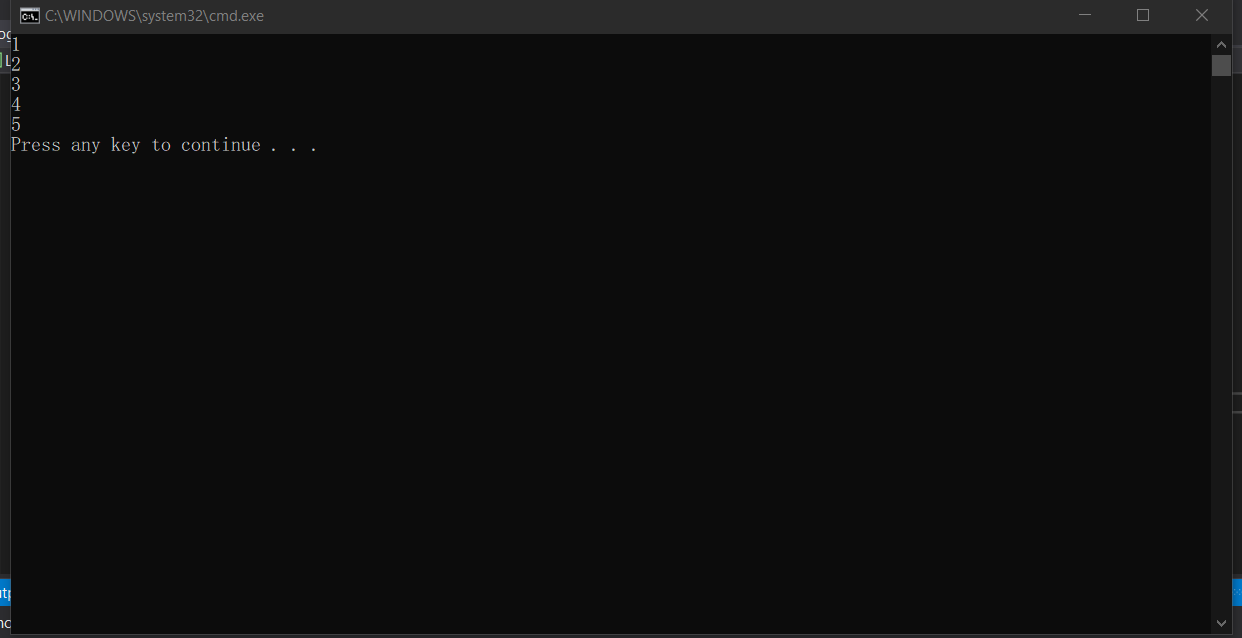
}

}

}

}

➢ รันโปรแกรมและบันทึกผล



การทดลอง การใช้งานคำสั่ง for สร้างสูตรคูณแม่ 2

using System;

namespace ConsoleApp2

{

class Multiply

{

static void Main()

{

for (int i = 1; i <= 12; i++)

{

Console.WriteLine("{0, 2} x 2 = {1, 2}",i,i\*2 );

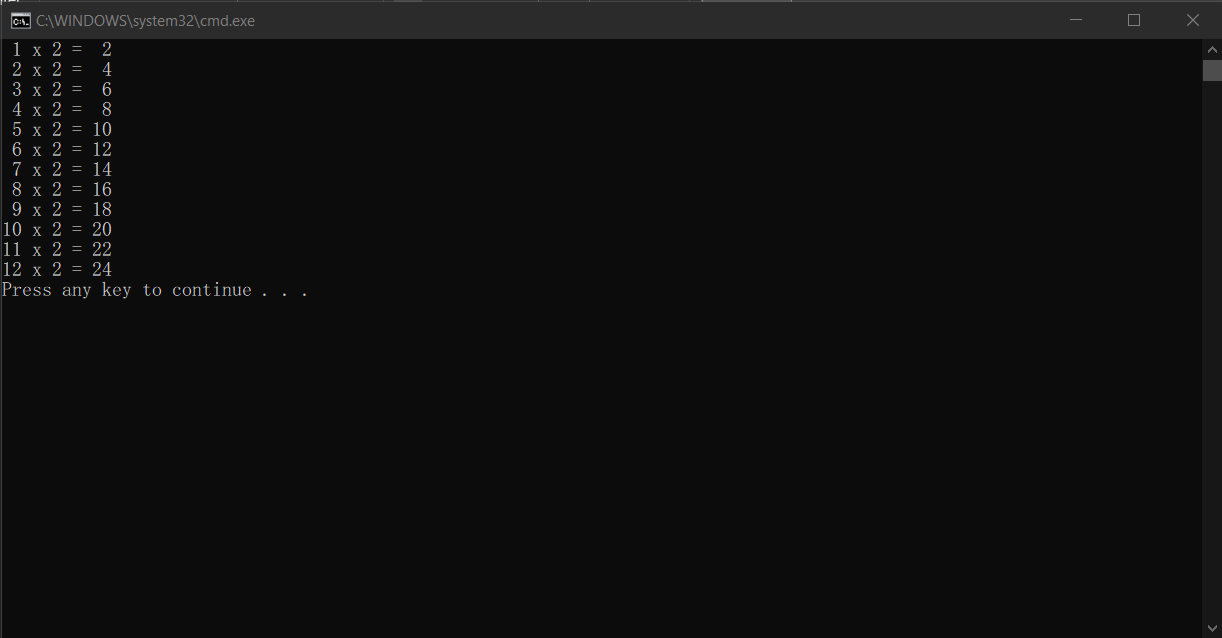
}

}

}

}

➢ รันโปรแกรมและบันทึกผล



การทดลอง การใช้งานคำสั่ง for สร้างสูตรคูณแม่ 2 ถึง 12

using System;

namespace ConsoleApp2

{

class Multiply

{

static void Main()

{

for (int i = 1; i <= 12; i++)

{

for (int j = 1; j <= 12; j++)

{

Console.Write ("{0, 4}", i \* j);

}

Console.WriteLine();

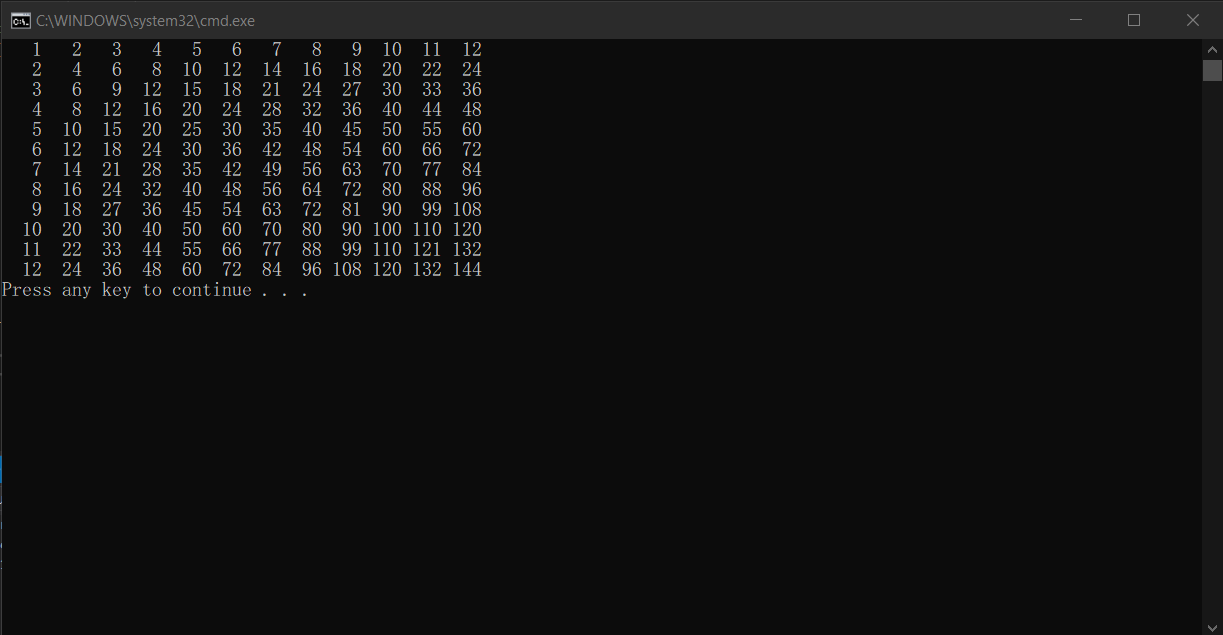
}

}

}

}

➢ รันโปรแกรมและบันทึกผล



**คำสั่ง break**

คำสั่ง break เป็นคำสั่งกระโดดแบบไม่มีเงื่อนไข ที่ใช้ร่วมกับคำสั่งวนรอบต่าง ๆ หรือคำสั่ง switch

การทดลอง การใช้งานคำสั่ง break ร่วมกับคำสั่งวนรอบ (for)

using System;

namespace ConsoleApp2

{

class BreakTest

{

static void Main()

{

for (int i = 1; i <= 100; i++)

{

if (i == 5)

{

break;

}

Console.WriteLine(i);

}

// Keep the console open in debug mode.

Console.WriteLine("Press any key to exit.");

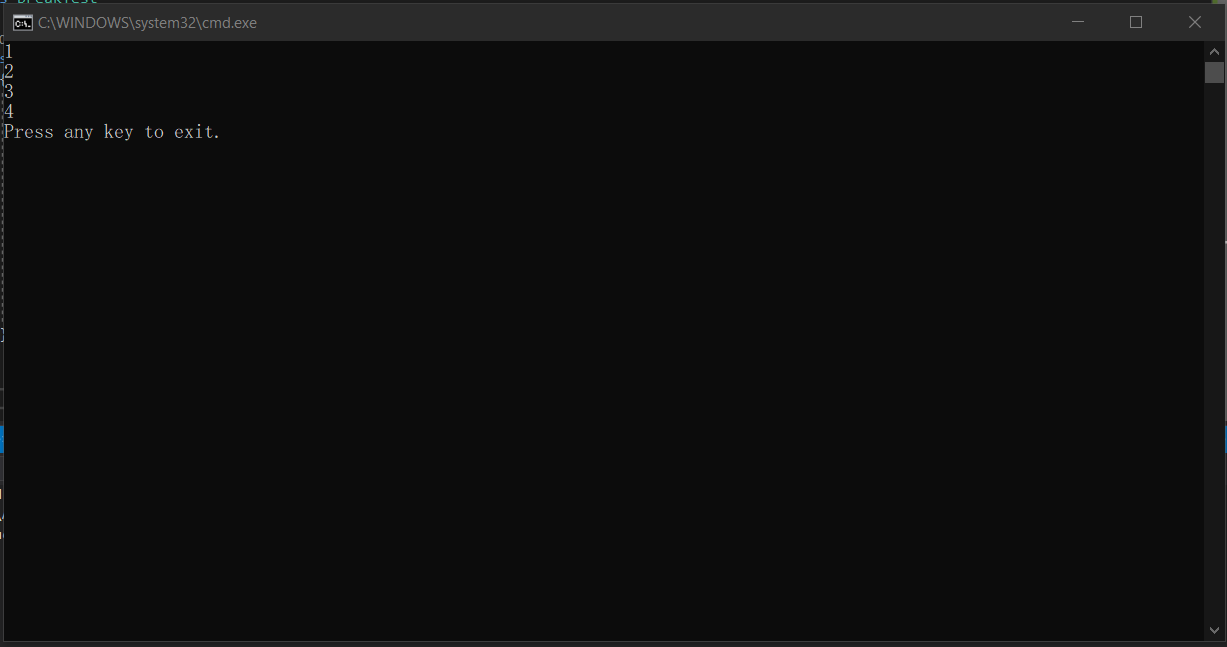
Console.ReadKey();

}

}

}

➢ รันโปรแกรมและบันทึกผล



การทดลอง การใช้งานคำสั่ง break ร่วมกับคำสั่งวนรอบ (for)

using System;

namespace ConsoleApp2

{

class BreakTest

{

static void Main()

{

for (int i = 1; i <= 100; i++)

{

if (i == 5)

{

break;

}

}

Console.WriteLine(i);

// Keep the console open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

}

}

➢ รันโปรแกรมและบันทึกผล

Error I

การทดลอง การใช้งานคำสั่ง break ร่วมกับคำสั่ง switch

using System;

namespace ConsoleApp2

{

class Switch

{

static void Main()

{

Console.Write("Enter your selection (1, 2, or 3): ");

string s = Console.ReadLine();

int n = Int32.Parse(s);

switch (n)

{

case 1:

Console.WriteLine("Current value is {0}", 1);

break;

case 2:

Console.WriteLine("Current value is {0}", 2);

break;

case 3:

Console.WriteLine("Current value is {0}", 3);

break;

default:

Console.WriteLine("Sorry, invalid selection.");

break;

}

// Keep the console open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

}

}

➢ รันโปรแกรมและบันทึกผล



**คำสั่ง continue**

คำสั่ง continue เป็นคำสั่งที่ใช้เพื่อข้าม statements ทั้งหมดที่ต่อท้ายเพื่อกลับไปเริ่มต้นรอบ (iteration) ใหม่ ในบล็อกของคำสั่ง while, do, for, หรือ foreach

**การทดลอง การใช้งานคำสั่ง continue**

using System;

namespace ConsoleApp2

{

class ContinueTest

{

static void Main()

{

for (int i = 1; i <= 10; i++)

{

if (i < 9)

{

continue;

}

Console.WriteLine(i);

}

// Keep the console open in debug mode.

Console.WriteLine("Press any key to exit.");

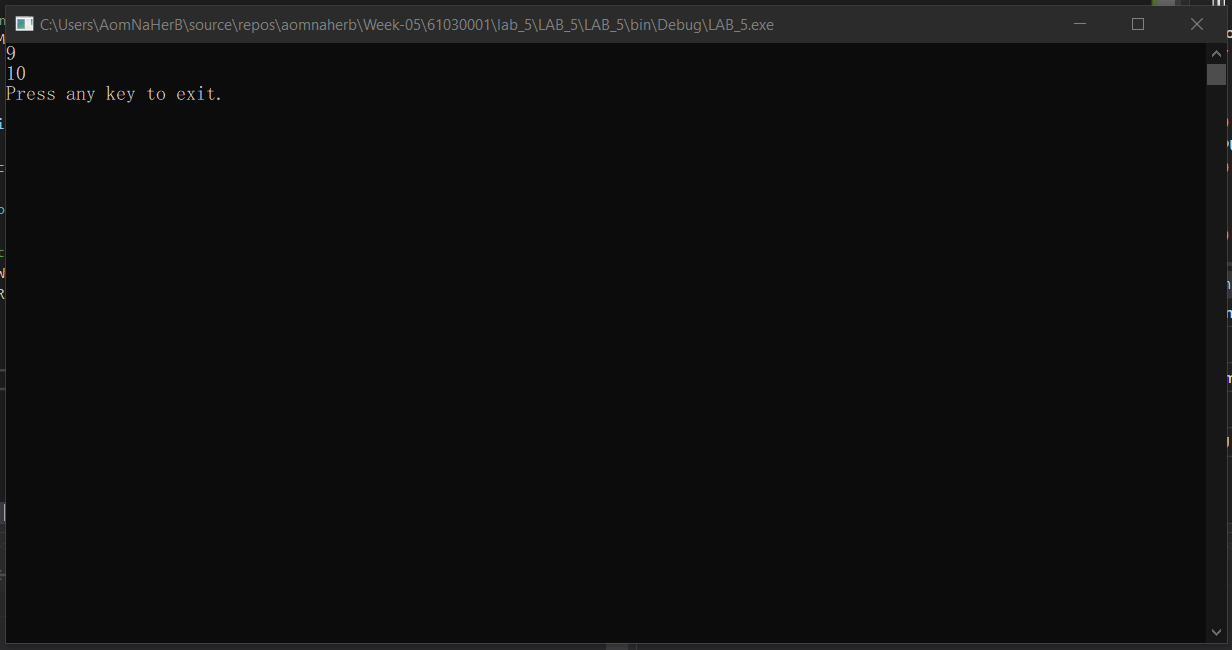
Console.ReadKey();

}

}

}

➢ รันโปรแกรมและบันทึกผล



**คำสั่ง foreach…in**

คำสั่ง foreach, in เป็นคำสั่งวนรอบที่สามารถดึง element ต่างๆ ที่ฝังอยู่ใน array หรือ collection ต่าง ๆ ออกมาครั้งละ 1 ตัว คำสั่ง foreach, in นี้จะทำงานคล้ายกับคำสั่ง for แต่มีข้อแตกต่างตรงที่คำสั่ง foreach, in มีจุดมุ่งหมายแค่เพียงการอ่านค่าออกมาใช้งาน หากต้องการเพิ่มหรือ update ค่าลงไปใน array หรือ collection ให้ใช้ for loop ตามปกติ

การทดลอง การใช้งานคำสั่ง foreach…in

using System;

namespace ConsoleApp2

{

class ForEachTest

{

static void Main(string[] args)

{

// Initial string array with month names.

string[] monthName = new string[] { "January","February",

"March","April","May","June","July","August",

"September","October","November","December"};

Console.WriteLine("-------- foreach,in loop");

foreach (string month in monthName)

{

System.Console.WriteLine(month);

}

System.Console.WriteLine();

// Compare the previous loop to a similar for loop.

Console.WriteLine("-------- for loop");

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

{

System.Console.WriteLine(monthName[i]);

| **Exception** | **Condition** |
| --- | --- |
| ArgumentException | A non-null argument that is passed to a method is invalid. |
| ArgumentNullException | An argument that is passed to a method is null. |
| ArgumentOutOfRangeException | An argument is outside the range of valid values. |
| DirectoryNotFoundException | Part of a directory path is not valid. |
| DivideByZeroException | The denominator in an integer or Decimal division operation is zero. |
| DriveNotFoundException | A drive is unavailable or does not exist. |
| FileNotFoundException | A file does not exist. |
| FormatException | A value is not in an appropriate format to be converted from a string by a conversion method such as Parse. |
| IndexOutOfRangeException | An index is outside the bounds of an array or collection. |
| InvalidOperationException | A method call is invalid in an object's current state. |
| KeyNotFoundException | The specified key for accessing a member in a collection cannot be found. |
| NotImplementedException | A method or operation is not implemented. |
| NotSupportedException | A method or operation is not supported. |
| ObjectDisposedException | An operation is performed on an object that has been disposed. |
| OverflowException | An arithmetic, casting, or conversion operation results in an overflow. |
| PathTooLongException | A path or file name exceeds the maximum system-defined length. |
| PlatformNotSupportedException | The operation is not supported on the current platform. |
| RankException | An array with the wrong number of dimensions is passed to a method. |
| TimeoutException | The time interval allotted to an operation has expired. |
| UriFormatException | An invalid Uniform Resource Identifier (URI) is used. |

}

System.Console.WriteLine();

// You can maintain a count of the elements in the collection.

int count = 0;x

foreach (string month in monthName)

{

count += 1;

System.Console.WriteLine("Element #{0}: {1}", count, month);

}

System.Console.WriteLine("Number of elements in the array: {0}", count);

// Keep the console open in debug mode.

Console.WriteLine("Press any key to exit.");

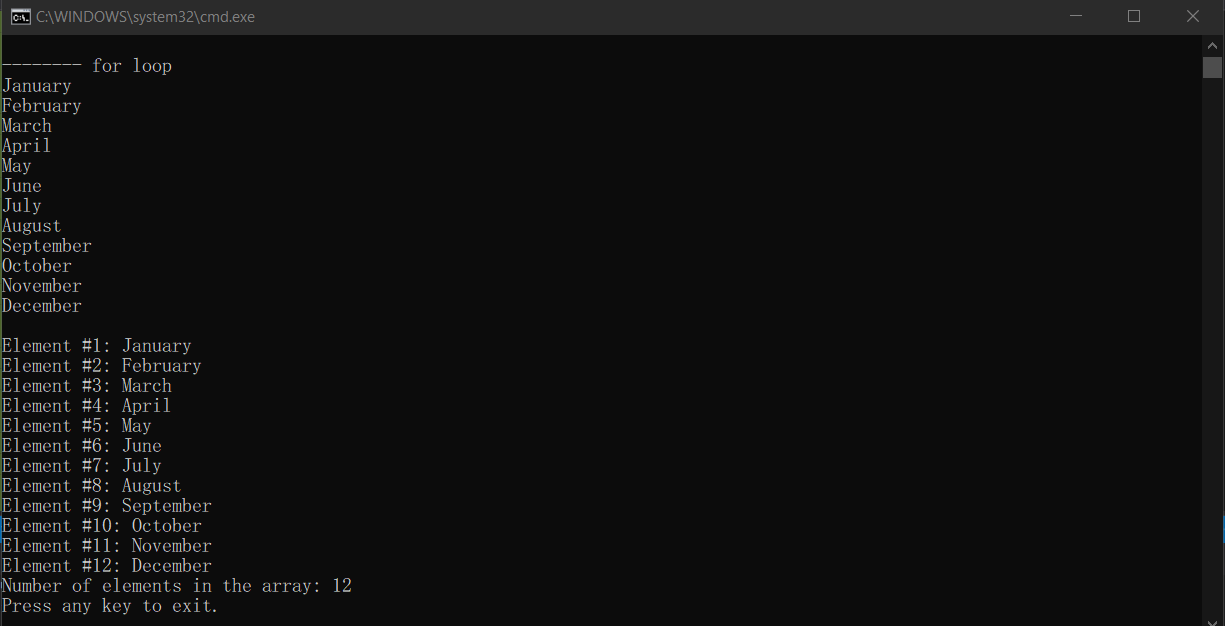
Console.ReadKey();

}

}

}

➢ รันโปรแกรมและบันทึกผล



**Reference เนื้อหาในส่วนนี้เป็นอ้างอิงสำหรับการเขียนโปรแกรม**

**Exceptions**