استفاده از اشاره گرها در C#

استفاده از اشاره گرها (Pionters) در C# به ندرت صورت می گیرد. ولی در پاره ای از مواقع نیاز به استفاده از آنها احساس می شود. بدین منظور پدید آورندگان C# امکانی برای استفاده کنندگان فراهم آورده اند که می توان با استفاده از آن اشاره گرها را در C# بوجود آورد و از آنها استفاده کرد. چند مورد استفاده از اشاره گرها به شرح زیر می باشد.

- کارایی بسیار زیاد.
- ارتباط و استفاده از ساختارهای باینری موجود.
- تعامد پیشرفته با اشیاء COM.

طراحان C# به کدهایی که در آنها از اشاره گرها استفاده شده نام Unsafe یا غیر مطمئن داده اند. استفاده از کد unsafe تنها در صورتی مجاز می باشد که همه چیز تحت کنترل بوده و کد مورد نظر از هر لحاظ قابل اطمینان باشد. در این مقاله بطور مختصر به چگونگی بکاربردن این خصوصیت C# اشاره شده است.

برای استفاده از اشاره گر نیاز به انجام اعمال زیر می باشد.

- علامت گذاری کلاس، ساختار و متد مورد نظر با شناسه unsafe.
- استفاده از اشاره گر برای برطرف ساختن نیاز موجود.
- کامپایل کردن کد ایجاد شده با سوئیچ unsafe/.

در زیر دو مثال برای استفاده از اشاره گرها بیان شده است.

سری فیبوناچی

در اینجا الگریتم سری فیبوناچی توسط C# و unsafe code شرح داده می شود. در سری فیبوناچی هر عدد از جمع دو عدد ماقبل آن بدست می آید. با این تفاوت که دو عدد اول سری، عدد ۱ می باشند.

```
// compile with: /unsafe
using System;
class Test
{
  public static unsafe void Main()
  {
    int* fib = stackalloc int[100];
    int* p = fib;
    *p++ = *p++ = 1;
    for (int i=2; i<100; ++i, ++p)
      *p = p[-1] + p[-2];
    for (int i=0; i<10; ++i)
      Console.WriteLine (fib[i]);
  }
}
```

نکته قابل توجه در این مثال استفاده از شناسه stackalloc می باشد. با استفاده از این شناسه می توان بلاکی از حافظه را در اختیار گرفت. فضای اختصاص داده شده توسط Garbage Collector قابل جمع اوری نمی باشد و محدوده آن در همان متد و یا بلاک استفاده شده می باشد. آدرس اول بلاک را می توان در یک اشاره گر ذخیره نمود. همانطور که در کد بالا مشاهده می نمایید اشاره گرها در C# شبیه به C می باشند.

کپی کردن عناصر آرایه

در این مثال عناصر یک آرایه در آرایه دیگر با استفاده از اشاره گرها ذخیره سازی می شوند.

```
// compile with: /unsafe
using System;
```

```
class Test
{
   // The unsafe keyword allows pointers to be used within
   // the following method:
   static unsafe void Copy(byte[] src, int srcIndex,
       byte[] dst, int dstIndex, int count)
   {
      if (src == null || srcIndex < 0 ||
          dst == null || dstIndex < 0 || count < 0)
      {
          throw new ArgumentException();
      }
      int srcLen = src.Length;
      int dstLen = dst.Length;
      if (srcLen - srcIndex < count ||
          dstLen - dstIndex < count)
      {
          throw new ArgumentException();
      }

      // The following fixed statement pins the location of
      // the src and dst objects in memory so that they will
      // not be moved by garbage collection.
      fixed (byte* pSrc = src, pDst = dst)
      {
        byte* ps = pSrc;
```

```
      byte* pd = pDst;

      // Loop over the count in blocks of 4 bytes, copying an
      // integer (4 bytes) at a time:
      for (int n =0 ; n < count/4 ; n++)
      {
        *((int*)pd) = *((int*)ps);
        pd += 4;
        ps += 4;
      }

      // Complete the copy by moving any bytes that weren't
      // moved in blocks of 4:
      for (int n =0; n < count%4; n++)
      {
        *pd = *ps;
        pd++;
        ps++;
      }
    }
}



static void Main(string[] args)
{
   byte[] a = new byte[100];
   byte[] b = new byte[100];
```

```
    for(int i=0; i<100; ++i)
      a[i] = (byte)i;
    Copy(a, 0, b, 0, 100);
    Console.WriteLine("The first 10 elements are:");
    for(int i=0; i<10; ++i)
      Console.Write(b[i] + " ");
    Console.WriteLine("\n");
  }
}
```

نکته قابل توجه در این مثال استفاده از عبارت fixed می باشد. با استفاده از این عبارت می توان از اختصاص فضای مجدد به متغیر یا متغییرها توسط Collector Garbage جلوگیری نمود.