

Supplement 3

Unsafe Code and Pointers in C#

Unsafe Code and Pointers in C#

Objectives

After completing this unit you will be able to:

- **Describe the use of unsafe code in C#.**
- **Use the C# pointer type.**

Unsafe Code

- **The mainstream use of C# is to write *managed code*, which runs on the Common Language Runtime.**
- **With the classical .NET Framework, it is quite possible for a C# program to call unmanaged code, such as a legacy COM component, which runs directly on the operating system.**
 - This facility is important, because a tremendous amount of legacy code exists, which is all unmanaged.
 - There is overhead in transitioning from a managed environment to an unmanaged one and back again.
- **C# provides another facility, called *unsafe code*, which allows you to bypass .NET memory management and get at memory directly, while still running on the CLR.**
 - In particular, in unsafe code you can work with **pointers**, which we will discuss later in this appendix.

Unsafe Blocks

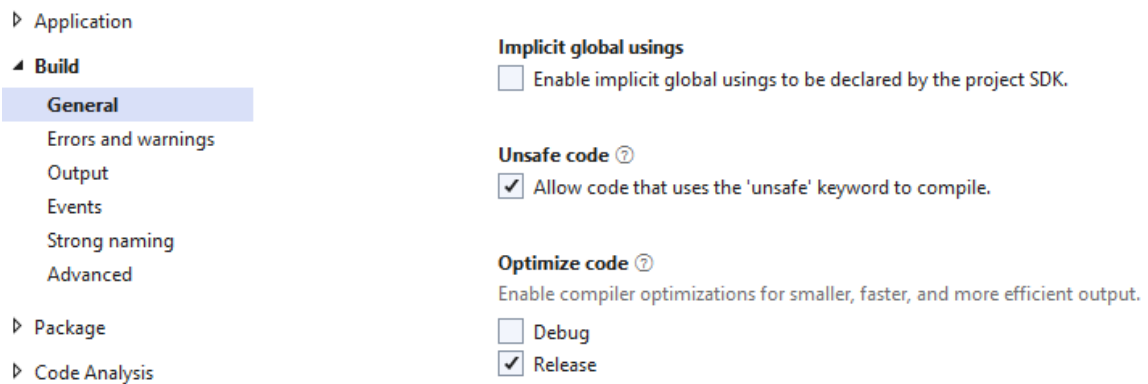
- The most circumspect use of unsafe code is within a block, which is specified using the C# keyword *unsafe*.
- The program *UnsafeBlock* illustrates using the *sizeof* operator to determine the size in bytes of various data types.
 - You will get a compiler error if you try to use the **sizeof** operator outside of unsafe code.

```
// UnsafeBlock.cs
using System;
struct Account
{
    private int id;
    private decimal balance;
}

public class UnsafeBlock
{
    public static void Main()
    {
        unsafe
        {
            Console.WriteLine("size of int = {0}",
                              sizeof(int));
            Console.WriteLine("size of decimal = {0}",
                              sizeof(decimal));
            Console.WriteLine("size of Account = {0}",
                              sizeof(Account));
        }
    }
}
```

Unsafe Option in Visual Studio

- **To set the unsafe option in Visual Studio, perform the following steps:**
 - Right-click over the project in the Solution Explorer and choose Properties.
 - In the Property Pages window that comes up, click on Build.
 - In the General section, check Allow unsafe code.
 - You can now compile your project in unsafe mode.



- **Here is the output from running the program:**

```
size of int = 4
size of decimal = 16
size of Account = 24
```

Pointers

- **C# has three kinds of data types:**
 - Value types, which directly contain their data
 - Reference types, which refer to data contained somewhere else
 - Pointer types
- **Pointer types can only be used in unsafe code.**
 - A pointer is an **address** of an actual memory location.
 - A pointer variable is declared using an asterisk after the data type.
 - To refer to the data a pointer is pointing to, use the **dereferencing** operator, which is an asterisk before the variable.
 - To obtain a pointer from a memory location, apply the **address of** operator, which is an ampersand in front of the variable.
- **Here are some examples.**

```
int* p;      // p is a pointer to an int
int a = 5;   // a is an int, with 5 stored
p = &a;      // p now points to a
*p = 12;     // 12 is now stored in location pointed
              // to by p. So a now has 12 stored
```

Pointers (Cont'd)

- **Pointers were widely used in the C programming language because functions in C only pass data by value.**
- **Thus, if you want a function to return data, you must pass a pointer rather than the data itself.**
- **The program *UnsafePointer* illustrates a *Swap* method, which is used to interchange two integer variables.**
 - Since the program is written in C#, we can pass data by reference.
 - We illustrate with two overloaded versions of the **Swap** method, one using **ref** parameters and the other using pointers.
 - Rather than using an **unsafe** block, this program uses **unsafe** methods, which are defined by including **unsafe** among the modifiers of the method.
 - Both the **Main** method and the one **Swap** method are unsafe.
- **Again, you should compile the program using the **unsafe** option.**
 - The first swap interchanges the values. The second swap brings the values back to their original state.

Swapping Via Pointers

```
static unsafe void Main(string[] args)
{
    int x = 55;
    int y = 777;
    Show("Before swap", x, y);
    Swap(ref x, ref y);
    Show("After swap", x, y);
    Swap(&x, &y);
    Show("After unsafe swap", x, y);
}
static void Show(string s, int x, int y)
{
    Console.WriteLine("{0}: x = {1}, y = {2}",
                      s, x, y);
}
static void Swap(ref int x, ref int y)
{
    int temp = x;
    x = y;
    y = temp;
}
static unsafe void Swap(int* px, int* py)
{
    int temp = *px;
    *px = *py;
    *py = temp;
}
```

- **Here is the output:**

Before swap: x = 55, y = 777

After swap: x = 777, y = 55

After unsafe swap: x = 55, y = 777

Fixed Memory

- **When working with pointers, there is a pitfall.**
 - Suppose you have obtained a pointer to a region of memory that contains data you are working on.
 - Since you have a pointer, you are accessing memory directly.
 - But suppose the garbage collector collects garbage and moves data about in memory.
- **Then your object may now reside at a different location, and your pointer may no longer be valid.**
- **To deal with such a situation, C# provides the keyword *fixed*, which declares that the memory in question is “pinned” and cannot be moved by the garbage collector.**
 - Note that you should use **fixed** only for temporary, local variables, and you should keep the scope as circumscribed as possible.
 - If too much memory is pinned, the CLR memory management system cannot manage memory efficiently.

Fixed Memory Illustration

- The program *UnsafeAccount* illustrates working with *fixed* memory.
 - This program declares an array of five **Account** objects, and then assigns them all the same value.
 - The attempt to determine the size of this array is commented out, because you cannot apply the **sizeof** operator to a managed type such as **Account[]**.
- It also illustrates the arrow operator for dereferencing a field in a struct when you have a pointer to the struct.
 - For example, if **p** is a pointer to an instance of the struct **Account** shown below, the following code will assign values to the account object pointed to by **p**.

```
p->id = 101;           // assign the id field
p->balance = 50.00m;   // assign the balance field
```

- The following code displays an account object.

```
private static unsafe void ShowAccount(
    Account* pAcc)
{
    Console.WriteLine("id = {0}, balance = {1:C}",
        pAcc->id, pAcc->balance);
}
```

Fixed Memory Illustration (Cont'd)

```
static unsafe void Main(string[] args)
{
    int id = 101;
    decimal balance = 50.55m;
    Account acc = new Account(id, balance);
    ShowAccount(&acc);
    Account[] array = new Account[5];
    // Console.WriteLine("size of Account[] = {0}",
    //     sizeof(Account[]));
    ShowArray(array);
    fixed (Account* pStart = array)
    {
        Account* pAcc = pStart;
        for (int i = 0; i < array.Length; i++)
            *pAcc++ = acc;
    }
    ShowArray(array);
}

static unsafe void ShowAccount(
    Account* pAcc)
{
    Console.WriteLine("id = {0}, balance = {1:C}",
        pAcc->id, pAcc->balance);
}

static void ShowAccount(Account acc)
{
    Console.WriteLine("id = {0}, balance = {1:C}",
        acc.id, acc.balance);
}

static void ShowArray(Account[] array)
{
    for (int i = 0; i < 5; i++)
    {
        ShowAccount(array[i]);
    }
}
```

Summary

- **Unsafe code in C# allows you to bypass .NET memory management and get at memory directly.**
- **You may use unsafe code within a block with the C# keyword *unsafe*.**
- **The *unsafe* keyword can also be applied to a method.**
- **Code with either an unsafe block or an unsafe method must be compiled with the unsafe compiler option.**
- **You can use the C# pointer type in unsafe code.**
- **When working with pointers you may need to use the *fixed* keyword to pin memory, preventing it from being moved around by the garbage collector.**
- **Unsafe code should be used sparingly if at all!**