عنوان: پیش پردازنده ها Preprocessors
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۰۹/۱۸ ۲۲:۰
آدرس: www.dotnettips.info
گروه‌ها: C#, preprocessors

احتمالا شما با پیش پردازنده ها کم و بیش آشنایی دارید؛ برای آشنایی با پیش پردازنده‌های موجود در سی شارپ می‌توانید به این آدرس بروید.
البته این پیش پردازنده‌ها به قدرتمندی سایر پیش پردازنده هایی که در زبان‌های دیگر مانند سی یا سی پلاس پلاس دیده‌اید نیستند. مثلا نمی‌توانند مقدار دیگری جز مقدارهای بولین دریافت کنند، یا از حافظه‌ی مصرفی استفاده کنند. همچنین باید به یاد داشته باشید که حتما باید قبل از شروع کد، از پیش پردازنده‌های استفاده کنید.

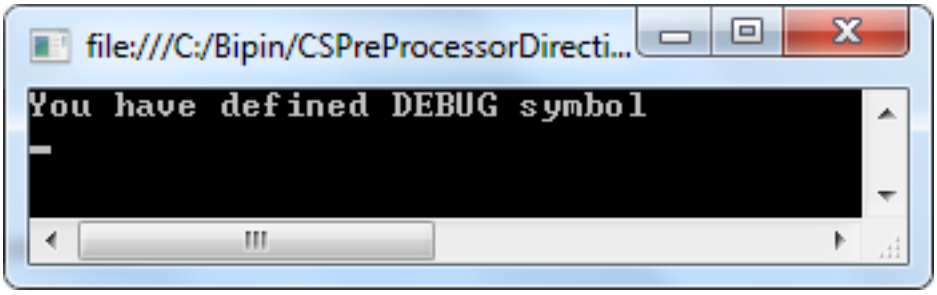
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

#define DEBUG
```

Cannot define/undefine preprocessor symbols after first token in file

برای تعریف یک سمبل symbol می‌توانید از پیش پردازنده‌ی #define استفاده و برای حذف آن هم از #undef استفاده کنید. رسم هست که سمبل‌ها با حروف بزرگ تعریف شوند.
عبارات #if,#else,#elif,#endif هم عبارات شرطی هستند که می‌توان برای چک کردن یک سمبل از آن‌ها استفاده کرد:

```
#define DEBUG
...
#if DEBUG
    Console.WriteLine("You have defined DEBUG symbol");
#endif
```

نتیجه آن را می‌توانید در تصویر زیر مشاهده کنید:




بدیهی است که همین سمبل DEBUG را undef کنید متن بالا نمایش داده نخواهد شد.
بهتر است به پیش پردازنده‌های دیگر هم نگاهی بیندازیم:

```
#if STANDARD
    Console.WriteLine("You have defined STANDARD symbol");
#elif PROFESSIONAL
    Console.WriteLine("You have defined PROFESSIONAL symbol");
#elif ULTIMATE
    Console.WriteLine("You have defined ULTIMATE symbol");
#endif
```

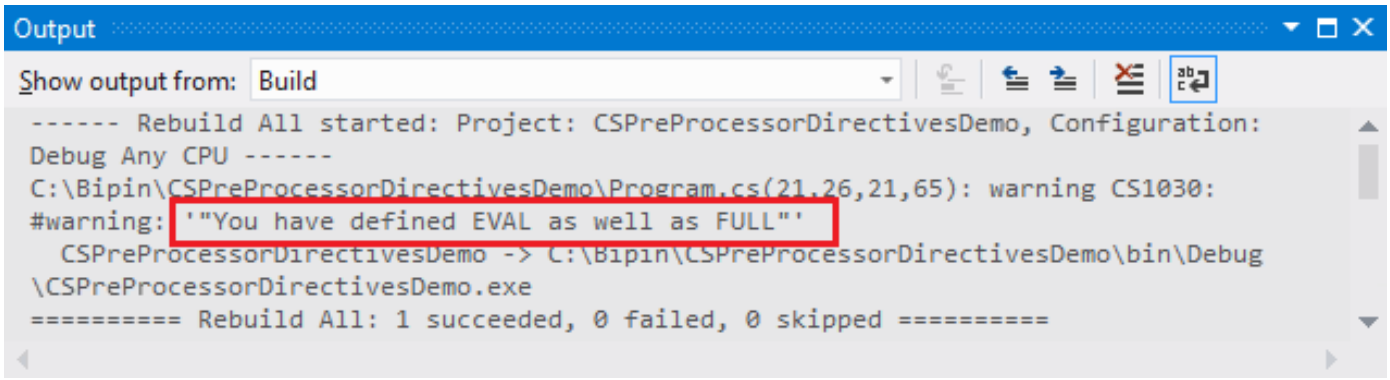
حتی می‌توانید از عملگرهای شرطی چون && یا || یا == یا != و... هم استفاده کنید. تکه کد زیر، از این عملگرها بهره جسته است:

```
#if STANDARD && EVAL
    Console.WriteLine("You have defined STANDARD and EVAL symbols");
#endif
```

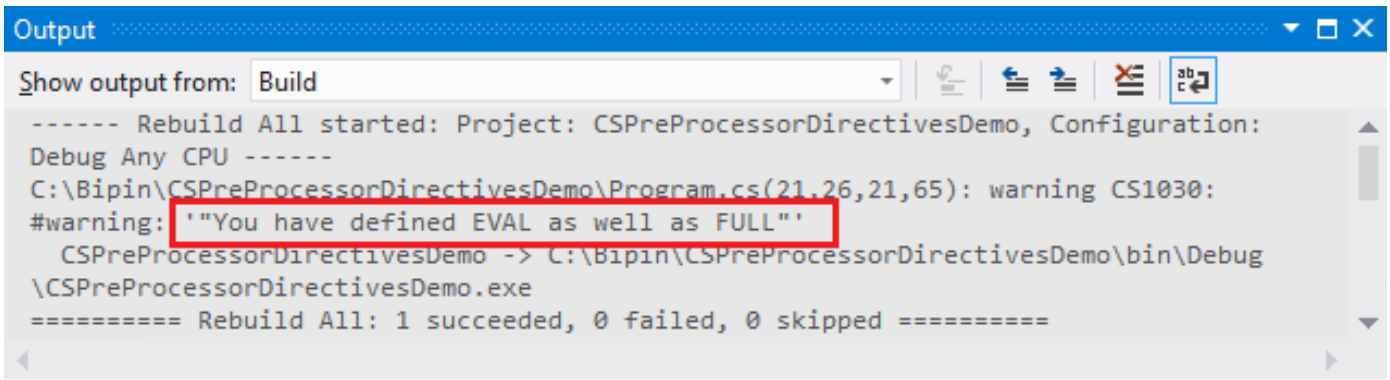
**پیش پردازنده‌های #warning و #error**

در پیش پردازنده #warning می‌توانید یک پیام هشدار یا اخطار را به پنجره‌ی warning ارسال کنید؛ ولی برنامه کماکان به اجرای خود ادامه می‌دهد. اما با #error برنامه هم پیام خطا را در پنجره مربوطه نمایش می‌دهد و هم باعث halt شدن برنامه می‌شود.

```
#if STANDARD && EVAL
    Console.WriteLine("You have defined STANDARD and EVAL symbols");
#endif
```




در کد بالا #warning را با #error جابجا می‌کنیم:




**region# و endregion#**

از این دو عبارت در بین کدها استفاده می‌کنیم. برای بلوک بندی کدها می‌توان از آن‌ها استفاده کرد. برای مثال دسته بندی کدهای نوشته شده مثل جدا کردن propertyها یا رویدادها یا متدها و ...، با محصور شدن تکه کدهای بین این دو، یک علامت + یا - برای انجام عمل expand و collapsed ایجاد می‌شود.

```
#region Public Properties
    public string CustomerID { get; set; }
    public string CompanyName { get; set; }
    public string ContactName { get; set; }
    public string Country { get; set; }
#endregion
```

**line#**

برای تغییر نام فایل و شماره خطوط در هنگام دیباگ (نمایش خطا و هشدارها در پنجره‌ی نمایش خطاها) به کار می‌رود.

مثلا به تکه کد زیر دقت کنید و همچنین به تصویر بعد از آن، بدون نوشتن line# دقت کنید:

```
namespace CSPreProcessorDirectivesDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            inta a = 100;
            Console.ReadLine();
        }
    }
}
```

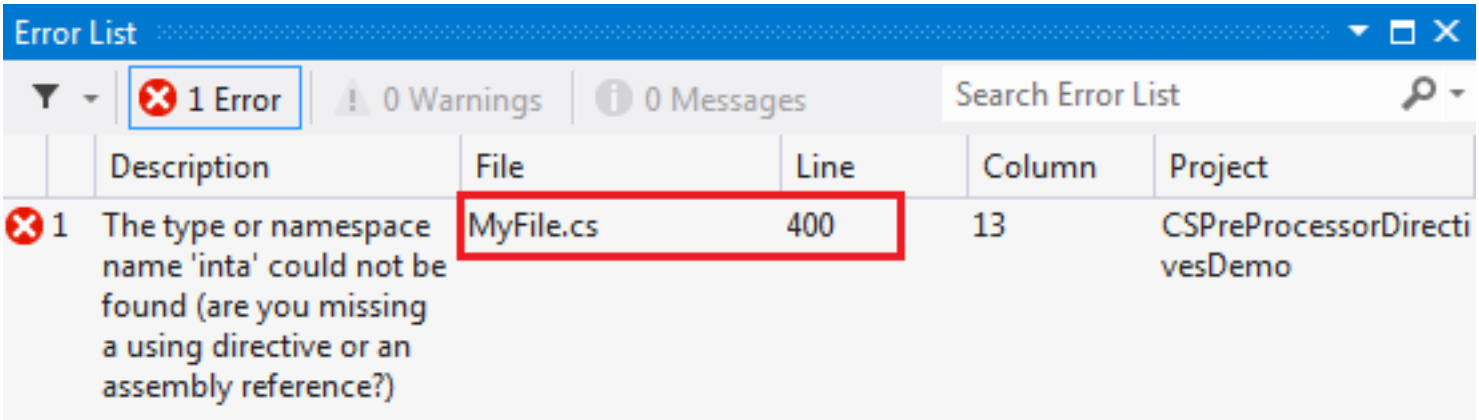
Error List

1 Error | 0 Warnings | 0 Messages | Search Error List

| | Description | File | Line | Column | Project |
|---|---|---|---|---|---|
| 1 | The type or namespace name 'inta' could not be found (are you missing a using directive or an assembly reference?) | Program.cs | 14 | 13 | CSPreProcessorDirectivesDemo |

خطای ما در خط 14 فایل program.cs رخ داده است. در تکه کد زیر پیش پردازنده line# را اضافه کردیم:

```
#line 400 "MyFile.cs"
inta a = 100;
```




همانطور که می‌بینید آدرس تکه کد یا خط بعد از آن تغییر پیدا کرد و از آنجا به بعد از 400 به بعد شمرده می‌شود.
طبق منابع نوشته شده این پیش پردازنده موقعی بیشتر سودمند هست که تکه کد، توسط ابزارهای خارجی یا سیستمی ویرایش شده باشد.
در صورتیکه از #line default استفاده کنید، از آن نقطه به بعد، نام فایل و شماره خطاها به صورت عادی اعلام می‌شوند و #line قبلی در نظر گرفته نمی‌شود تا شاید اگر دوباره به #line جدیدی برخورد کند.
#line hidden هم تکه کدهای مربوطه را از دید دیباگر مخفی می‌کند مثل موقعیکه برنامه نویس، کد به کد یا خط به خط برنامه را دیباگ می‌کند ولی از اینجا به بعد از روی این خطوط رد می‌شود تا به یک #line دیگر برسد. منظور از رد شدن، عدم اجرای خطوط نیست؛ بلکه دیباگ خط به خط می‌باشد.

**#progma**

این پیش پردازنده از دو بخش نام دستور و آگومان‌ها تشکیل شده است:

```
#pragma pragma-name pragma-arguments
```

دات نت از دو نام دستور warning و checksum پشتیبانی می‌کند؛ آرگومان‌هایی که با دستور warning می‌پذیرد:

```
#pragma warning disable
#pragma warning restore
```

با آرگومان disabled تمامی هشدارهای خطوط بعد از آن نادیده گرفته شده و اعلام نمی‌شوند و از restore برای بازگشت از حالت disabled به کار می‌رود. همچنین برای غیر فعال کردن هشدار برای خط یا خطوط خاص هم میتوانید به صورت زیر بنویسید:

```
#pragma warning disable 414
#pragma warning disable 414, 3021
```

**#checksum**

```
#pragma checksum "filename" "{guid}" "checksum bytes"
```

از این یکی برای ذخیره هشدارها و خطاها در program database یا PDB استفاده می‌شود (برای مواقعیکه پروژه شما قرار است به یک com یا dll تبدیل شود؛ کاربردی زیادی دارد). آرگومان اول نام فایل که بعدا برای مانیتور کردن به راحتی بین کلاس‌ها تشخیص داده شود و دومی که GUID است و همین GUID را باید برای فایل مشخص کنید.

```
// Guid for the interface IMyInterface.
[Guid("F9168C5E-CEB2-4faa-B6BF-329BF39FA1E4")]
interface IMyInterface
{
    void MyMethod();
```

```
}

// Guid for the coclass MyTestClass.
[Guid("936DA01F-9ABD-4d9d-80C7-02AF85C822A8")]
public class MyTestClass : IMyInterface
{
    public void MyMethod() {}
}
```

و checksum _bytes که باید به صورت هگزادسیمال در حالت رشته‌ای نوشته شود و باید بیانگر یک عدد زوج باشد؛ در صورتیکه یک عدد فرد را مشخص کنید، کمپایلر پیش پردازنده شما را در نظر نمی‌گیرد. نهایتا به صورت زیر نوشته می‌شود:

```
class TestClass
{
    static int Main()
    {
        #pragma checksum "file.cs" "{3673e4ca-6098-4ec1-890f-8fceb2a794a2}" "{012345678AB}" // New
checksum
    }
}
```

**منابع :** http://www.codeguru.com/csharp/.net/using-preprocessor-directives-in-c.htm

http://msdn.microsoft.com/en-us/library/ms173226.aspx