หน่วยที่ 4

เครื่องมือที่ใช้สำหรับ

เขียนโปรแกรมเชิงวัตถุเบื้องต้น

หัวข้อเรื่อง

4.1 วิธีการเรียกใช้โปรแกรม Visual C#

4.2 โครงสร้างคำสั่งโปรแกรม Visual C#

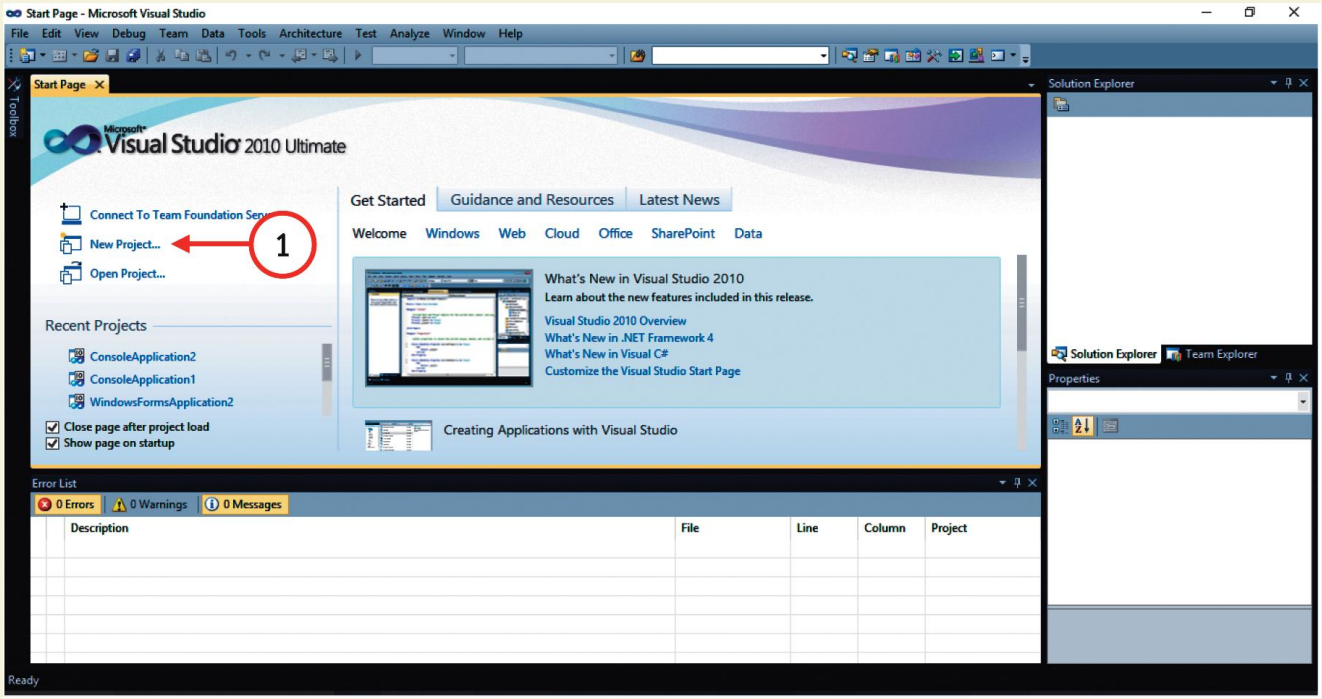
4.3 องค์ประกอบพื้นฐาน Visual C#

4.4 การเขียนโปรแกรมในโหมด Console Application ด้วย Visual C#

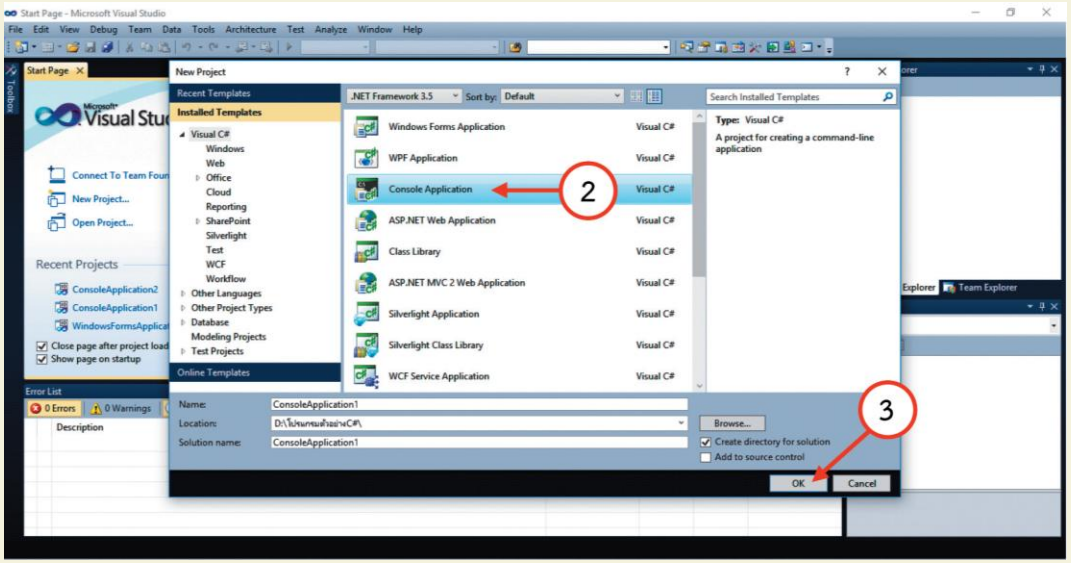
4.1 วิธีการเรียกใช้โปรแกรม Visual C#

โปรแกรม Visual C# หลังจากติดตั้งโปรแกรมเสร็จแล้ว ก็จะสามารถเข้าใช้โปรแกรมได้ทันทีโดยมีขั้นตอนดังนี้

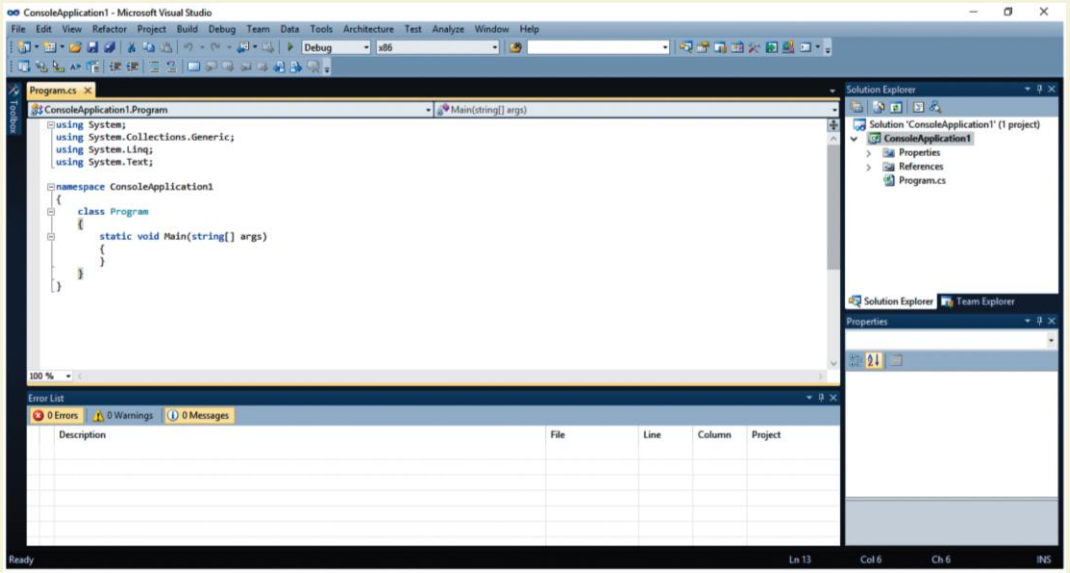
เมื่อเข้าสู่โปรแกรม Visual Studio 2010 จะพบหน้าต่างดังนี้



รูปที่ 4.1 เลือก New Project



รูปที่ 4.2 เลือก Console Application แล้วคลิก OK



รูปที่ 4.3 หน้าต่างโปรแกรม Visual C#

จากนั้นให้ทดลองเขียนโปรแกรมตัวอย่างดังนี้ต่อไปนี้

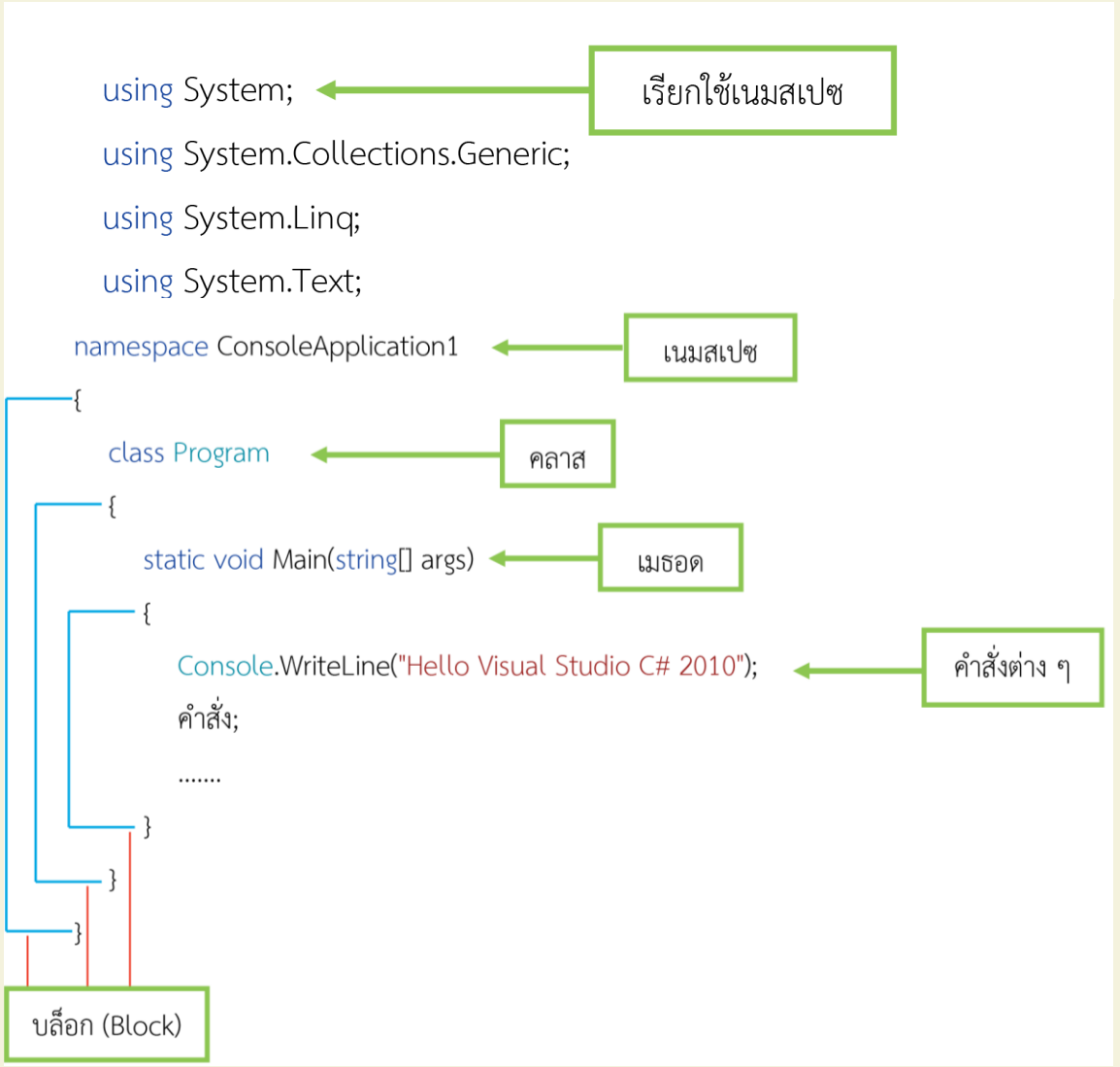
```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello Visual Studio C# 2010");
            Console.ReadKey();
        }
    }
}
```

ผลลัพธ์ที่ได้จากการรันโปรแกรมดังนี้

Hello Visual Studio C# 2010

4.2 โครงสร้างคำสั่งโปรแกรม Visual C#

โปรแกรมภาษา Visual C# มีรูปแบบโครงสร้างคำสั่งดังต่อไปนี้



4.3.1 บล็อก {...}

รูปแบบของโปรแกรม Visual C# จะใช้บล็อก {...} ในการกำหนดจุดเริ่มต้นและจุดสิ้นสุดของการทำงานในแต่ละส่วนของโปรแกรม ซึ่งภายในแต่ละบล็อกอาจมีบล็อกย่อย ๆ ซ้อนลงไปได้อีก ตามลักษณะของงาน เช่น

```
Class className
{
    If (.....)
    {
        for (.....)
        {
        .....
        .....
        }
.....
.....
    }
}
```

4.3.2 เครื่องหมายสิ้นสุดคำสั่ง (;)

ในโปรแกรม Visual C# เราจะใช้เครื่องหมายเซมิโคลอน (;) เป็นตัวแสดงจุดสิ้นสุดของแต่ละคำสั่ง หากเราไม่ใส่เครื่องหมายนี้เพื่อคั่นระหว่างคำสั่งแล้ว โปรแกรมจะถือว่าเป็นคำสั่งเดียวกันไปตลอดถึงแม้ว่าจะอยู่คนละบรรทัดก็ตาม เช่น

```
X = 20;
Y = “xxx”;
Z = x +
20;
```

เมื่อเครื่องหมาย ; เป็นตัวบ่งบอกว่าเป็นจุดสิ้นสุดของคำสั่ง คำสั่งต่าง ๆ จึงจะสามารถมาอยู่ในบรรทัดเดียวกันได้ เช่น

```
X = 20; y = “xxx”; z = x + 20;
```

แต่ในการเขียนรหัสคำสั่งในลักษณะนี้ จะไม่นิยมเขียน เนื่องจากอ่านโปรแกรมได้ยาก และรหัสโปรแกรมดูไม่เป็นระเบียบ แต่อาจนำมาใช้ในบางกรณีได้

4.3.3 การเขียนคำอธิบายประกอบในรหัสโปรแกรม

คำอธิบาย หมายถึง การเขียนข้อความใด ๆ ที่ไม่ใช่คำสั่งแต่เขียนปะปนกันไปกับคำสั่งอื่น ๆ เพื่ออธิบายเรื่องใดเรื่องหนึ่งเอาไว้เพื่อความเข้าใจในรหัสโปรแกรมตรงส่วนนั้น ๆ โดยทั้งนี้เพื่อให้โปรแกรมไม่สับสนว่าส่วนใดเป็นคำสั่ง ส่วนใดเป็นแค่เพียงคำอธิบายการแทรกคำอธิบาย โดยสามารถทำได้ 2 ลักษณะดังนี้ คือ

1. รูปแบบ // คำอธิบาย

จะใช้สำหรับอธิบายบรรทัดเดียว โดยโปรแกรมจะถือว่าตั้งแต่สัญลักษณ์ // เป็นต้นไปจนถึงสิ้นสุดบรรทัดจะเป็นคำอธิบายทั้งหมด จะไม่นำมาพิจารณาในการประมวลผลของโปรแกรม เช่น

```
// สูตรการคำนวณหาขนาดพื้นที่สี่เหลี่ยมผืนผ้า
RectangleArea=width*length;
หรือ
RectangleArea=width*length; //สูตรการคำนวณหาขนาดพื้นที่สี่เหลี่ยมผืนผ้า
```

2. รูปแบบ /* คำอธิบาย */

ในกรณีที่คำอธิบายโปรแกรมของเราค่อนข้างยาว จำเป็นต้องเขียนหลาย ๆ บรรทัด การใช้// หลาย ๆ ครั้ง จึงอาจไม่สะดวกนัก เราสามารถใช้ /*...*/ ครอบคำอธิบายนั้นแทนได้ ซึ่งโปรแกรมจะถือว่า /*เป็นต้นไปจะเป็นคำอธิบายทั้งหมด จนกว่าจะเจอสัญลักษณ์ */ จึงจะถือว่าเป็นการสิ้นสุดคำอธิบาย เช่น

```
/* นี่เป็นส่วนของคำอธิบาย
   ที่ไม่มีผลต่อการทำงานของโปรแกรม
   มีไว้เพื่อช่วยให้ทำความเข้าใจในการเขียน
รหัสคำสั่งได้ง่ายขึ้น */
```

4.4.1 คำสั่งการแสดงข้อมูลออกทางจอภาพ แบ่งออกเป็น 2 คำสั่ง ดังต่อไปนี้

1. คำสั่ง Write()

รูปแบบคำสั่ง

Console.Write(**อักขระ/สตริง/ตัวเลข/ตัวแปร/นิพจน์/เมธอด**);

คำอธิบาย การแสดงข้อมูลสามารถแสดงได้หลายรูปแบบดังต่อไปนี้

(1) **อักขระ** หมายถึง ตัวอักขระทั่ว ๆ ไป โดยตัวอักขระจะต้องอยู่ภายใต้เครื่องหมาย ‘…’และในเครื่องหมายนี้จะมีอักขระได้เพียง 1 ตัวอักขระเท่านั้น เช่น

```
Console.Write('A');
```

(2) **สตริง** หมายถึง ประโยคข้อความต่าง ๆ โดยสตริงจะต้องอยู่ภายใต้เครื่องหมาย “…”และจะมีกี่ตัวอักขระก็ได้ เช่น

```
Console.Write("Hello");
```

(4) **ตัวแปร** หมายถึง ตัวแปรทุกตัวในโปรแกรม ไม่ว่าจะเป็นตัวแปรชนิดสตริง ตัวเลขหรือบูลีน เช่น

```
int age = 50;
Console.Write(age);
```

(5) **นิพจน์** หมายถึง นิพจน์ต่าง ๆ เช่น นิพจน์ทางคณิตศาสตร์ นิพจน์เปรียบเทียบ เป็นต้น เช่น

```
int age = 50;
Console.Write(age/2); //นิพจน์ทางคณิตศาสตร์
Console.Write(age>30); //นิพจน์เปรียบเทียบ
```

(6) **เมธอด** หมายถึง ทั้งเมธอดสำเร็จและเมธอดที่เราสร้างขึ้นที่มีการส่งค่ากลับ เช่น

```
Console.Write(Math.sqrt(9));
```

2. คำสั่ง WriteLine()

รูปแบบคำสั่ง

Console.WriteLine(**อักขระ/สตริง/ตัวเลข/ตัวแปร/นิพจน์/เมธอด**);

จะเห็นว่าคำสั่ง WriteLine() สามารถแสดงข้อมูลได้เหมือนกับคำสั่ง Console.Write() แต่แตกต่างกันที่ผลลัพธ์ในการแสดงผล ดังตัวอย่างต่อไปนี้

```
Console.Write(1);
Console.Write(2);
Console.Write(3);
```

ผลลัพธ์ 123

```
Console.WriteLine(1);
Console.WriteLine(2);
Console.WriteLine(3);
```

ผลลัพธ์ 1
2
3

4.4.2 คำสั่งอ่านข้อมูลจากคีย์บอร์ด

เป็นคำสั่งสำหรับอ่านหรือรับข้อมูลจากคีย์บอร์ด ณ จุดที่รับคำสั่งหรือจุดที่เคอร์เซอร์รออยู่ มี2 รูปแบบคือ

1. Console.Read() เป็นการอ่านค่าข้อมูลในรูปแบบของ Integer (int) ดังตัวอย่าง

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication1
{
    class Program
        {
            static void Main(string[] args)
            {
                int i = Console.Read();
            }
        }
}
```

2. Console.ReadLine() เป็นการอ่านค่าข้อมูลในรูปแบบของ string ดังตัวอย่าง

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication1
{
    class Program
        {
            static void Main(string[] args)
            {
                string str = Console.ReadLine();
            }
        }
}
```

ตัวอย่างที่ 1

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication1
{   class Program
    {
        static void Main(string[] args)
        {
        Console.Read();
        }
    }
}
```

ผลลัพธ์โปรแกรม

Empty program _

ตัวอย่างที่ 2

```
using System;
using System.Collections.Generic;
using System.Text;
    namespace ConsoleApplication1
    {
        class Program
        {
            static void Main(string[] args)
                {
                    Console.Write("Programming in C# is easy.");
                    Console.Read();
                }
        }
    }
```

ผลลัพธ์โปรแกรม

Programming in C# is easy._