4.8 Unsafe Code and Pointers

C# supports direct memory manipulation via pointers within blocks of code marked
ompiler option. Pointer types are primarily useful
also be used for accessing memory outside the
al hotspots.

ıere is a corresponding pointer type V*. A pointer
ıis is considered to be of type V, but pointer
pointer type. Table 4-2 lists the main pointer
operators.

Table 4-2. Principal pointer operators

4.8.2 Unsafe Code

By marking a type, type member, or statement block with the `unsafe` keyword, you're permitted to use pointer types and perform C++ style pointer operations on memory within that scope. Here is an example of using pointers with a managed object:

```
unsafe void RedFilter(int[,] bitmap) {
  const int length = bitmap.Length;
  fixed (int* b = bitmap) {
    int* p = b;
    for(int i = 0; i < length; i++)
      *p++ &= 0xFF;
  }
}
```

Unsafe code typically runs faster than a corresponding safe implementation, which in this case would have required a nested loop with array indexing and bounds checking. An unsafe C# method may also be faster than calling an external C function, since there is no overhead associated with leaving the managed execution environment.

4.8.3 The fixed Statement

```
fixed ([value-type | void ]* name = [&]? expr )
 statement-block
```

The `fixed` statement is required to pin a managed object, such as the bitmap in the previous example. During the execution of a program, many objects are allocated and deallocated from the heap. In order to avoid unnecessary waste or fragmentation of memory, the garbage collector moves objects around. Pointing to an object is futile if its address could change while referencing it, so the `fixed` statement tells the garbage collector to "pin" the object and not move it around. This may have an impact on the efficiency of the runtime, so fixed blocks should be used only briefly, and heap allocation should be avoided within the fixed block.

C# returns a pointer only from a value type, never directly from a reference type. Syntactically, arrays and strings are an exception to this, since they actually return a pointer to their first element (which must be a value type), rather than the objects themselves.

Value types declared inline within reference types require the reference type to be pinned, as follows:

```
class Test {
  int x;
  static void Main( ) {
    Test test = new Test ( );
    unsafe {
       fixed(int* p = &test.x) { // pins test
         *p = 9;
       }
       System.Console.WriteLine(test.x);
    }
  }
}
```

4.8.4 The Pointer-to-Member Operator

In addition to the & and * operators, C# also provides the C++-style `->` operator, which can be used on structs:

```
struct Test {
   int x;
   unsafe static void Main( ) {
      Test test = new Test( );
      Test* p = &test;
      p->x = 9;
      System.Console.WriteLine(test.x);
   }
}
```

4.8.5 The stackalloc Keyword

Memory can be allocated in a block on the stack explicitly using the `stackalloc` keyword. Since it is allocated on the stack, its lifetime is limited to the execution of the method, just as with any other local variable. The block may use `[ ]` indexing, but is purely a value type with no additional self-describing information or bounds-checking that an array provides.

```
int* a = stackalloc int [10];
for (int i = 0; i < 10; ++i)
   Console.WriteLine(a[i]); // print raw memory
```

```
class Test {
  unsafe static void Main ( ) {
    short[ ] a = {1,1,2,3,5,8,13,21,34,55};
      fixed (short* p = a) {
        // sizeof returns size of value-type in bytes
        Zap (p, a.Length * sizeof (short));
      }
    foreach (short x in a)
      System.Console.WriteLine (x); // prints all zeros
  }
  unsafe static void Zap (void* memory, int byteCount) {
    byte* b = (byte*)memory;
      for (int i = 0; i < byteCount; i++)
        *b++ = 0;
  }
}
```

4.8.7 Pointers to Unmanaged Code

Pointers are also useful for accessing data outside the managed heap (such as when interacting with C DLLs or COM), or when dealing with data not in the main memory (such as graphics memory or a storage medium on an embedded device).

0 Comments **etutorials** 🔒 **Disqus' Privacy Policy** **Petr Antoš**

♡ **Favorite** Tweet Share **Sort by Best**

Start the discussion…

Be the first to comment.

✉ **Subscribe** **Add Disqus to your siteAdd DisqusAdd** ⚠ **Do Not Sell My Data**

-34%