سی شارپ به زبان ساده

مروری بر برنامه نویسی

دنیای برنامه نویسی دنیای عجیبی است،برنامه نویسی به شما اجازه می دهد که با کامپیوترتان مانند یک رئیس رفتار کنید. برنامه نویسی می تواند شغل یا تفریح شما باشد،اما هدف اصلی از برنامه نویسی ارائه راه حلی برای بر طرف کردن مشکلات مختلف است،برنامه نویسی هنر برطرف کردن مشکلات با استفاده از کد است،ما می توانیم با استفاده از زبانهای برنامه نویسی برنامه بنویسیم،زبانهای برنامه نویسی زیادی وجود دارند که از این بین چندین زبان از بقیه مشهورتر می باشند،زبانهای برنامه نویسی به مرور زمان گسترش و تکامل می یابند. وقتی که می خواهید یک زبان برنامه نویسی را یاد بگیرید ابتدا باید یک زبان برنامه نویسی انتخاب کنید،اما کدام زبان برنامه نویسی را باید انتخاب کرد؟شاید یکی از بهترین زبانهای برنامه نویسی برنامه سی شارپ باشد. در این مقالات ما به شما نحوه برنامه نویسی به زبان سی شارپ را به صورت تصویری آموزش می دهیم،سعی کنید حتما بعد از خواندن مقالات آنها را به صورت عملی تمرین کنیدو اینکه قابلیت و مفهوم کدها را بفهمید نه آنها را حفظ کنید.

دات نت فریم ورک (.NET Framework) چیست؟

.NET Framework یک چارچوب است که توسط شرکت مایکروسافت برای توسعه انواع نرم افزارها علی الخصوص ویندوز طراحی شد.NET Framework. همچنین میتواند برای توسعه نرم افزارهای تحت وب مورد استفاده قرار بگیرد.تا کنون چندین نسخه از .NET Framework. انتشار یافته که هر بار قابلیتهای جدیدی به آن اضافه شده است.

.NET Framework. شامل کتابخانه کلاس محیط کاری (FCL) که در بر گیرنده کلاس ها، ساختارها،داده های شمارشی و... می باشد.مهمترین قسمت .NET Framework. زبان مشترک زمان اجرا (CLR) است که محیطی را فراهم می آورد که برنامه ها در آن اجرا شوند. این چارچوب ما را قادر می سازد که برنامه هایی که تحت آن نوشته شده اند اعم از C#.Net ، Visual Basic.Net و C++ را بهتر درک کنیم. کدهایی که تحت CLR و دات نت اجرا می شوند کدهای مدیریت شده نامیده می شوند چون CLRجنبه های مختلف نرم افزار را در زمان اجرا مدیریت می کند. در زمان کامپایل کدها به زبان مشترک میانی (CIL) که نزدیک و تقریبا شبیه به زبان اسمبلی است ترجمه می شوند. ما باید کدهایمان را به این زبان ترجمه کنیم چون فقط این زبان برای دات نت قابل فهم است. برای مثال کدهای C# و Visual Basic.Net هر دو به زبان مشترک میانی (CIL) ترجمه می شوند. به همین دلیل است که برنامه های مختلف در دات نت که با زبان های متفاوتی نوشته شده اند می توانند با هم ارتباط برقرار کنند. اگر یک زبان سازگار با دات نت می خواهید باید یک کامپایلر ایجاد کنید که کدهای شما را به زبان میانی ترجمه کند. کدهای ترجمه شده توسط CIL در یک فایل اسمبلی مانند exe. یا dll. ذخیره می شوند.کدهای ترجمه شده به زبان میانی به کامپایلر فقط در زمان (JIT) منتقل می شوند.این کامپایلر در لحظه فقط کدهایی را که برنامه در آن زمان نیاز دارد به زبان ماشین ترجمه می کند.

در زیر نحوه تبدیل کدهای سی شارپ به یک برنامه اجرایی به طور خلاصه آمده است :

1. **برنامه نویس برنامه خود را با یک زبان دات نت مانند سی شارپ می نویسد.**
2. **کدهای سی شارپ به کدهای معادل آن در زبان میانی تبدیل می شوند.**
3. **کدهای زبان میانی در یک فایل اسمبلی ذخیره می شوند.**
4. **وقتی کدها اجرا می شوند کامپایلر JIT کدهای زبان میانی را در لحظه به کدهایی که برای کامپیوتر قابل خواندن باشند تبدیل می کند.**

دات نت ویژگی دیگری به نام سیستم نوع مشترک (CTS) نیز دارد که بخشی از CLR است و نقشه ای است برای معادل سازی انواع داده ها در دات نت. با CTS نوع int در سی شارپ و نوع Integer در ویژوال بیسیک یکسان هستند چون هر دو از نوع System.Int32 مشتق می شوند. پاک کردن خانه های بلا استفاده حافظه در یک فایل(Garbage collection) یکی دیگر از ویژگیهای دات نت فریم ورک است. هنگامی که از منابعی، زیاد استفاده نشود دات نت فریم ورک حافظه استفاده شده توسط برنامه را آزاد می کند.

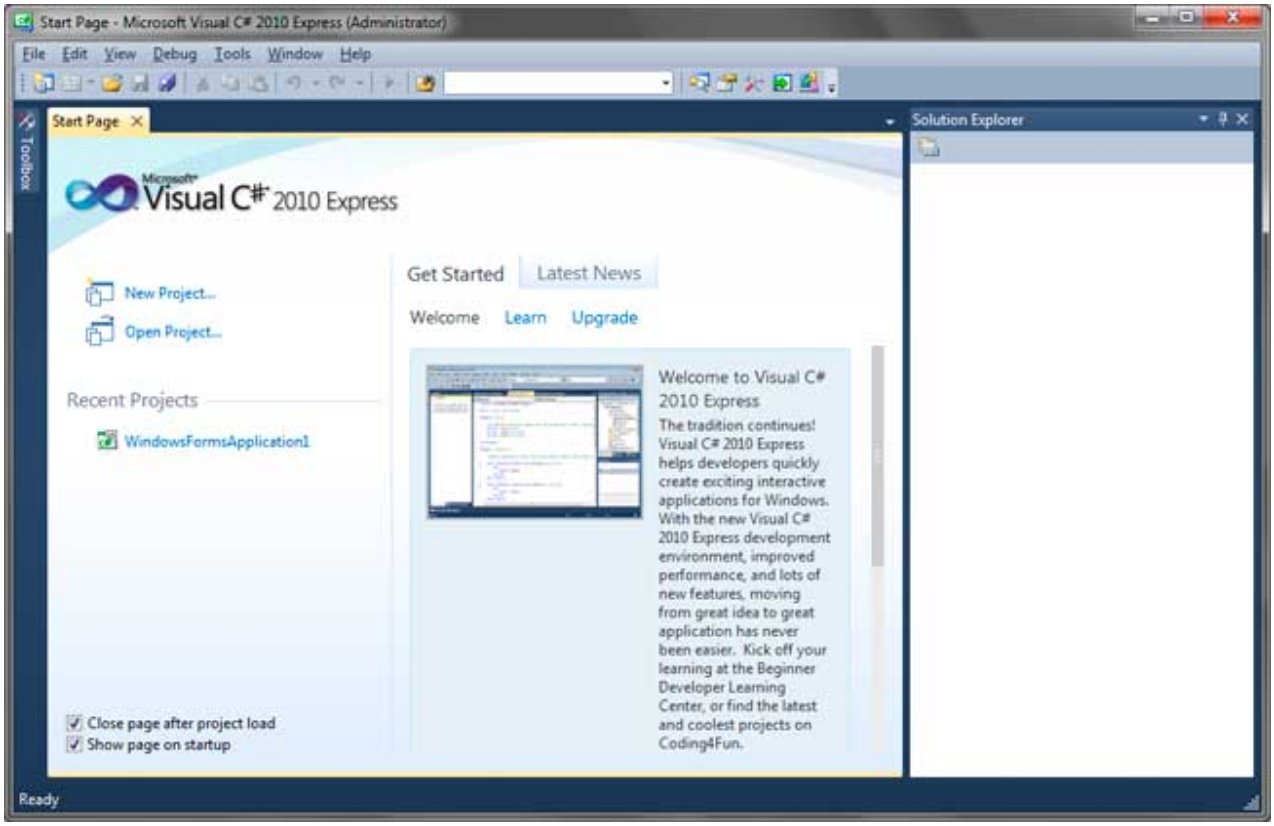
سی شارپ چیست؟

سی شارپ (C#) یک زبان برنامه نویسی شی گرا است که توسط شرکت مایکروسافت ساخته شده است.سی شارپ ترکیبی از قابلیتهای خوب C++ و Java است.اگر با این دو زبان آشنایی دارید این شانس را دارید زبان سی شارپ را راحت یاد بگیرید. این زبان به قدری راحت است که هم کسانی که قبلا برنامه نویسی نکرده اند و هم دانش آموزان می توانند راحت آن را یاد بگیرند. از سی شارپ می توان برای ساخت برنامه های تحت ویندوز ،تحت وب ،وب سرویس ها ،برنامه های موبایل و بازی ها استفاده کرد. می توان به جای واژه ویژوال سی شارپ از کلمه سی شارپ استفاده کرد اما ویژوال سی شارپ به معنای استفاده همزمان از سی شارپ و محیط گرافیکی ویژوال استودیو می باشد.زبان برنامه نویسی سی شارپ تنها زبانی است که مخصوصا برای دات نت فریم ورک طراحی شده است. سی شارپ از کتابخانه کلاس دات نت که شامل مجموعه بزرگی از اجزا از قبل ساخته شده است استفاده می کند.این اجزا به ساخت هر چه سریعتر برنامه ها کمک می کنند.سی شارپ یک برنامه بسیار قدرتمند و شی گرا است و با آن می توان برنامه هایی با قابلیت مدیریت بیشتر و درک آسان ایجاد کرد. ساختار این زبان نسبت به زبانهای دیگر بسیار آسان و قابل فهم است.

برای اجرای یک برنامه سی شارپ ابتدا باید دات نت فریم ورک نصب شود. سی شارپ یکی از زبانهایی است که از تکنولوژی های دیگر دات نت مانندASP.NET, Silverlight و XNA پشتیبانی میکند. همچنین یک محیط توسعه یکپارچه دارد که ان نیز به نوبه خود دارای ابزارهای مفیدی است که به شما در کدنویسی در سی شارپ کمک می کند. این زبان به طور دائم توسط مایکروسافت به روز شده و ویژگیهای جدیدی به آن اضافه می شود. سی شارپ یکی از بهترین زبانهای برنامه نویسی دات نت است.

ویژوال سی شارپ اکسپرس و ویژوال استودیو

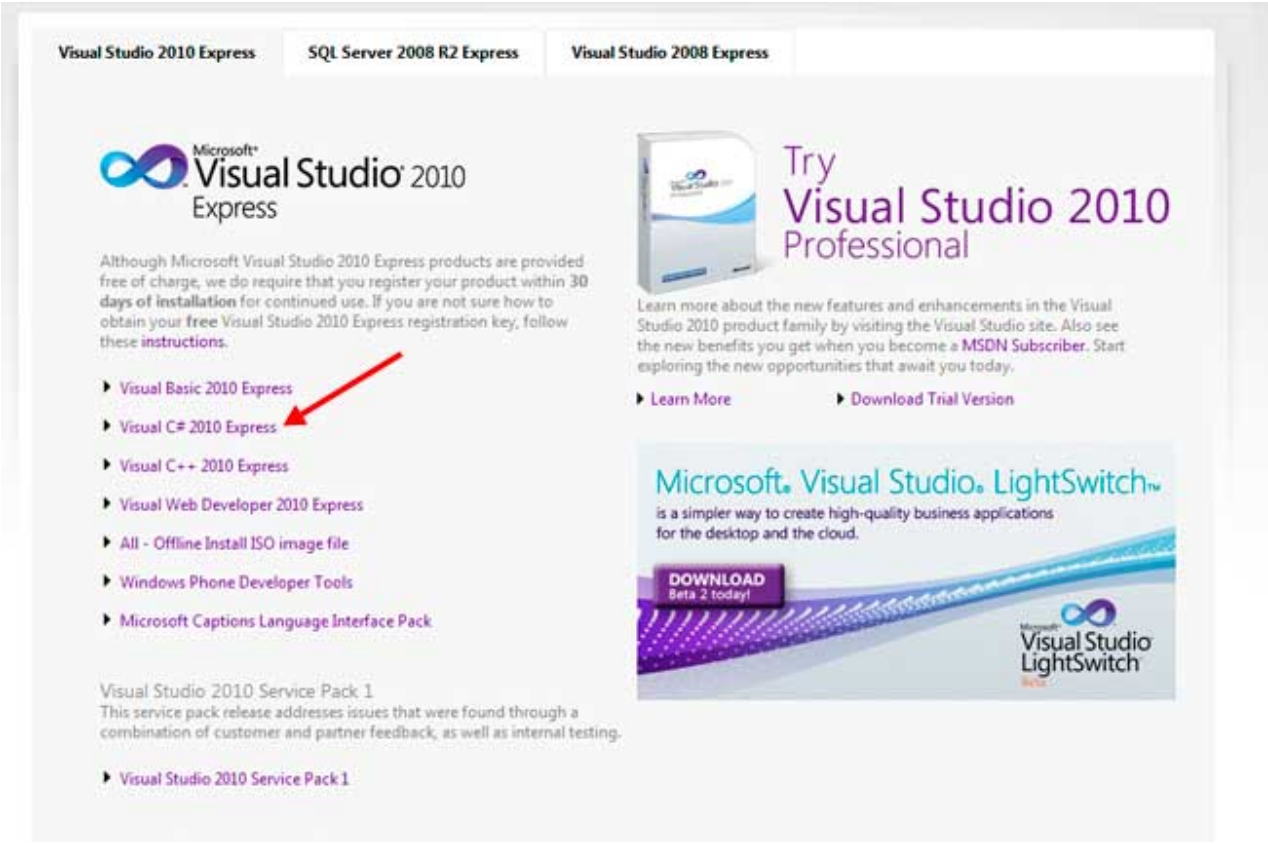
ویژوال استودیو 2010 و ویژوال سی شارپ 2010 محیط های توسعه یکپارچه ای هستند که دارای ابزارهایی برای کمک به شما برای توسعه برنامه های سی شارپ و دات نت می باشند.شما می توانید یک برنامه سی شارپ را با استفاده از برنامه notepad یا هر برنامه ویرایشگر متن دیگر بنویسید و با استفاده از کامپایلر سی شارپ از آن استفاده کنید، اما این کار بسیار سخت است چون اگر برنامه شما دارای خطا باشد خطایابی آن سخت می شود.توجه کنید که کلمه ویژوال استودیو هم به ویژوال استودیو و هم به ویژوال سی شارپ اشاره دارد.توصیه می کنیم که از محیط ویژوال استودیو برای ساخت برنامه استفاده کنید چون این محیط دارای ویژگی های زیادی برای کمک به شما جهت توسعه برنامه های سی شارپ می باشد.تعداد زیادی از پردازش ها که وقت شما را هدر می دهند به صورت خودکار توسط ویژوال استودیو انجام می شوند.یکی از این ویژگی ها اینتل لایسنس (Intellisense) است که شما را در تایپ سریع کدهایتان کمک می کند.یکی دیگر از ویژگیهای اضافه شده break point است که به شما اجازه می دهد در طول اجرای برنامه مقادیر موجود در متغیرها را چک کنید.ویژوال استودیو برنامه شما را خطایابی می کند و حتی خطاهای کوچک (مانند بزرگ یا کوچک نوشتن حروف) را برطرف می کند ،همچنین دارای ابزارهای طراحی برای ساخت یک رابط گرافیکی است که بدون ویژوال استودیو برای ساخت همچین رابط گرافیکی باید کدهای زیادی نوشت.با این برنامه های قدرتمند بازدهی شما افزایش می یابد و در وقت شما با وجود این ویژگیهای شگفت انگیز صرفه جویی می شود.ویژوال سی شارپ اکسپرس (Visual C# Express) آزاد است و می توان ان را دانلود و از آن استفاده کرد.این برنامه ویژگیهای کافی را برای شروع برنامه نویسی C# در اختیار شما قرار می دهد.این نسخه (نسخه اکسپرس) کامل نیست و خلاصه شده نسخه اصلی است، به هر حال استفاده از ویژوال سی شارپ اکسپرس برای انجام تمرینات این سایت کافی است :



ویژوال استودیو**(VS) دارای محیطی کاملتر و ابزارهای بیشتری جهت عیب یابی و رسم نمودارهای مختلف است که در ویژوال سی شارپ اکسپرس وجود ندارند. ویژوال استودیو فقط به سی شارپ خلاصه نمی شود و دارای زبانهای برنامه نویسی دیگری از جمله ویژوال بیسیک نیز می باشد.رابط کاربری سی شارپ و ویژوال استودیو بسیار شبیه هم است و ما در این کتاب بیشتر تمرینات را با استفاده از سی شارپ انجام می دهیم.**

دانلود کردن ویژوال سی شارپ اکسپرس

دانلود ویژوال سی شارپ اکسپرس آزاد است. بعد از ورود به سایت مایکروسافت (http://www.microsoft.com/express/Downloads/) شکل زیر را مشاهده می کنید.



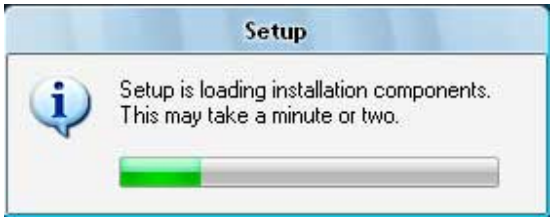
در این صفحه بر روی Visual C# Express 2010 کلیک کرده وسپس زبانتان را انتخاب می کنید.بعد از انتخاب زبان منتظر بمانید که برنامه به صورت خودکار دانلود شود. بعد از دانلود فایل آن را در مسیری دلخواه ذخیره کنید.

نصب ویژوال سی شارپ

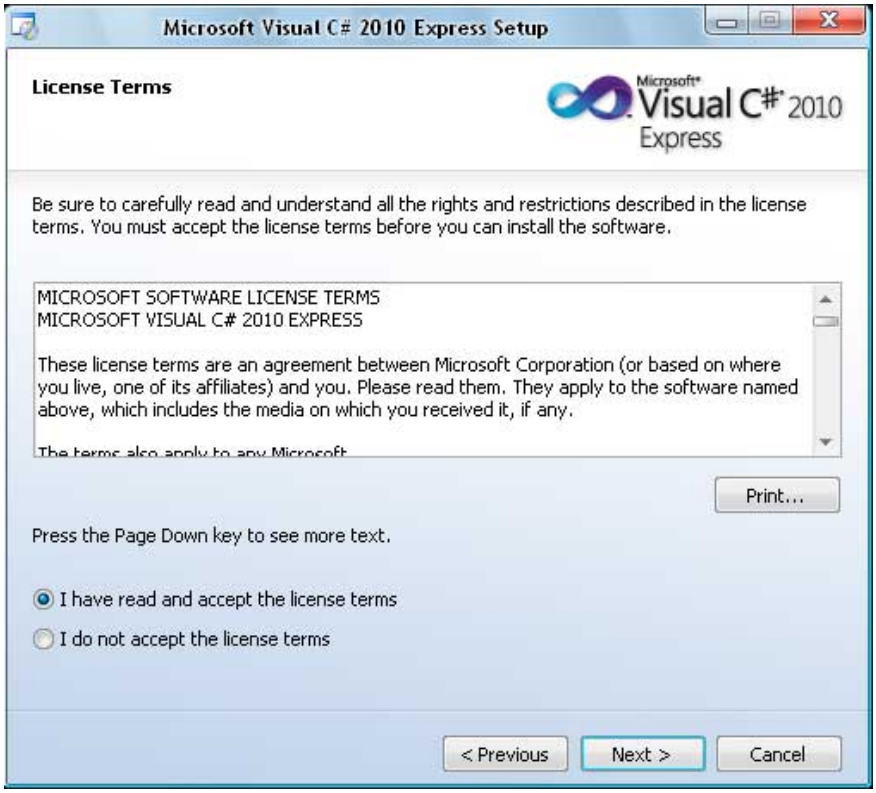
ابتدا نرم افزار ویژوال سی شارپ اکسپرس را دانلود کنید ولی به این نکته توجه کنید که سیستم شما برای نصب Visual C# Express باید دارای مشخصات زیر باشد :

سیستم عامل	Windows XP SP3 (All editions except Starter), Windows Vista SP2 (All editions except Starter), Windows 7
CPU	Computer with 1.6GHz or faster processor.
RAM	At least 1GB (32 bit) or 2GB(64 bit)
فضای هارد	At least 3GB of hard disk space.
کرات گرافیک	DirectX 9 capable video card. 1024×768 resolution or higher.

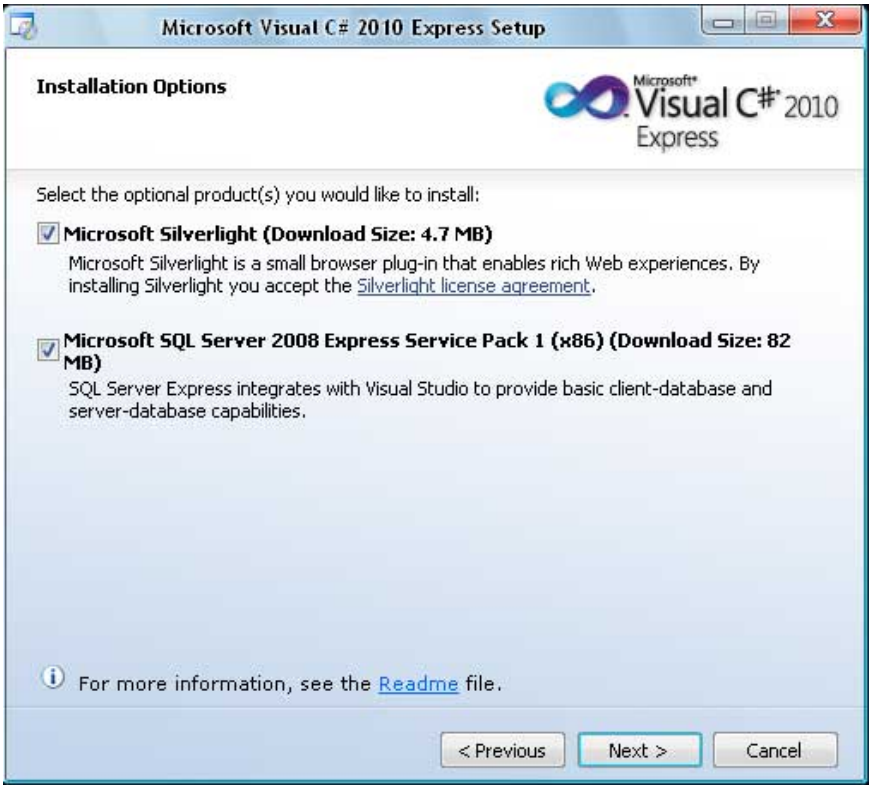
فایل دانلود شده را اجرا کنید. البته فراموش نشود که سیستمتان باید به اینترنت وصل باشد تا فایلی که دانلود کرده اید و در حکم یک نصب کننده می باشد اگر به فایلهای دیگری برای نصب نیاز داشت آنها را از اینترنت دانلود کند. حال روی فایل نصب کننده دو بار کلیک کنید تا شکل زیر ظاهر شود :



سپس یک صفحه خوش آمد گویی مشاهده خواهید کرد که دارای یک چک باکس (checkbox) است که آن را می توانید مانند شکل بالا تیک بزنید یا نزنید سپس بر روی دکمه Next کلیک می کنید.

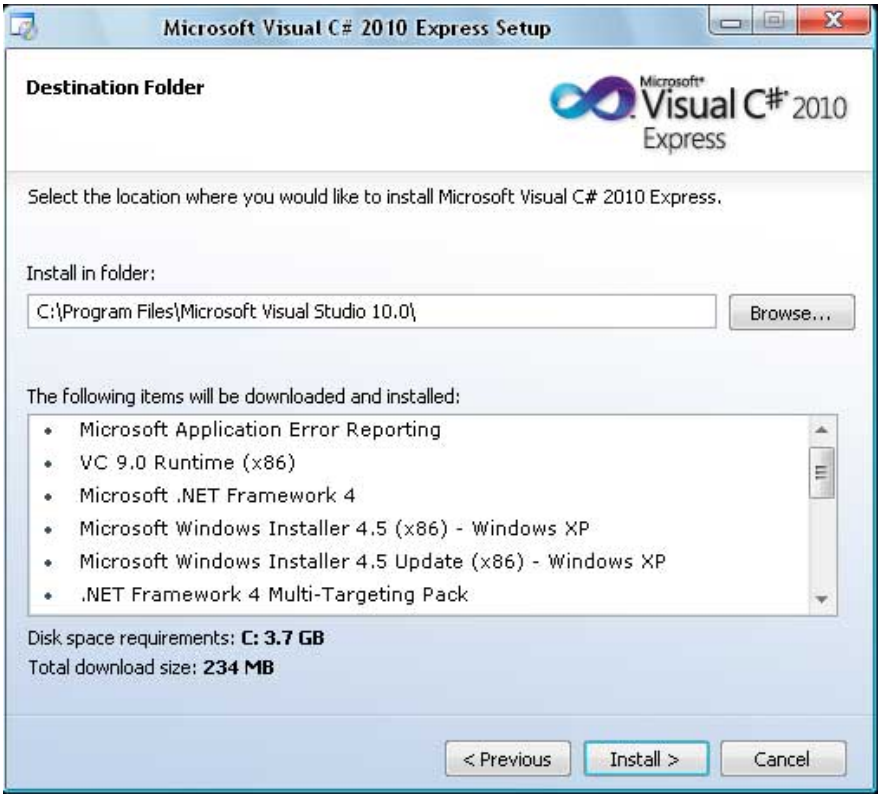


بعد از اینکه شکل بالا ظاهر شد گزینه بالا را علامت می زنید و سپس روی دکمه Next کلیک می کنید :

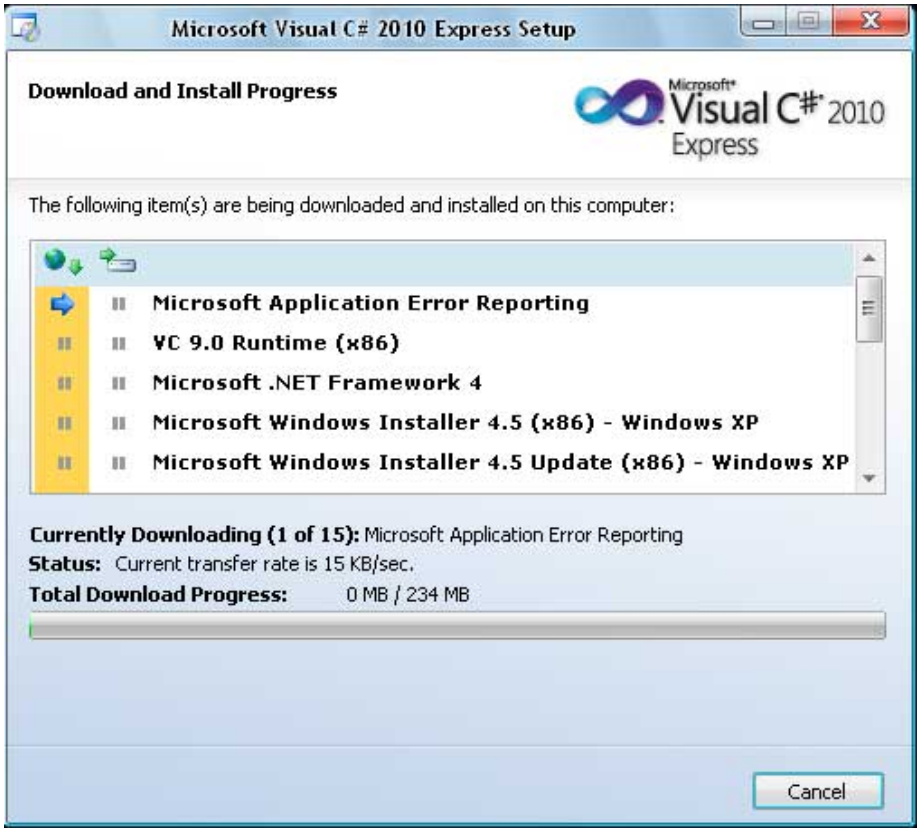


در مرحله بعد دو نرم افزار دیگر برای نصب به شما پیشنهاد می شود که اگر از قبل شما آنها را نصب کرده باشید این مرحله نمایش داده نمی شود.

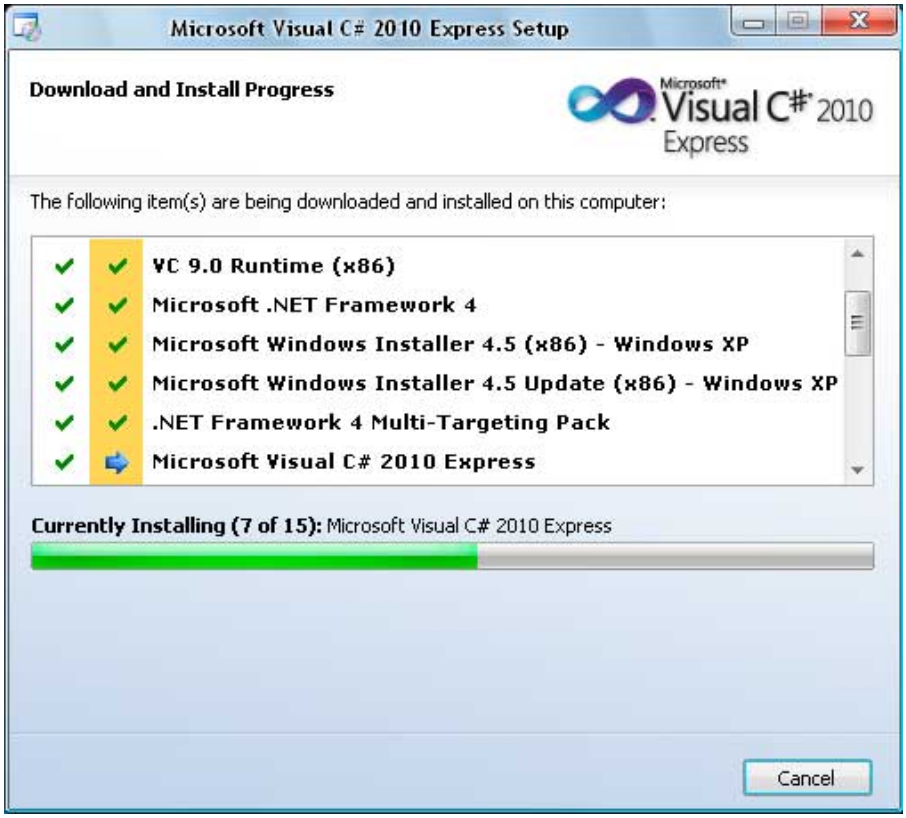
توصیه می شود که هر دو چک باکس مخصوصا Microsoft SQL Server Express را تیک بزنید که به شما اجازه می دهد بانک اطلاعاتی ایجاد کنید. اگر این گزینه را تیک بزنید حجم فایل دانلودی افزایش می یابد.سپس بر روی دکمه Next کلیک کنید.



در این مرحله مکانی که می خواهید برنامه در آن نصب شود از شما سوال می شود.پیشنهاد می کنیم که این مرحله را دستکاری نکنید. لیست برنامه هایی که قرار است نصب شوند در این مرحله نشان داده خواهد شد.تعداد این برنامه ها بستگی به این دارد که آیا قبلا در سیستم شما نصب شده باشند یا نه. مطمئن شوید که هنوز به اینرنت وصلید و سپس دکمه Install را فشار دهید.

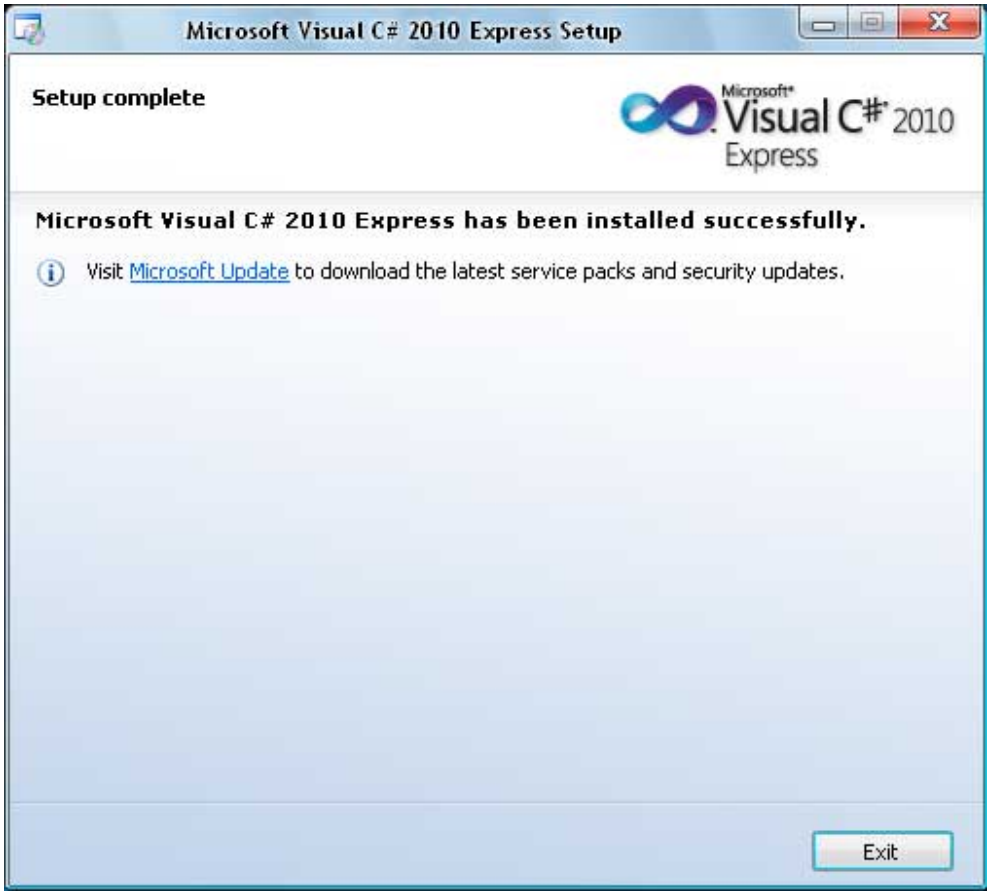


با کلیک بر روی install برنامه هایی که برای نصب لازم هستند از اینترنت دانلود می شوند.هنگامی که همه فایلها دانلود شدند می توانید اینترنت را قطع کنید.



برنامه ای که قبلا ذکر شد و به عنوان نصب کننده در ابتدا دانلود کردید پس از دانلود فایلهای لازم شروع به نصب آنها برای راه اندازی سی شارپ می کند.

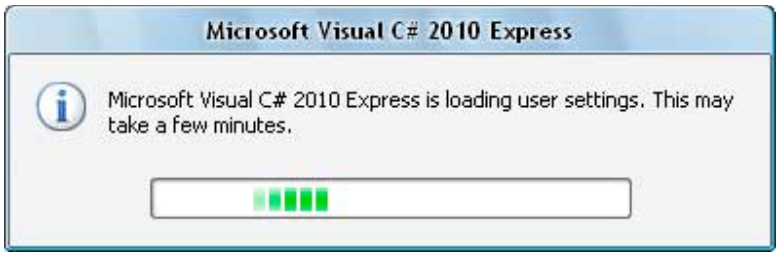
شما ممکن است در طول نصب برنامه مجبور به ریستارت کردن کامپیوترتان شوید که این به دلیل نصب دات نت 4 می باشد. پس از ریستارت، نصب برنامه به صورت اتوماتیک شروع می شود. صبر کنید تا نصب برنامه کامل شود.



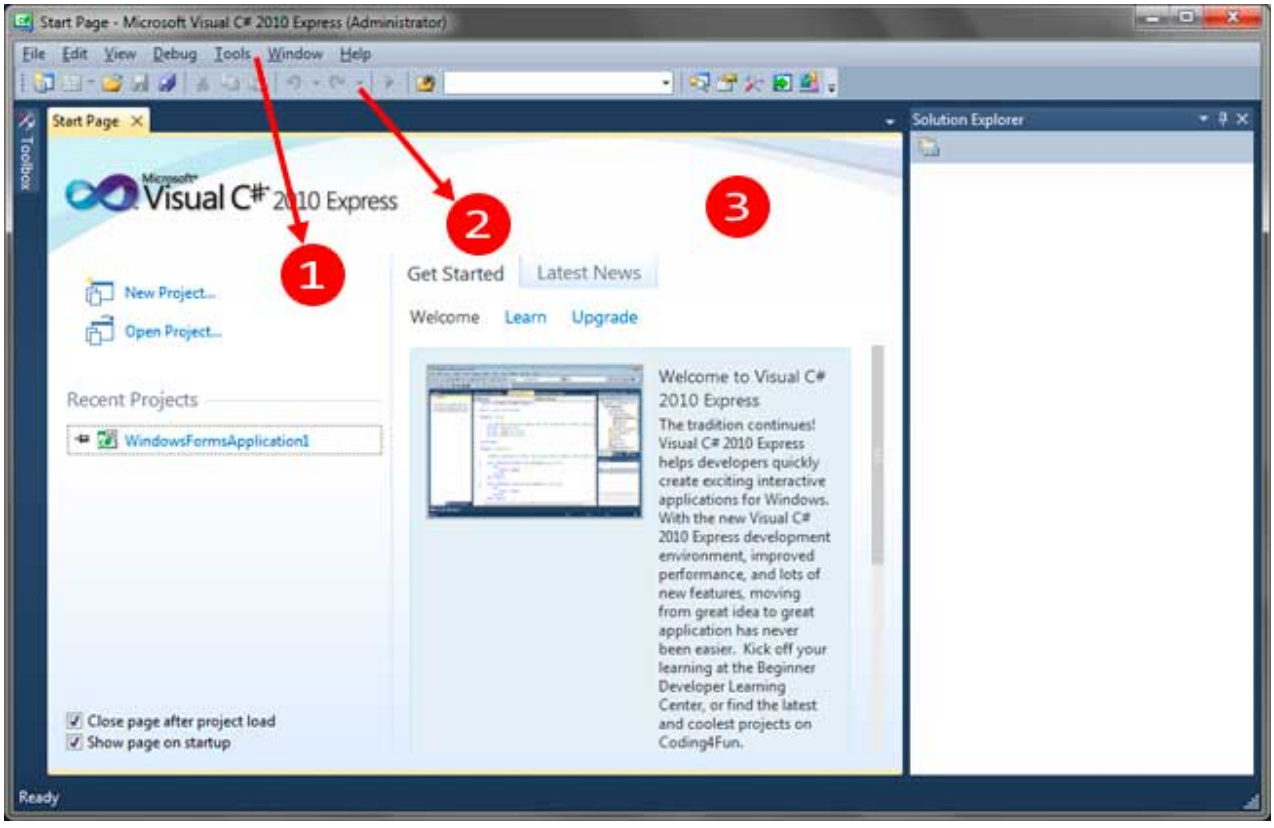
حال با نمایش پنجره فوق برنامه نصب شده است و شما می توانید با سی شارپ برنامه نویسی کنید.

به ویژوال سی شارپ 2010 خوش آمدید

در این بخش می خواهیم در باره قسمت های مختلف محیط ویژوال سی شارپ به شما مطالبی آموزش دهیم. لازم است که با انواع ابزارها و ویژگیهای این محیط آشنا شوید. برنامه ویژوال سی شارپ را اجرا کنید. اگر برای اولین بار است که برنامه را اجرا می کنید صفحه زیر برای شما نمایش داده خواهد شد، صبر کنید تا صفحه به صورت اتوماتیک بسته شود :



بعد از اینکه صفحه بالا بسته شد وارد صفحه آغازین ویژوال سی شارپ می شویم.



این صفحه بر طبق عناوین خاصی طبقه بندی شده که در مورد آنها توضیح خواهیم داد.

منو بار (Menu Bar)

منو بار (1) که شامل منوهای مختلفی برای ساخت، توسعه، نگهداری، خطایابی و اجای برنامه ها است.با کلیک بر روی هر منو دیگر منوهای وابسته به آن ظاهر می شوند.به این نکته توجه کنید که منوبار دارای آیتم های مختلفی است که فقط در شرایط خاصی ظاهر می شوند.به عنوان مثال آیتم های منوی Project در صورتی نشان داده خواهند شد که پروژه فعال باشد.

در زیر برخی از ویژگیهای منوها آمده است :

منو	توضیح
File	شامل دستوراتی برای ساخت پروژه یا فایل، باز کردن و ذخیره پروژه ها و خروج از آنها می باشد
Edit	شامل دستوراتی جهت ویرایش از قبیل کپی کردن، جایگزینی و پیدا کردن یک مورد خاص می باشد
View	به شما اجازه می دهد تا پنجره های بیشتری باز کرده و یا به آیتم های toolbar آیتمی اضافه کنید.
Project	شامل دستوراتی در مورد پروژه ای است که شما بر روی آن کار می کنید.
Debug	به شما اجازه کامپایل ، اشکال زدایی و اجرای برنامه را می دهد
Data	شامل دستوراتی برای اتصال به دیتابس ها می باشد.
Format	شامل دستوراتی جهت مرتب کردن اجزای گرافیکی در محیط گرافیکی برنامه می باشد.
Tools	شامل ابزارهای مختلف، تنظیمات و ...برای ویژوال سی شارپ و ویژوال استودیو می باشد.
Window	به شما اجازه تنظیمات ظاهری پنجره ها را می دهد.
Help	شامل اطلاعاتی در مورد برنامه ویژوال استودیو می باشد

The Toolbars

Toolbar (2) به طور معمول شامل همان دستوراتی است که در داخل منو ها قرار دارند.Toolbar همانند یک میانبر عمل می کند. هر دکمه در Toolbar دارای آیکونی است که کاربرد آنرا نشان می دهد. اگر در مورد عملکرد هر کدام از این دکمه ها شک داشتید می توانید با نشانگر موس بر روی آن مکث کوتاهی بکنید تا کاربرد آن به صورت یک پیام (tool tip) نشان داده شود.برخی از دستورات مخفی هستند و تحت شرایط خاص ظاهر می شوند.همچنین می توانید با کلیک راست بر روی منطقه خالی از toolbarو یا از مسیر View > Toolbarsدستورات بیشتری به آن اضافه کنید. برخی

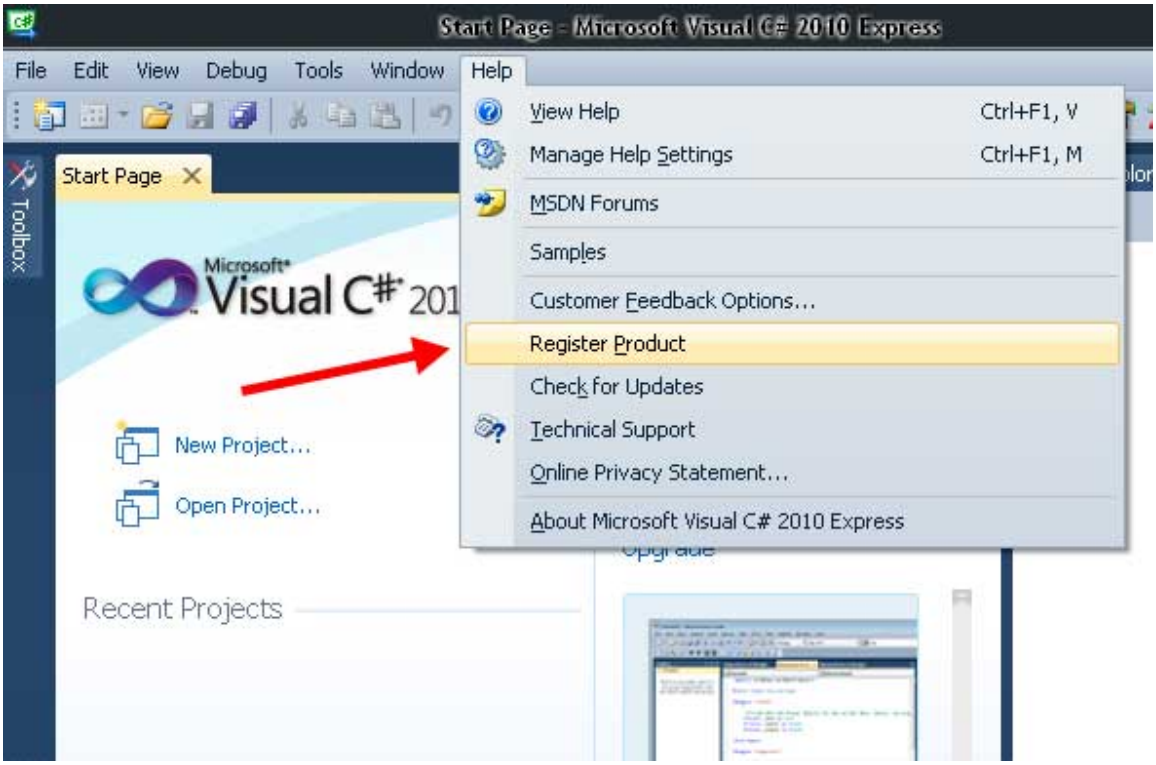
از دکمه ها دارای فلش های کوچکی هستند که با کلیک بر روی آنها دیگر دستورات وابسته به آنها ظاهر می شوند. سمت چپ هر toolbar به شما اجازه جا به جایی آن را می دهد.

صفحه آغازین (Start Page)

برای ایجاد یک پروژه و باز کردن آن از این قسمت استفاده می شود.همچنین اگر از قبل پروژه ای ایجاد کرده اید می توانید آن را در Recent Projects مشاهده و اجرا کنید. بخشهای مهم ویژوال سی شارپ توضیح داده شد در مورد بخشهای بعدی در درسهای آینده توضیحات بیشتری خواهیم داد.

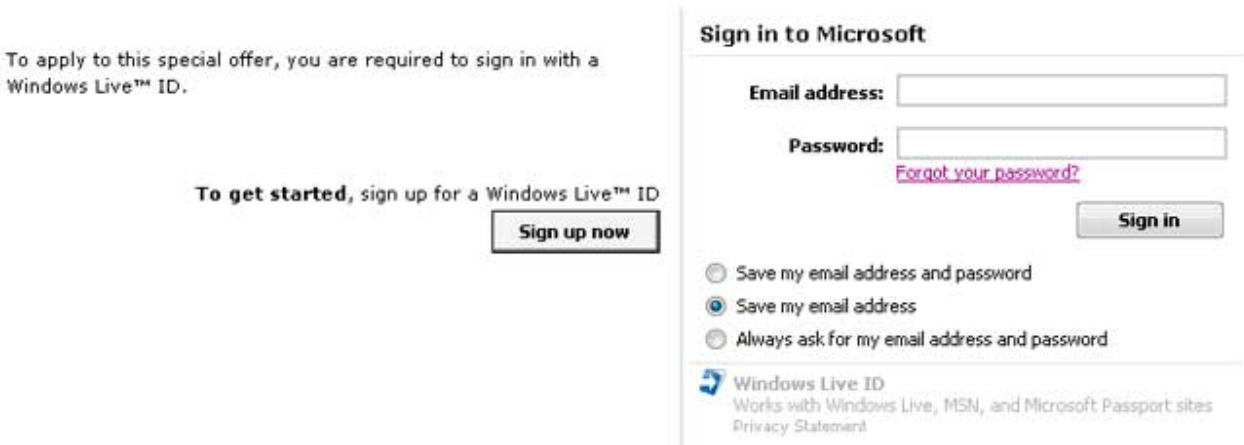
قانونی کردن ویژوال سی شارپ 2010

ویژوال سی شارپ اکسپرس 2010 رایگان است ولی شما برای استفاده دائمی از آن باید آن را قانونی کنید.برای این کار لازم است که یک کلید فعالسازی برای فعال کردن این نسخه کپی از شرکت مایکروسافت دریافت کنید. برنامه سی شارپ را باز کنید و از منوبار به مسیر زیر بروید : Help > Register Product



مشاهده می کنید که پنجره ای ظاهر می شود که از شما کلید فعالسازی را می خواهد.بر روی گزینه **Obtain a registration key online** کلیک کنید تا فرایند دریافت کلید فعالسازی شروع شود. توجه داشته باشید که برای این کار باید کامپیوترتان به اینترنت وصل باشد.

سپس یک پنجره ظاهر می شود که از شما آدرس ایمیلتان را می خواهد. اگر ایمیل دارید که باید وارد ایمیلتان شوید در غیر اینصورت باید یک ایمیل ایجاد کنید. برای ساخت ایمیل بر روی گزینه **Sign up now** کلیک کنید.



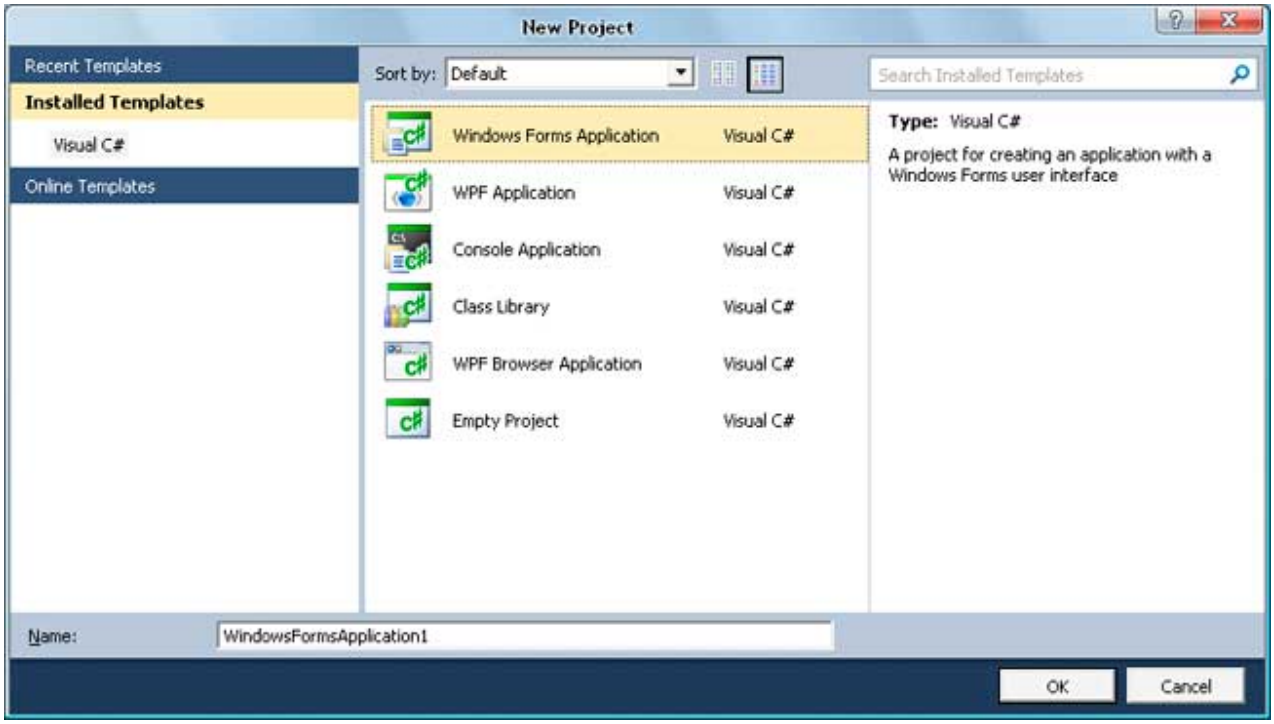
وقتی که وارد شدید مشاهده می کنید که از شما جزییاتی در مورد نام و ایمیل و محل سکونت از شما سوال می شود.بعد از جواب دادن به سوالات روی دکمه **Continue** کلیک کنید. حال صبر کنید تا کلید فعالسازی در مرورگر شما نمایش داده شود.

آنرا کپی کرده و به محیط ویژوال سی شارپ بر گردید و در تکست باکس (textBox) مربوطه paste کنیدو سپس دکمه Register Now را کلیک کنید.اکنون می توانید برای همیشه از ویژوال سی شارپ اکسپرس استفاده کنید.

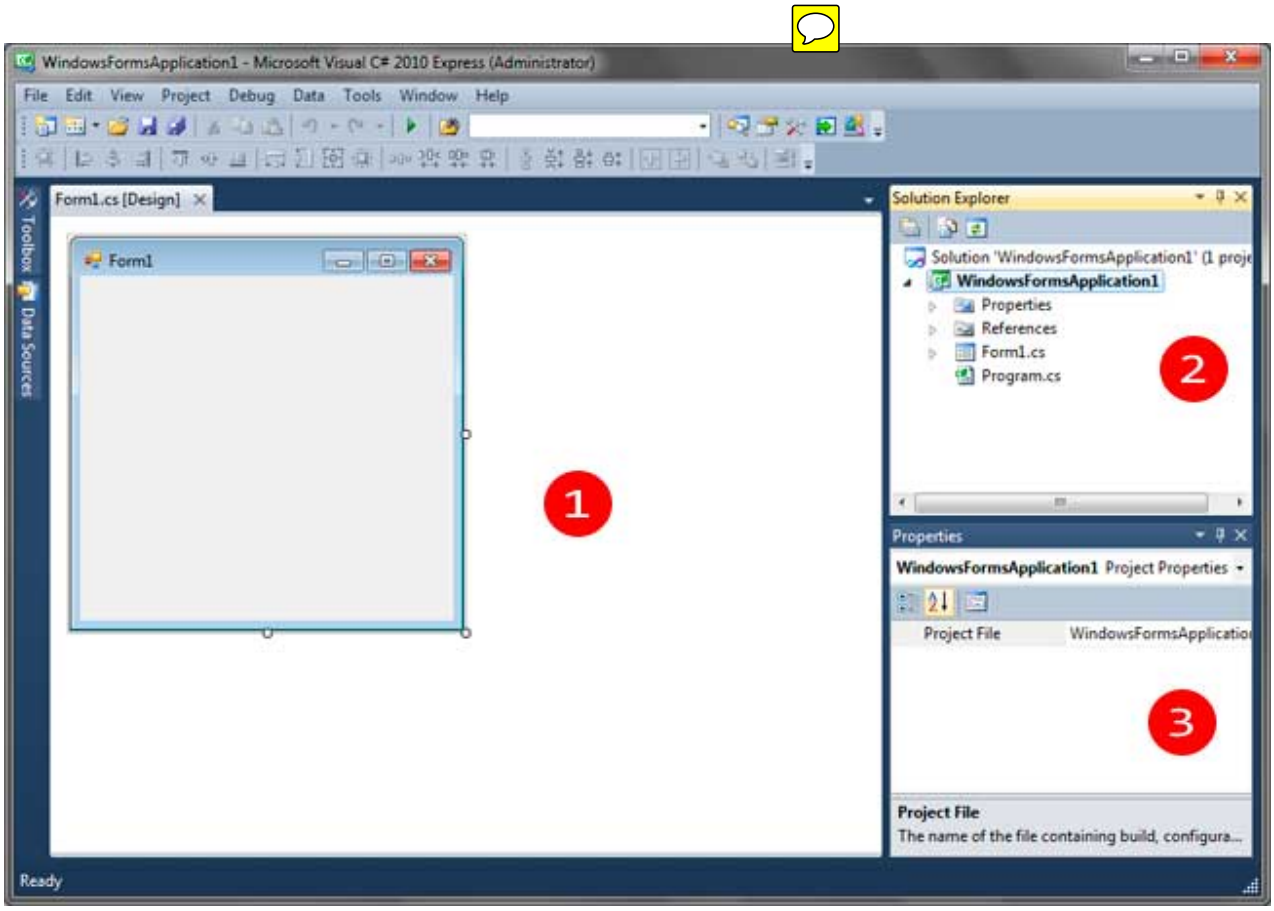
گردشی در ویژوال سی شارپ 2010

ویژوال سی شارپ اکسپرس از تعداد زیادی پنجره و منو تشکیل شده است که هر کدام برای انجام کار خاصی به کار می روند. اجازه دهید با نفوذ بیشتر در محیط ویژوال سی شارپ با این قسمتها آشنا شویم. از مسیر File > New Project یک پنجره فرم ایجاد کنید.

پنجره ای به شکل زیر نمایش داده خواهد شد.



همانطور که در شکل بالا نشان داده شده است گزینه **Windows Forms Application** و یک اسم برای پرژوه انتخاب می کنیم و بر روی دکمه **OK** کلیک می کنیم تا صفحه زیر نمایان شود :



مشخصات فرم بالا عبارت است از :

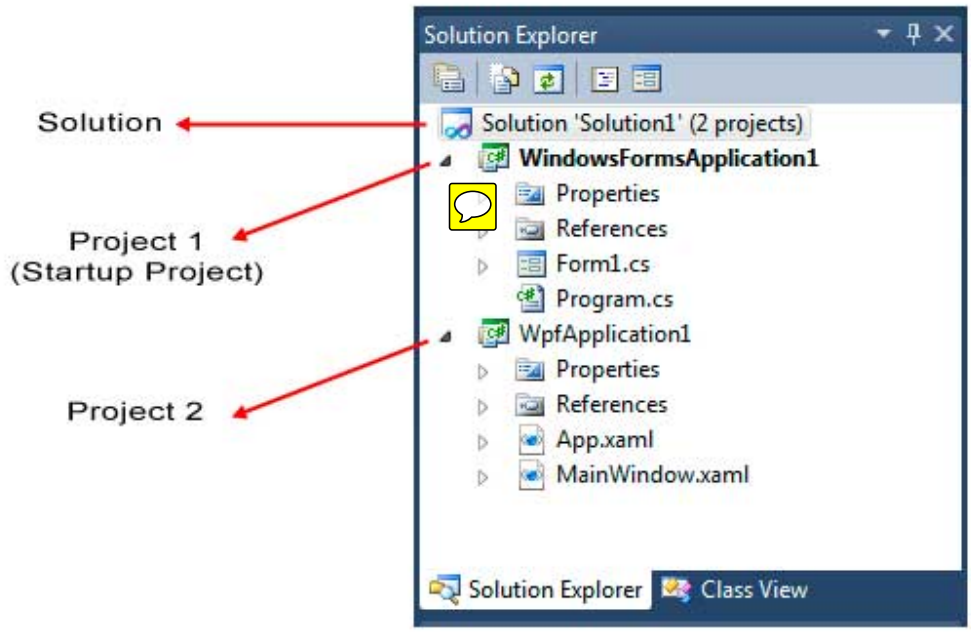
1- صفحه طراحی : این صفحه در حکم یک ناحیه برای طراحی فرم های ویندوزی شما است. فرم های ویندوزی رابطهای گرافیکی بین کاربر و کامپیوتر هستند و محیط ویندوز نمونه بارزی از یک رابط گرافیکی یا **GUI** است. شما در این صفحه می توانید کنترلهایی مانند دکمه ها ، برچسب ها و ....به فرمتان اضافه کنید. جزییات بیشتر در مورد فرمهای ویندوزی و کنترلها و برنامه نویسی شی گرا در فصل فرم های ویندوزی آمده است. اما توصیه می شود ابتدا مبانی برنامه نویسی را مطالعه کنید.

2- مرورگر پروژه **(Solution Explorer)** : **Solution Explorer** پروژه و فایلهای مربوط به آن را نشان می دهد. یک **Solution** برنامه ای که توسط شما ساخته شده است را نشان می دهد.ممکن است این برنامه یک پروژه ساده یا یک پروژه چند بخشی باشد. اگر **Solution**

Explorer در صفحه شما نمایش داده نمی شود می توانید از مسیر Solution Explorer > Other Windows > View و یا با کلیدهای میانبر Ctrl+Alt+L آنرا نمایان کنید.

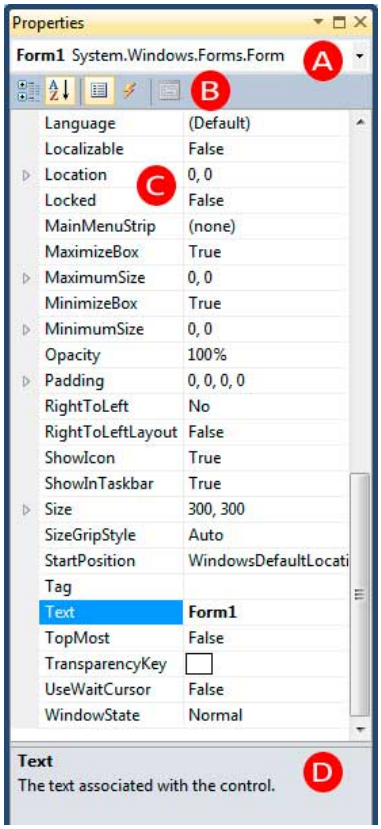
اگر چندین پروژه در حال اجرا هستند پروژه ای که با خط برجسته (Bold) نشان داده شده پروژه فعال می باشد و هنگام اجرای برنامه اجرا می شود.

اگر بخواهید پروژه ای را که فعال نیست اجرا کنید، بر روی Solution Explorer کلیک راست کنید و سپس گزینه Set as StartUp Project را انتخاب نمایید.Solution Explorer زیر یک Solution با 2 پروژه را نشان می دهد. هر پروژه شامل فایلها و فولدرهای مربوط به خود است.



دکمه اول از سمت چپ در شکل بالا اجازه ویرایش خواص پروژه را به شما می دهد.برخی فایلهای مربوط به پروژه در Solution Explorer مخفی هستند.می توانید با کلیک بر روی دکمه دوم آنها را نمایان نمایید.دکمه سوم Solution Explorer را بعد از حذف و اضافه کردن فایلها بروز می کند.بقیه دکمه ها بسته به نوع پروژه نمایان شده و کار می کنند. با دو بار کلیک بر روی هر کدام از فایلها در Solution Explorer می توان محتویات آنها را مشاهده کرد.

3- پنجره خواص (Properties)

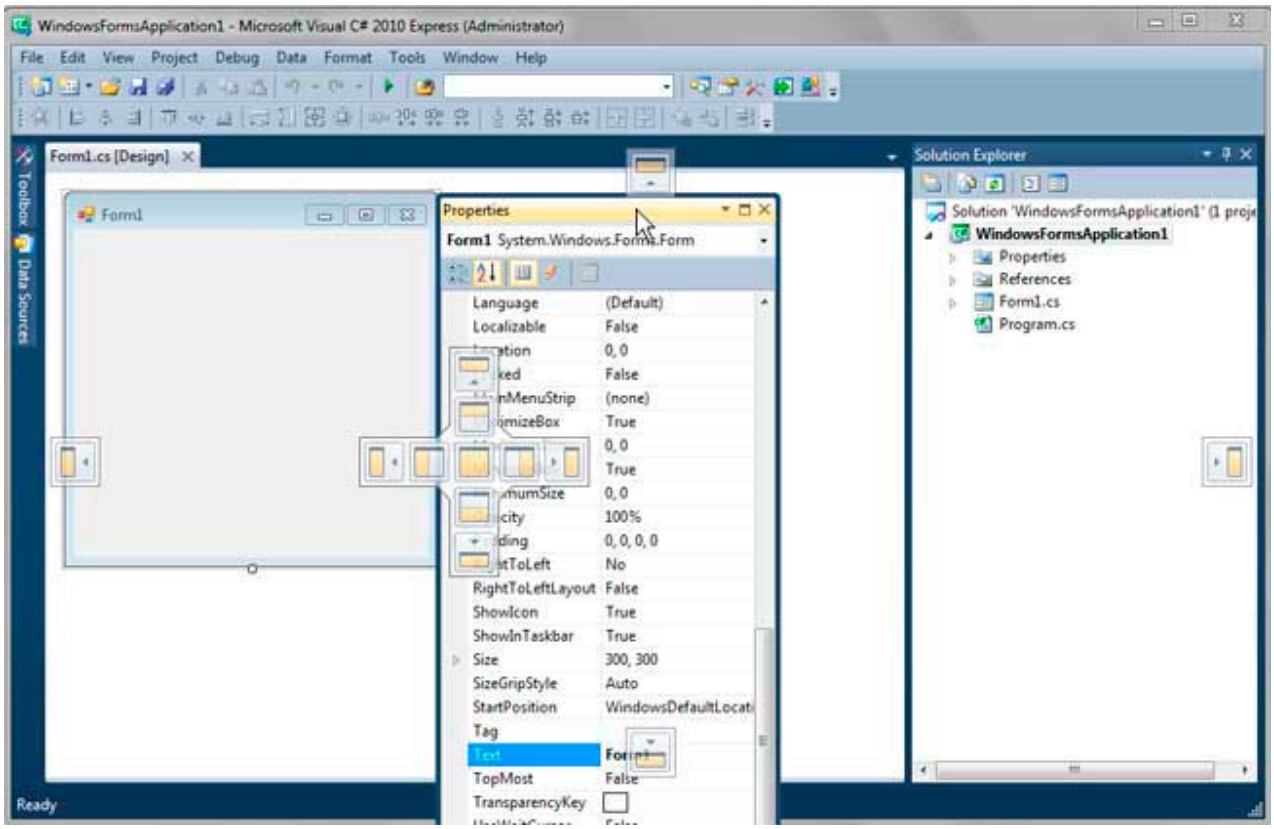


پنجره خواص **(Properties)** خواص و رویدادهای مختلف هر آیتم انتخاب شده اعم از فرم ، فایل ، پروژه و کنترل را نشان می دهد. اگر این پنجره مخفی است می توانید از مسیر **View > Other Windows > Properties Window** یا کلید میانبر **F4** آنرا ظاهر کنید. در مورد خواص در درسهای آینده مفصل توضیح خواهیم داد. خاصیت ها ، ویژگیها و صفات اشیا را نشان می دهند. به عنوان مثال یک ماشین دارای خواصی مانند رنگ ، سرعت، اندازه و مدل است. اگر یک فرم یا کنترل را در صفحه طراحی و یا یک پروژه یا فایل را در **SolutionExplorer** انتخاب کنید پنجره خواص مربوط به آنها نمایش داده خواهد شد. این پنجره همچنین دارای رویدادهای مربوط به فرم یا کنترل انتخاب شده می باشد. یک رویداد **(event)** اتفاقی است که در شرایط خاصی پیش می آید مانند وقتی که بر روی دکمه **(button)** کلیک و یا متنی را در داخل جعبه متن **(text box)** اصلاح می کنیم. کمبو باکس **(combo box)** شکل بالا که با حرف **A** نشان داده شده است به شما اجازه می دهد که شی مورد نظرتان (دکمه، فرم و...)را که می خواهید خواص آنرا تغییر دهید انتخاب کنید. این کار زمانی مفید است که کنترلهای روی فرم بسیار کوچک یا به هم نزدیک بوده و انتخاب آنها سخت باشد. در زیر کمبو باکس بالا دکمه های مفیدی قرار دارند **(B)**. برخی از این دکمه ها در شرایط خاصی فعال می شوند.دکمه اول خاصیت اشیا را بر اساس دسته های مختلفی مرتب می کند. دومین دکمه خواص را بر اساس حروف الفبا مرتب می کند که پیشنهاد می کنیم از این دکمه برای دسترسی سریع به خاصیت مورد نظرتان استفاده کنید. سومین دکمه هم وقتی ظاهر می شود که یک کنترل یا یک فرم را در محیط طراحی انتخاب کنیم. این دکمه به شما اجازه دسترسی به خواص فرم ویا کنترل انتخاب شده را می دهد. چهارمین دکمه (که به شکل یک رعد و برق نمایش داده شده) رویدادهای فرم ویا کنترل انتخاب شده را می دهد. در پایین شکل بالا توضیحات کوتاهی در مورد خاصیت ها و رویداد ها نشان داده می شود. بخش اصلی پنجره خواص **(C)**شامل خواص و رویدادها است.در ستون سمت چپ نام رویداد یا خاصیت و در ستون سمت راست مقدار آنها آمده است. در پایین پنجره خواص جعبه توضیحات **(D)** قرار دارد که توضیحاتی درباره خواص و رویدادها در آن نمایش داده می شود.

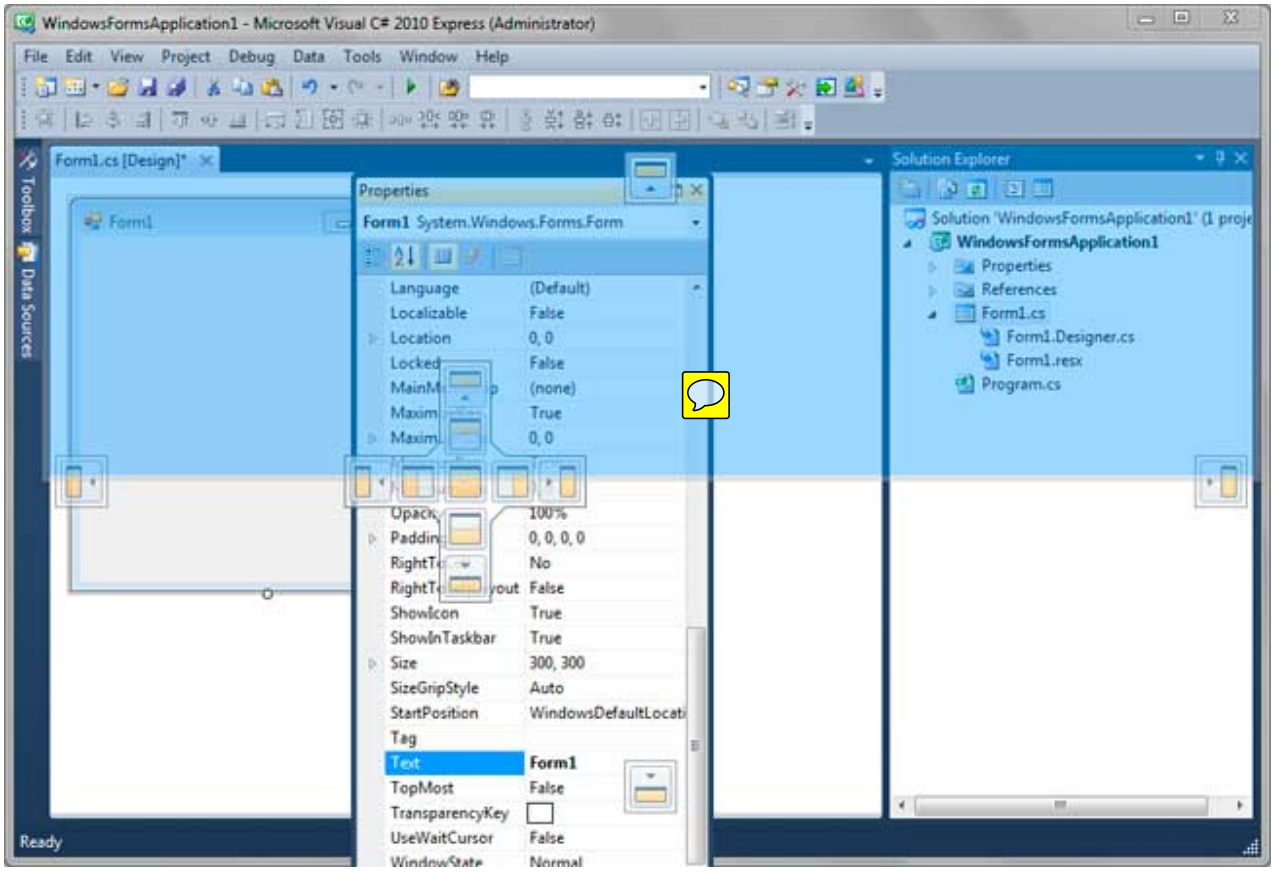
تغییر ظاهر ویژوال سی شارپ اکسپرس 2010

اگر موقعیت پنجره ها و یا ظاهر برنامه ویژوال سی شارپ را دوست نداشته باشید، می توانید به دلخواه آن را تغییر دهید.

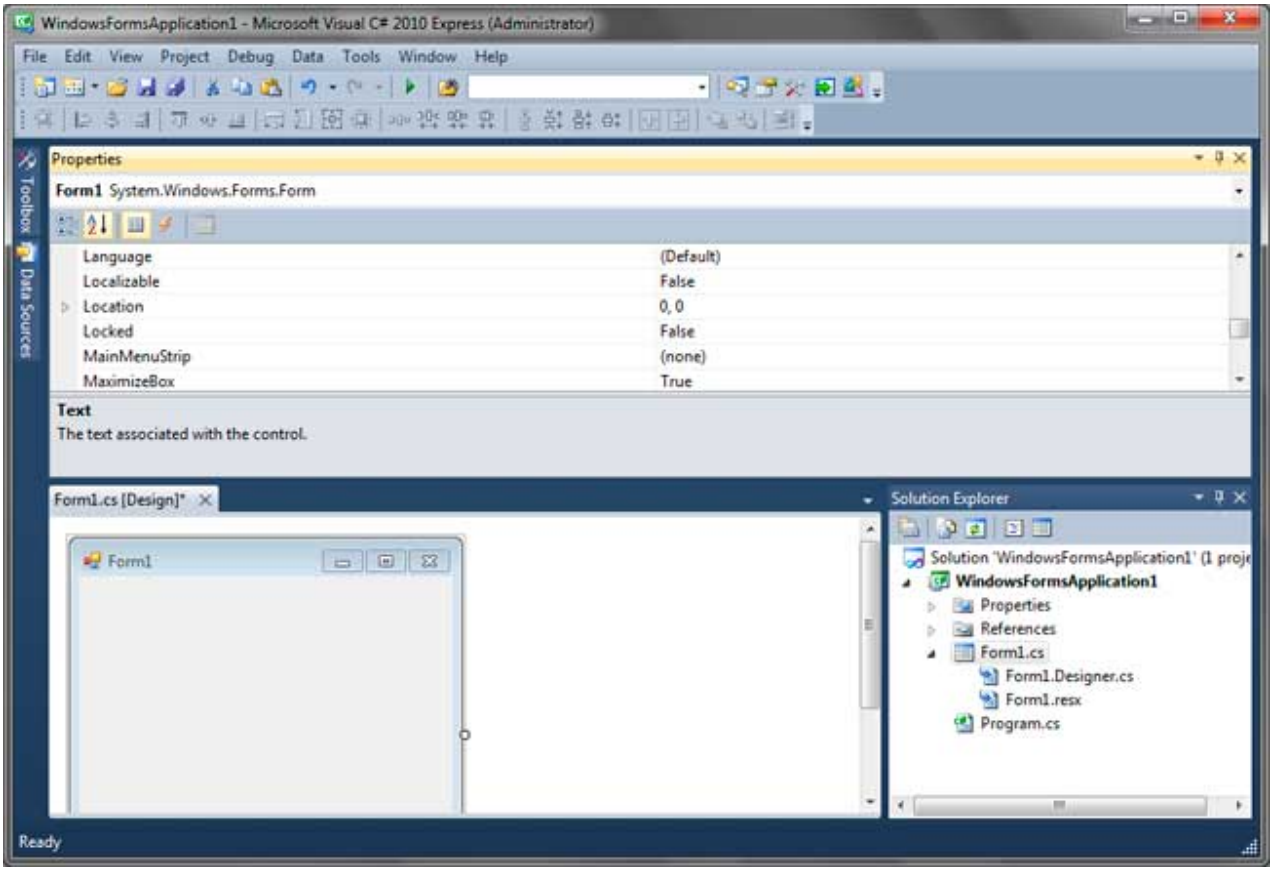
برای این کار بر روی نوار عنوان (title bar) کلیک کرده و آنرا می کشید تا پنجره به شکل زیر به حالت شناور در آید.



در حالی که هنوز بر روی پنجره کلیک کرده اید و آن را می کشید یک راهنما (فلشی با چهار جهت) ظاهر می شود و شما را در قرار دادن پنجره در محل دلخواه کمک می کند. به عنوان مثال شما می توانید پنجره را در بالاترین قسمت محیط برنامه قرار دهید.منطقه ای که پنجره قرار است در آنجا قرار بگیرد به رنگ آبی در می آید.

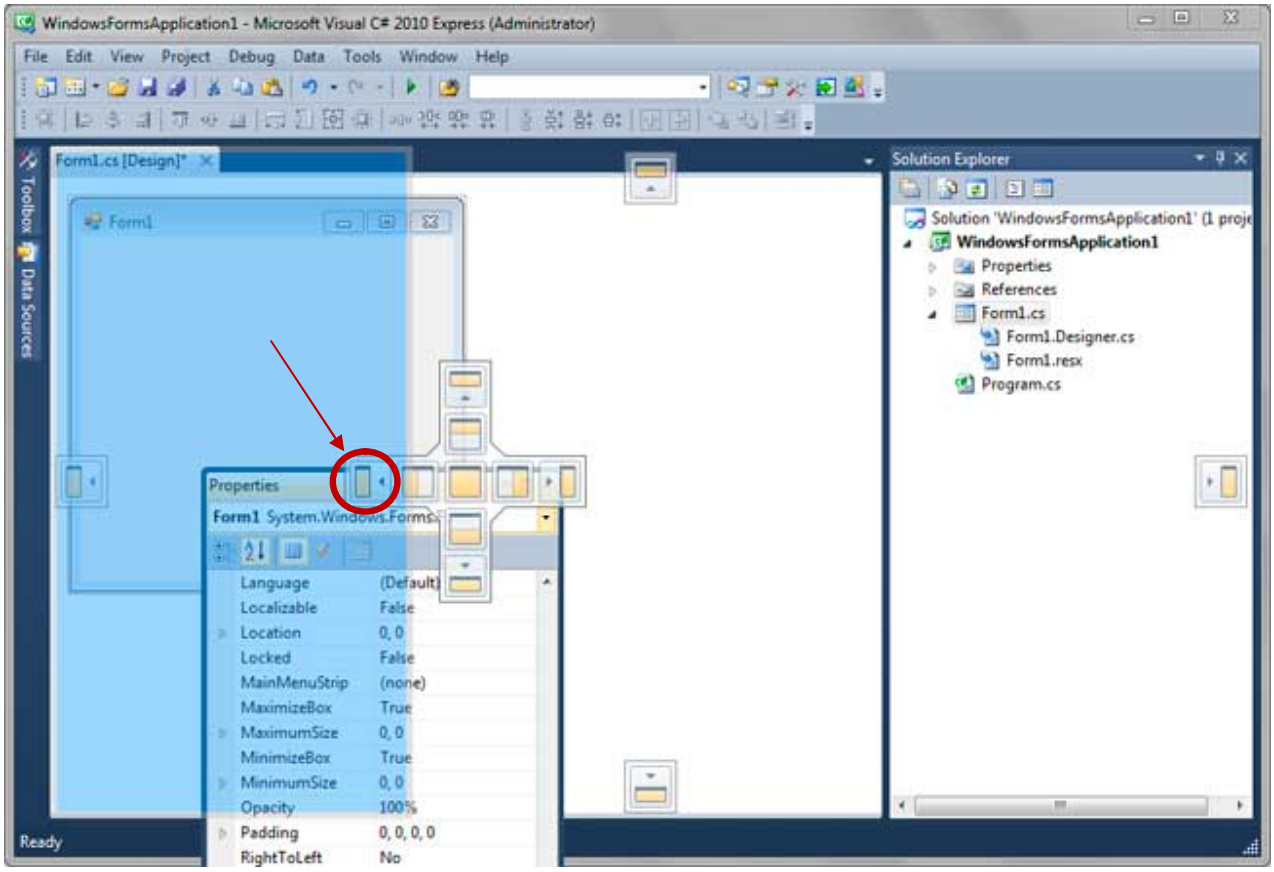


منطقه ای که پنجره قرار است در آنجا قرار بگیرد به رنگ آبی در می آید.

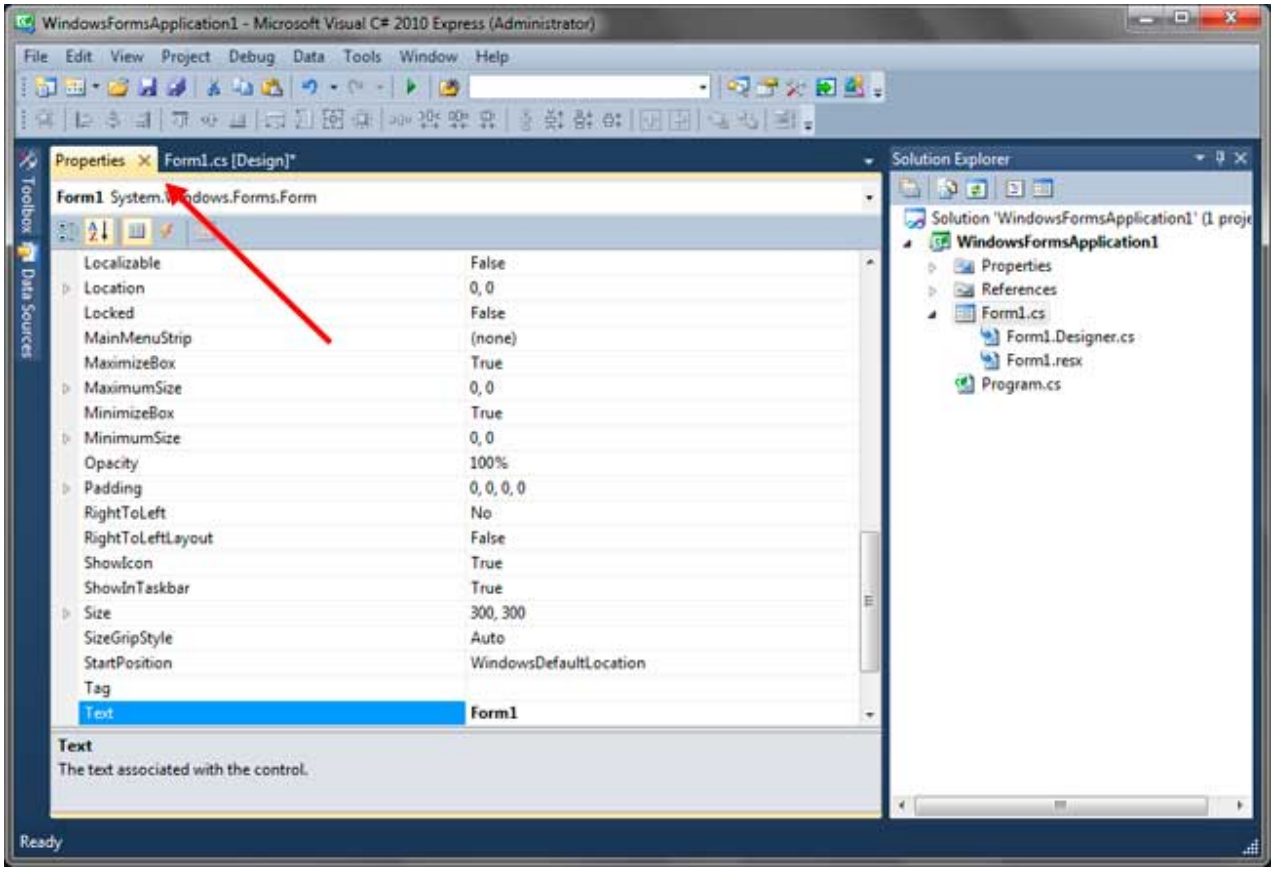


پنجره در قسمت بالای محیط قرار داده شده است

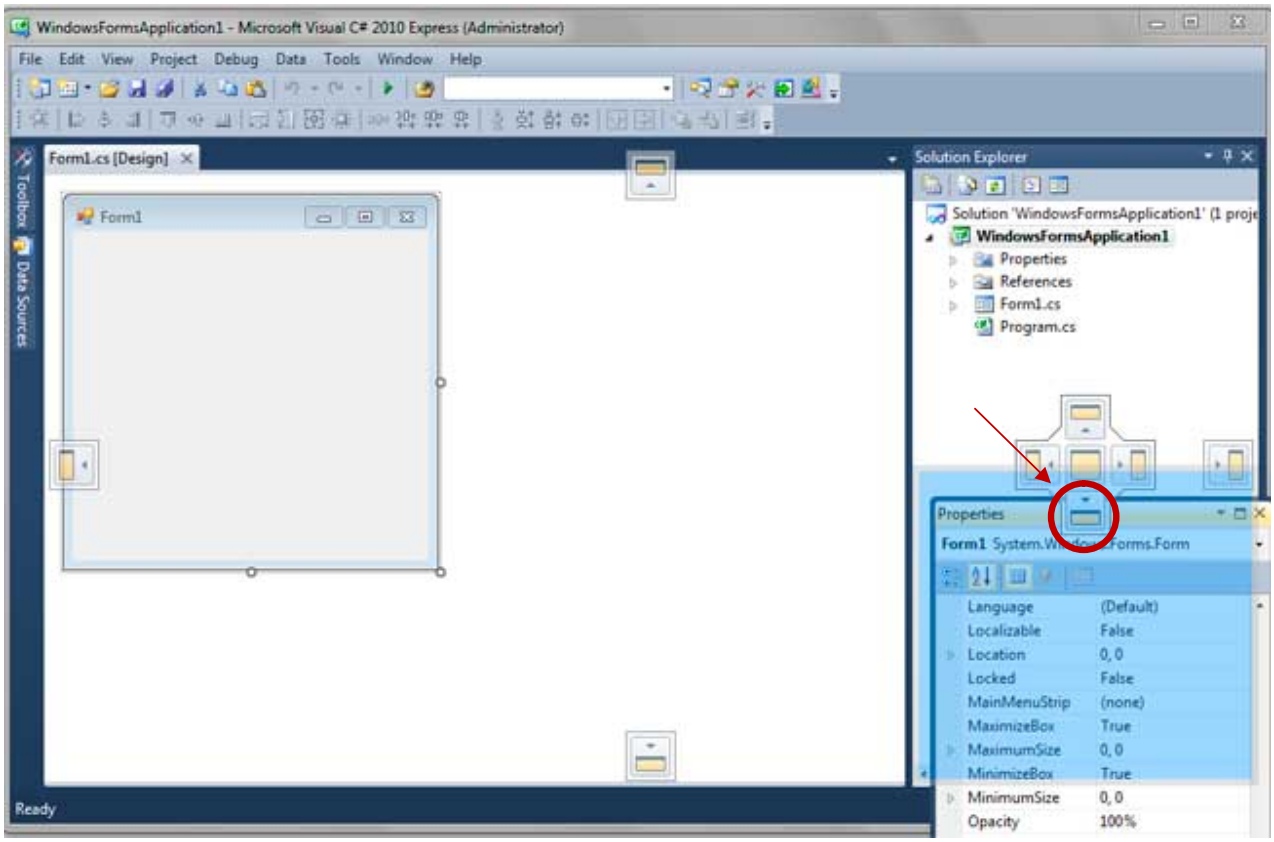
راهنمای صلیب شکل حاوی جعبه های مختلفی است که به شما اجازه می دهد پنجره انتخاب شده را در محل دلخواه محیط ویژوال سی شارپ قرار دهید. به عنوان مثال پنجره Properties را انتخاب کنید و آنرا به چپ ترین قسمت صلیب در پنجره نمایش داده شده نزدیک و رها کنید، مشاهده می کنید که پنجره مذکور در سمت چپ پنجره Design View قرار می گیرد.



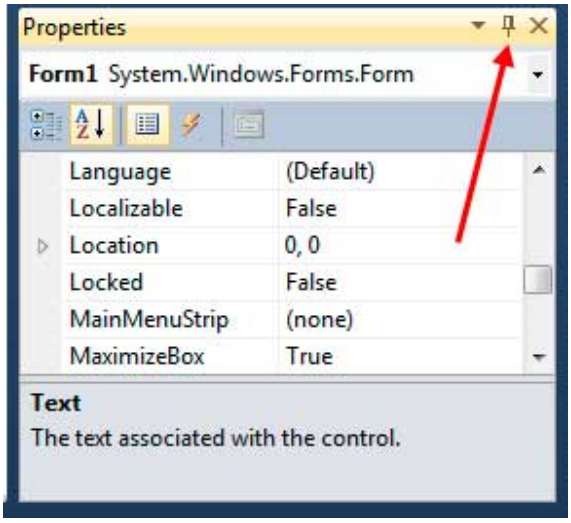
کشیدن پنجره به مرکز صلیب راهنما باعث ترکیب آن با پنجره مقصد می شود که در مثال بالا شما به توانید به عنوان یک تب به پنجره **Properties** دست پیدا کنید.



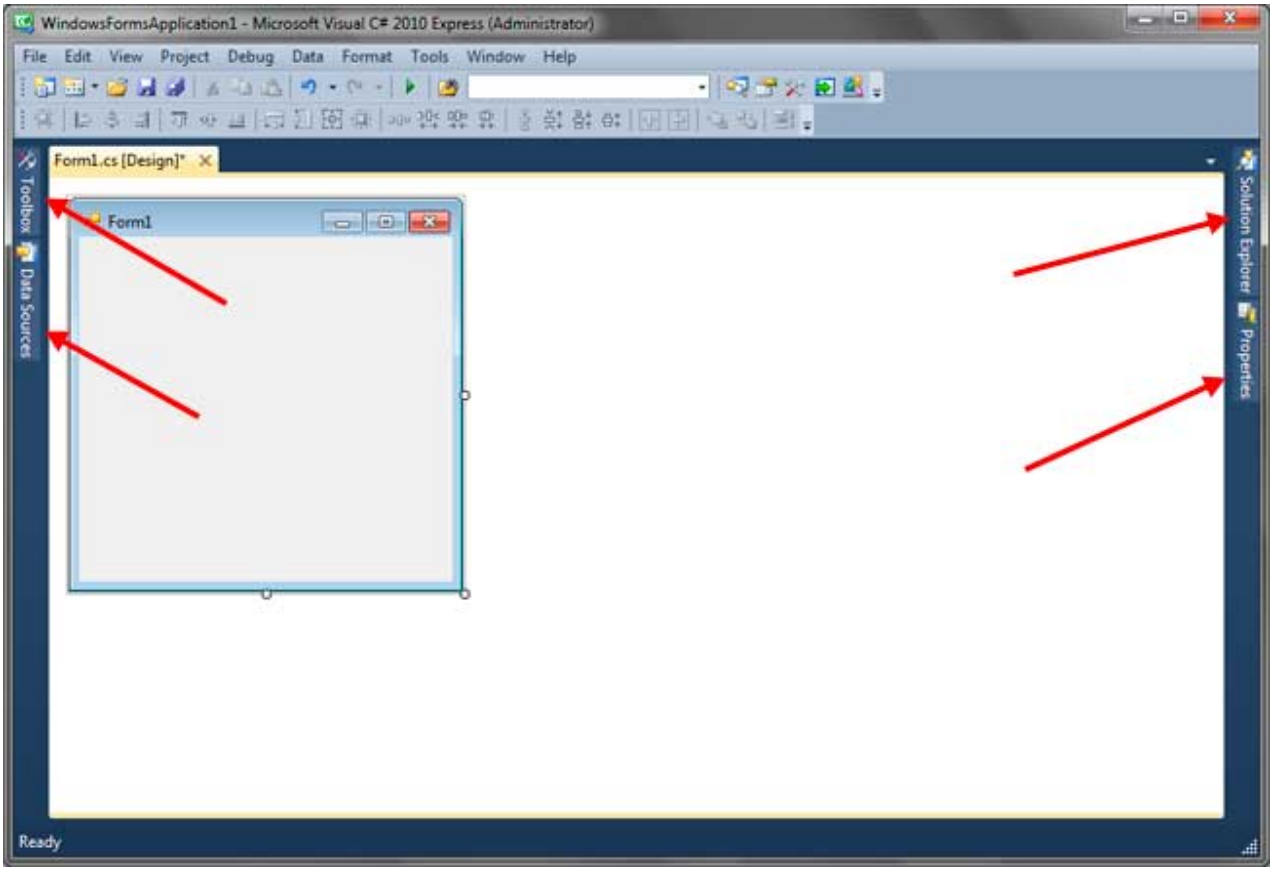
اگربه عنوان مثال پنجره Properties را روی پنجره Solution Explorer بکشید، یک صلیب راهنمای دیگر نشان داده می شود، با کشیدن پنجره به قسمت پایینی صلیب پنجره Properties زیر پنجره Solution Explorer قرار خواهد گرفت.



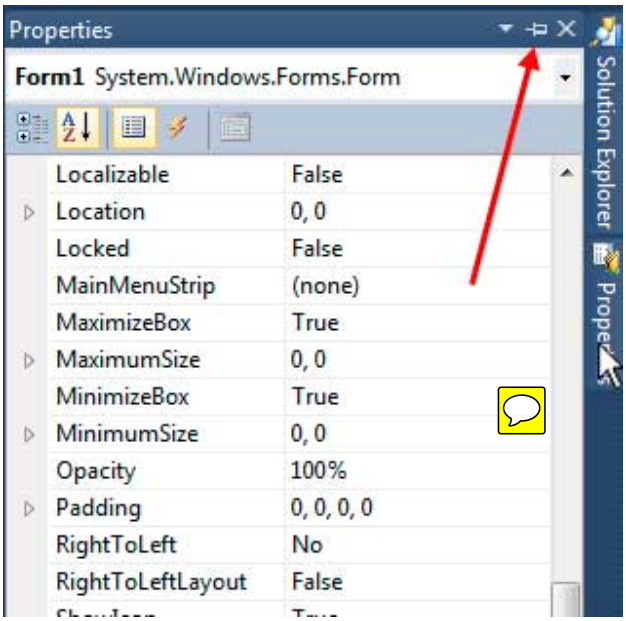
قسمتی از محیط برنامه که می خواهید پنجره در آنجا قرار بگیرد به رنگ آبی در می آید. ویژوال سی شارپ همچنین دارای خصوصیتی به نام **autohide** است که به صورت اتوماتیک پنجره ها را مخفی می کند. هر پنجره دارای یک آیکون سنجاق مانند نزدیک دکمه **close** می باشد.



بر روی این آیکون کلیک کنید تا ویژگی **auto-hide** فعال شود. برای دسترسی به هر یک از پنجره ها می توان با موس بر روی آنها توقف یا بر روی تب های کنار محیط ویژوال سی شارپ کلیک کرد.



برای غیر فعال کردن این ویژگی در هر کدام از پنجره ها کافیست پنجره را انتخاب کرده و دوباره بر روی آیکون مورد نظر کلیک کنید.



به این نکته توجه کنید که اگر شکل آیکون افقی بود بدین معناست که ویژگی فعال و اگر شکل آن عمودی بود به معنای غیر فعال بود ویژگی **auto-hide** می باشد.

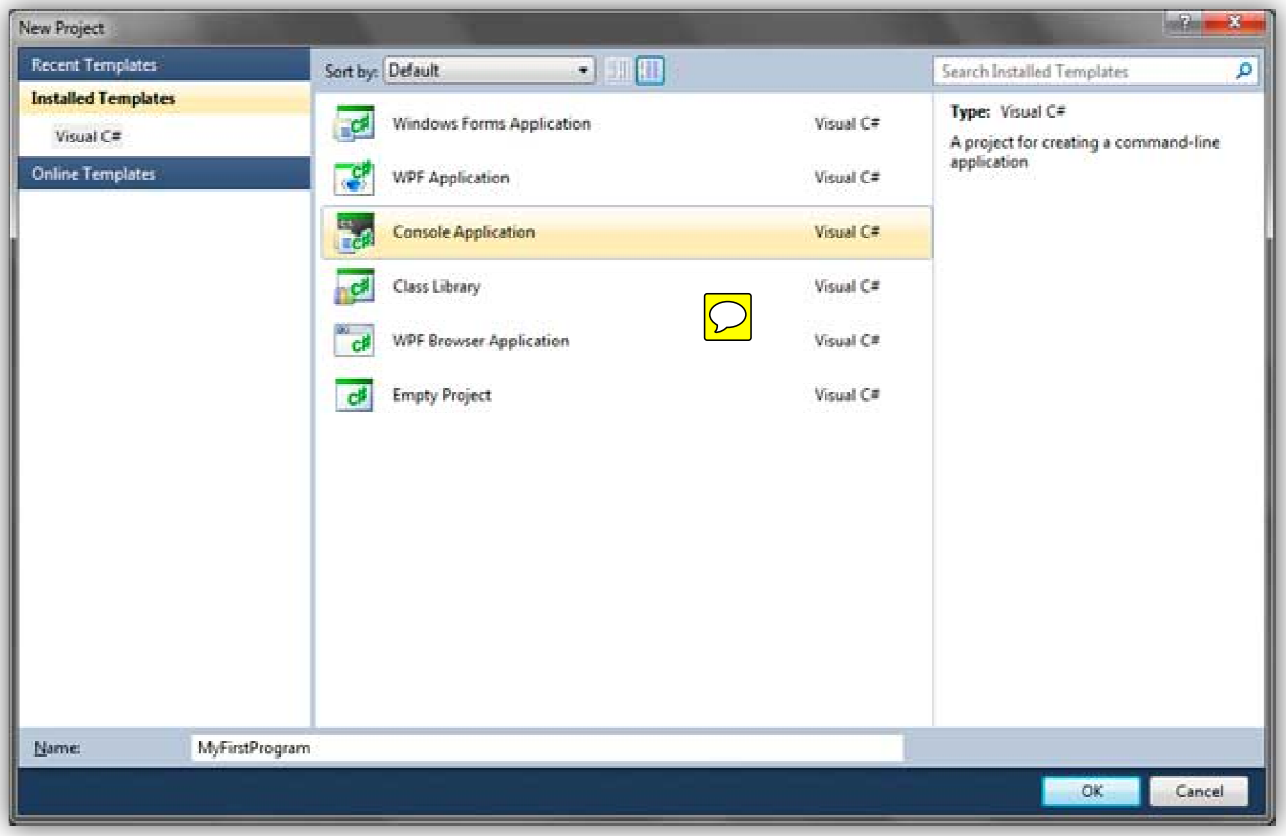
ساخت یک برنامه ساده

اجازه بدهید یک برنامه بسیار ساده به زبان سی شارپ بنویسیم.این برنامه یک پیغام را در محیط کنسول نمایش می دهد.در این درس می خواهم ساختار و دستور زبان یک برنامه ساده سی شارپ را توضیح دهم.

برنامه **Visual C# Express** را اجرا کنید.

از مسیر **File > New Project** یک پروژه جدید ایجاد کنید.

حال با یک صفحه مواجه می شوید که از شما می خواهد نام پروژه تان را انتخاب و آن را ایجاد کنید(شکل زیر)

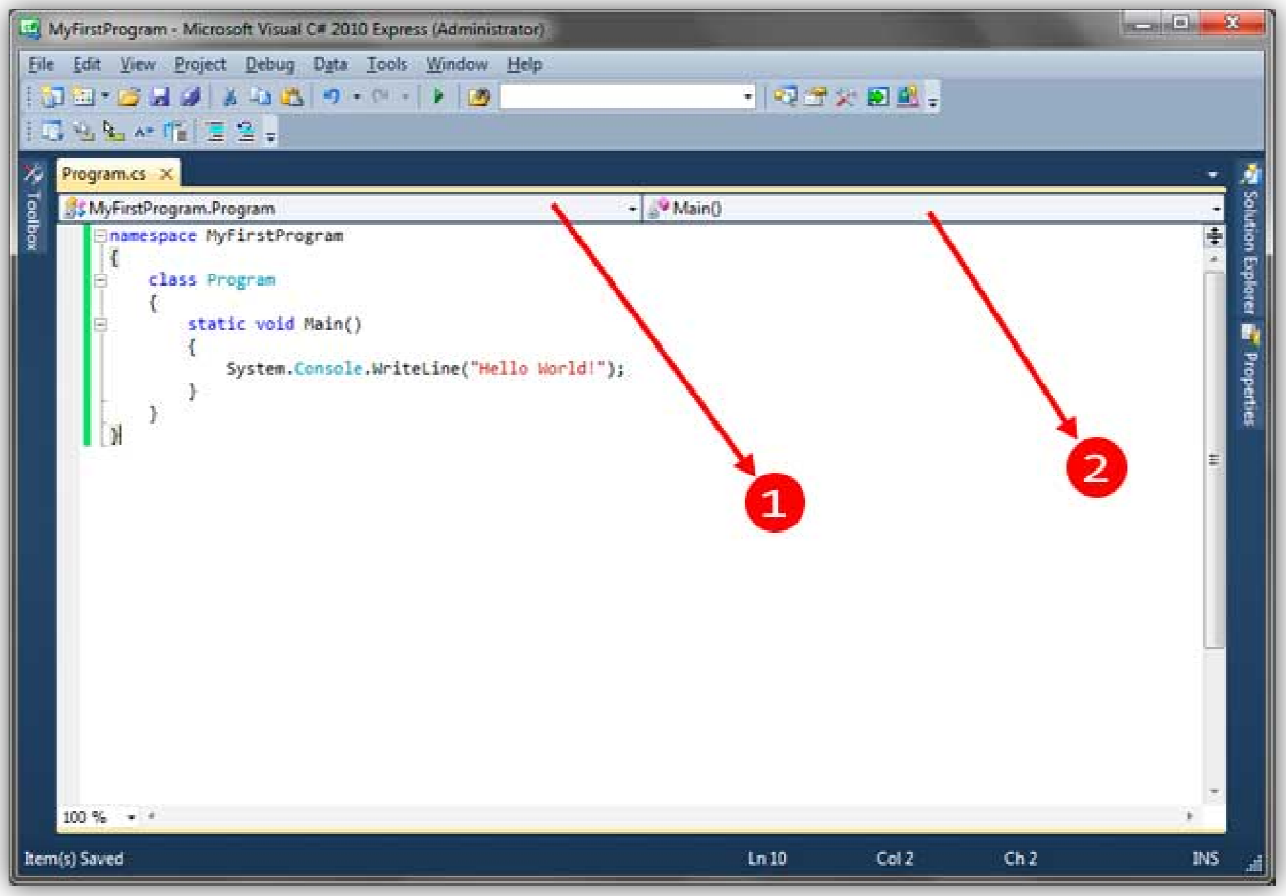


گزینه **Console Application** را انتخاب کنید و نام پروژه تان را **MyFirstProgram** انتخاب کنید. یک **Console Application** برنامه ای تحت داس در محیط وینوز است و فاقد محیط گرافیکی می باشد. بهتر است برنامه خود را در محیط کنسول بنویسید تا بیشتر با مفهوم

برنامه نویسی آشنا شوید،بعد از اینکه آموزش مبانی زبان به پایان رسید،برنامه نویسی در محیط ویندوز و بخش بصری سی شارپ را آموزش خواهیم داد.

بعد از فشردن دکمه **OK** ، برنامه **Visual C#** یک **solution** در یک فولدر موقتی ایجاد می کند،یک **solution** مجموعه ای از پروژه هاست ،اما در بیشتر تمرینات شامل یک پروژه می باشد،فایل **solution** دارای پسوند **sln** می باشد و شامل جزییاتی در مورد پروژه ها و فایلهای وابسته به آن می باشد،پروژه جدید همچنین حاوی یک فایل با پسوند **csproj.** می باشد که آن نیز شامل جزییاتی در مورد پروژه ها و فایلهای وابسته به آن می باشد.

حال می خواهیم شما را با محیط کد نویسی آشنا کنیم.



محیط کدنویسی جایی است که ما کدها را در آن تایپ می کنیم،کدها در محیط کدنویسی به صورت رنگی تایپ می شوند در نتیجه تشخیص بخشهای مختلف کد را راحت می کند.

منوی سمت چپ (شماره 1) شامل لیست کلاسها ،ساختارها ، انواع شمارشی و...و منوی سمت راست (شماره 2)شامل اعضای کلاسها ،ساختارها ، انواع شمارشی و... می باشد،نگران اصطلاحاتی که به کار بردیم نباشید آنها را در فصول بعد توضیح خواهم داد.

همه فایلهای دارای کد در سی شارپ دارای پسوند cs. هستند.

در محل کد نویسی کدهایی از قبل نوشته شده که برای شروع شما آنها را پاک کنید و کدهای زیر را در محل کدنویسی بنویسید :

```
namespace MyFirstProgram
{
    class Program
    {
        static void Main()
        {
            System.Console.WriteLine("Welcome to Visual C# Tutorials!");
        }
    }
}
```

ساختار یک برنامه در سی شارپ

مثال بالا ساده ترین برنامه ای است که شما می توانید در سی شارپ بنویسید. هدف در مثال بالا نمایش یک پیغام در صفحه نمایش است،هر زبان برنامه نویسی دارای قواعدی برای کدنویسی است.

اجازه بدهید هر خط کد را در مثال بالا توضیح بدهیم.

در خط اول فضای نام (namespace) تعریف شده است که شامل کدهای نوشته شده توسط شما است و از تداخل نامها جلوگیری می کند،در باره فضای نام در درسهای آینده توضیح خواهیم داد.

در خط دوم آکولاد ({) نوشته شده است،آکولاد برای تعریف یک بلوک کد به کار می رود،سی شارپ یک زبان ساخت یافته است که شامل کدهای زیاد و ساختارهای فراوانی می باشد،هر آکولاد باز({) در سی شارپ باید دارای یک آکولاد بسته (}) نیز باشد،همه کدهای نوشته شده از خط 2 تا خط 10 یک بلوک کد یا بدنه فضای نام است.

در خط 3 یک کلاس تعریف شده است،در باره کلاسها در فصلهای آینده توضیح خواهیم داد،در مثال بالا کدهای شما باید در داخل یک کلاس نوشته شود،بدنه کلاس شامل کد های نوشته شده از خط 4 تا 9 می باشد.

خط 5 متد Main یا متد اصلی نامیده می شود. هر متد شامل یک سری کد است که وقتی اجرا می شوند که متد را صدا بزنیم.در باره متد و نحوه صدا زدن آن در فصول بعدی توضیح خواهیم داد.متد Main نقطه آغاز اجرای برنامه است.این بدان معناست که ابتدا تمام کدهای داخل متد Main و سپس بقیه کدها اجرا می شود.در باره متد Main در فصول بعدی توضیح خواهیم داد.متد Main و سایر متدها دارای آکولاد و کدهایی در داخل آنها می باشند و وقتی کدها اجرا می شوند که متدها را صدا بزنیم.هر خط کد در سی شارپ به یک سیمیکولن (;) ختم می شود.اگر سیمیکولن در آخر خط فراموش شود برنامه با خطا مواجه می شود.

مثالی از یک خط کد در سی شارپ به صورت زیر است :

```
System.Console.WriteLine("Welcome to Visual C# Tutorials!");
```

این خط کد پیغام Welcome to Visual C# Tutorials! را در صفحه نمایش نشان می دهد.از متد WriteLine() برای چاپ یک رشته استفاده می شود.یک رشته گروهی از کاراکترها است که به وسیله دابل کوتیشن (") محصور شده است. مانند : "Welcome to Visual C# Tutorials!"

یک کاراکتر میتواند یک حرف،عدد، علامت یا ...باشد.در کل مثال بالا نحوه استفاده از متد WriteLine است که در داخل کلاس Consoleکه آن نیز به نوبه خود در داخل فضای نام MyFirstProgram قرار دارد را نشان می دهد.

توضیحات بیشتر در درسهای آینده آمده است.سی شارپ فضای خالی و خطوط جدید را نادیده می گیرد.بنابراین شما می توانید همه برنامه را در یک خط بنویسید.اما اینکار خواندن و اشکال زدایی برنامه را مشکل می کند.

یکی از خطاهای معمول در برنامه نویسی فراموش کردن سیمیکولن در پایان هر خط کد است.به مثال زیر توجه کنید :

```
System.Console.WriteLine(
"Welcome to Visual C# Tutorials!");
```

سی شارپ فضای خالی بالا را نادیده می گیرد و از کد بالا اشکال نمیگیرد.

اما از کد زیر ایراد می گیرد.

```
System.Console.WriteLine( ;
"Welcome to Visual C# Tutorials!");
```

به سیمیکولن آخر خط اول توجه کنید.برنامه با خطای نحوی مواجه می شود چون دو خط کد مربوط به یک برنامه هستند و شما فقط باید یک سیمیکولن در آخر آن قرار دهید.

همیشه به یاد داشته باشید که سی شارپ به بزرگی و کوچکی حروف حساس است.یعنی به طور مثال MAN و man در سی شارپ با هم فرق دارند.

رشته ها و توضیحات از این قاعده مستثنی هستند که در درسهای آینده توضیح خواهیم داد.

مثلا کدهای زیر با خطا مواجه می شوند و اجرا نمی شوند :

```
system.console.writeline("Welcome to Visual C# Tutorials!");
SYSTEM.CONSOLE.WRITELINE("Welcome to Visual C# Tutorials!");
sYsTem.cONsoLe.wRItelIne("Welcome to Visual C# Tutorials!");
```

تغییر در بزرگی و کوچکی حروف از اجرای کدها جلوگیری می کند.

اما کد زیر کاملا بدون خطا است :

```
System.Console.WriteLine("WELCOME TO VISUAL C# TUTORIALS!");
```

همیشه کدهای خود را در داخل آکولاد بنویسید.

```
{
   statement1;
}
```

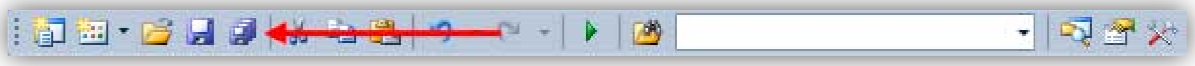
این کار باعث می شود که کدنویسی شما بهتر به چشم بیاید و تشخیص خطاها راحت تر باشد.

یکی از ویژگیهای مهم سی شارپ نشان دادن کدها به صورت تو رفتگی است بدین معنی که کدها را به صورت تو رفتگی از هم تفکیک می کند و این در خوانایی برنامه بسیار موثر است.

ذخیره پروژه و برنامه

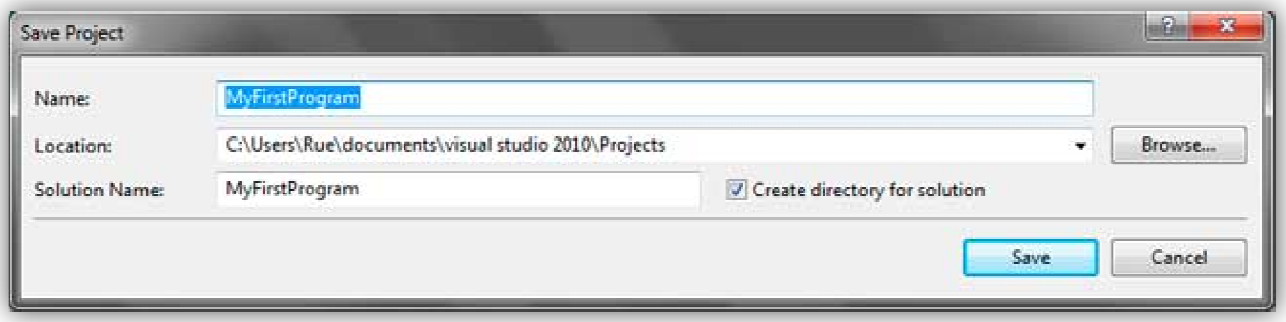
برای ذخیره پروژه و برنامه می توانید به مسیر **File > Save All** بروید یا از کلیدهای میانبر **Ctrl+Shift+S** استفاده کنید.

همچنین می توانید از قسمت **Toolbar** بر روی شکل زیر کلیک کنید :



که در این صورت برنامه شما فورا ذخیره می شود.

وقتی که بر روی شکل بالا کلیک می کنید صفحه زیر باز می شود که از شما می خواهد که نام و محل ذخیره پروژه را مشخص کنید.



در کادر جلوی کلمه **Name** ،نام و در جلوی کلمه **Location** با استفاده از دکمه **Browse** میتوان محل ذخیره پروژه را در هارد دیسک تعیین کرد.حال بر روی دکمه **Save** کلیک کنید تا برنامه ذخیره شود.برای باز کردن یک پروژه یا برنامه از منوی **File** گزینه **Open** را انتخاب

می کنید یا بر روی آیکون پوشه در **toolbar** درست قبل از آیکون **Save** کلیک کنید.سپس به محلی که پروژه در آنجا ذخیره شده میروید و فایلی با پسوند **sln** یا پروژه با پسوند **csproj**. را باز می کنید.

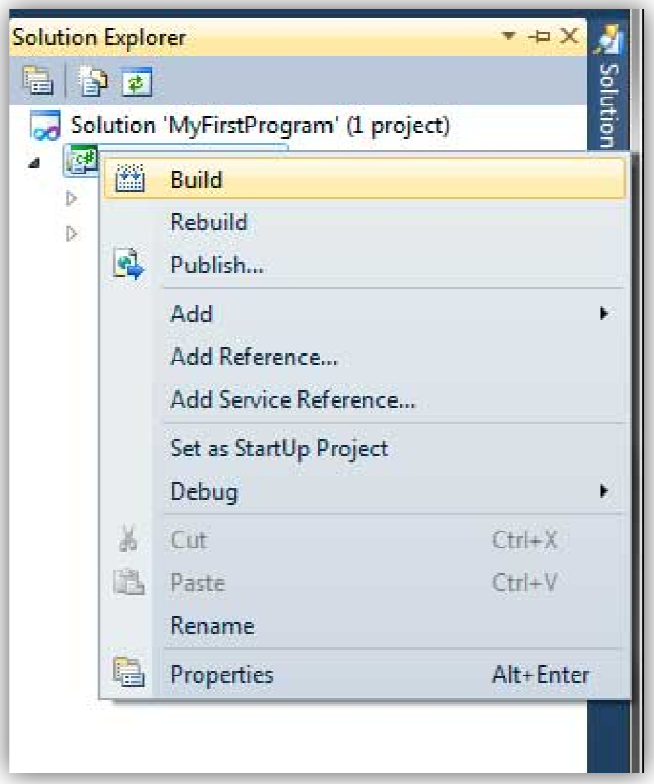
اصلاح برنامه

قبلا یاد ذکر شد که کدهای ما قبل از اینکه آنها را اجرا کنیم ابتدا به زبان میانی مایکروسافت ترجمه می شوند.

برای کامپایل برنامه از منوی **Debug** گزینه **Build Solution** را انتخاب می کنید یا دکمه **F6** را بر روی صفحه کلید فشار می دهیم.

این کار همه پروژه های داخل **solution** را کامپایل میکند.

برای کامپایل یک قسمت از **solution** به **Solution Explorer** می رویم و بر روی آن قسمت راست کلیک کرده و از منوی باز شوند گزینه **build** را انتخاب می کنید. مانند شکل زیر :



اجرای برنامه

وقتی ما برنامه مان را اجرا می کنیم سی شارپ به صورت اتوماتیک کدهای ما را به زبان میانی مایکروسافت کامپایل می کند،دو راه برای اجرای برنامه وجود دارد :

- اجرا همراه با اشکال زدایی (Debug)
- اجرا بدون اشکال زدایی (Non-Debug)

اجرای بدون اشکال زدایی برنامه، خطاهای برنامه را نادیده می گیرد،با اجرای برنامه در حالت **Non-Debug** سریعا برنامه اجرا می شود و شما با زدن یک دکمه از برنامه خارج می شوید،در حالت پیش فرض حالت **Non-Debug** مخفی است و برای استفاده از آن می توان از منوی **Debug** گزینه **Start Without Debuging** را انتخاب کرد یا از دکمه های ترکیبی **Crl + F5** استفاده نمود:

```
Welcome to Visual C# Tutorials!
Press any key to continue . . .
```

به این نکته توجه کنید که پیغام **Press any key to continue...** جز خروجی به حساب نمی آید و فقط نشان دهنده آن است که برنامه در حالت **Non-Debug** اجرا شده است و شما می توانید با زدن یک کلید از برنامه خارج شوید. دسترسی به حالت **Debug Mode** آسان تر است و به صورت پیشفرض برنامه ها در این حالت اجرا می شوند.

از این حالت برای رفع خطاها و اشکال زدایی برنامه ها استفاده می شود که در درسهای آینده توضیح خواهیم داد.

شما همچنین می توانید از **break points** و قسمت **Help** برنامه در مواقعی که با خطا مواجه می شوید استفاده کنید.

برای اجرای برنامه با حالت **Debug Mode** می توانید از منوی **Debug** گزینه **Start Debugging** را انتخاب کرده و یا دکمه **F5** را فشار دهید. همچنین می توانید بر روی فلش سبز رنگ در **toolbar** کلیلک کنید (شکل زیر).

اگر از حالت **Debug Mode** استفاده کنید برنامه نمایش داده شده و فورا ناپدید می شود،برای جلوگیری از این اتفاق شما می توانید از کلاس و متد **System.Console.ReadKey()** برای توقف برنامه و گرفتن ورودی از کاربر جهت خروج از برنامه استفاده کنید. (درباره متدها در درسهای آینده توضیح خواهیم داد.)

```
namespace MyFirstProgram
{
    class Program
    {
        static void Main()
        {
            System.Console.WriteLine("Welcome to Visual C# Tutorials!");
            System.Console.ReadKey();
        }
    }
}
```

حال برنامه را در حالت **Debug Mode** اجرا می کنیم، مشاهده می کنید که برنامه متوقف شده و از شما در خواست ورودی می کند، به سادگی و با زدن دکمه **Enter** از برنامه خارج شوید، من از حالت **Non-Debug** به این علت استفاده کردم تا نیازی به نوشتن کد اضافی **Console.ReadKey()** نباشد،از این به بعد هر جا ذکر شد که برنامه را اجرا کنید برنامه را در حالت **Non-Debug** اجرا کنید،وقتی به مبحث استثناء ها رسیدیم از حالت **Debug** استفاده می کنیم، حال با خصوصیات و ساختار اولیه سی شارپ آشنا شدید در دسهای آینده مطالب بیشتری از این زبان برنامه نویسی قدرتمند خواهید آموخت،

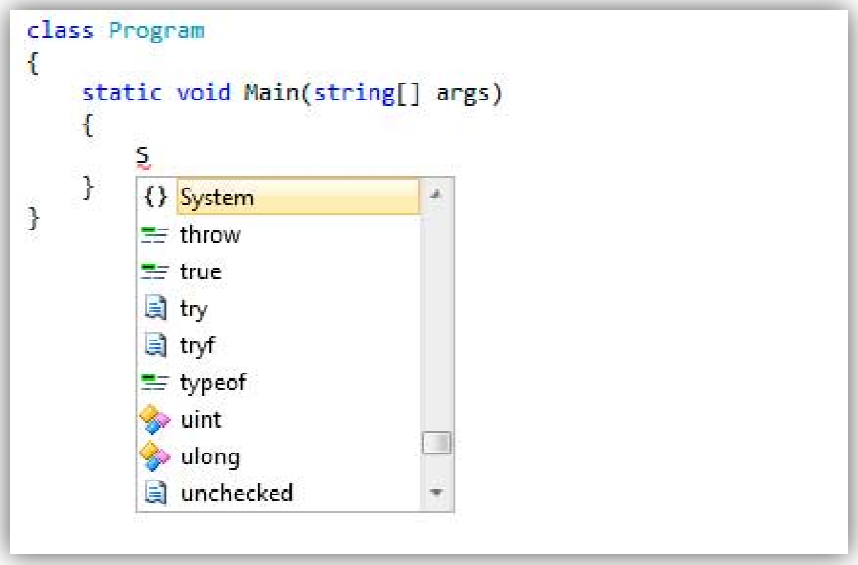
استفاده از IntelliSense

شاید یکی از ویژگیهای مهم **Visual Studio** ، اینتل لایسنس **(IntelliSense)** باشد.**IntelliSense** ما را قادر می سازد که به سرعت به کلاسها و متدها و....دسترسی پیدا کنیم.وقتی که شما در محیط کدنویسی حرفی را تایپ کنید .**IntelliSense** فورا فعال می شود.

کد زیرا را در داخل متد **Main** بنویسید.

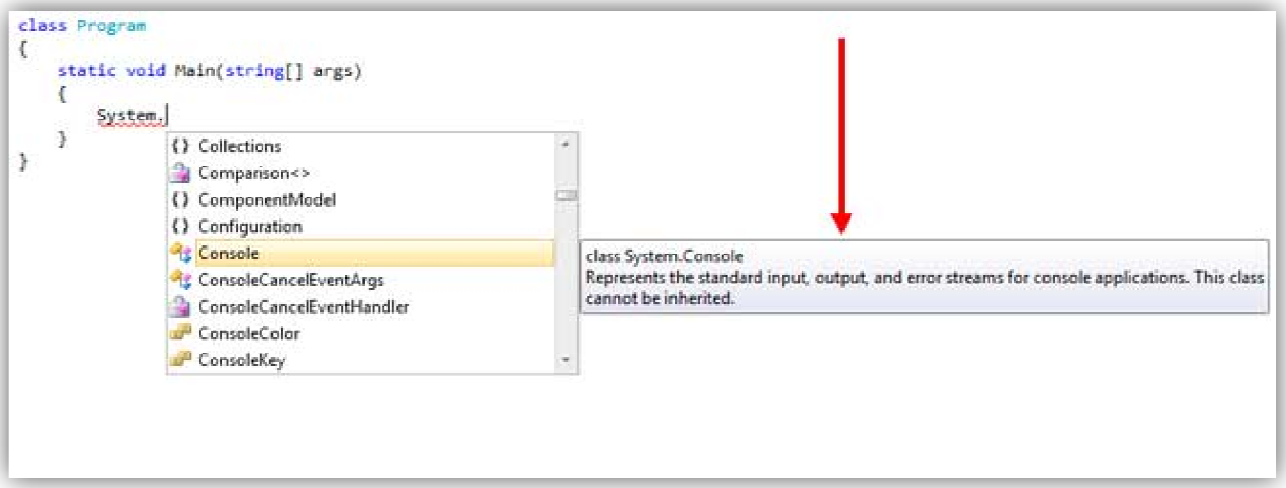
```
System.Console.WriteLine("Welcome to Visual C# Tutorials!");
```

اولین حرف را تایپ کنید تا **IntelliSense** فعال شود.



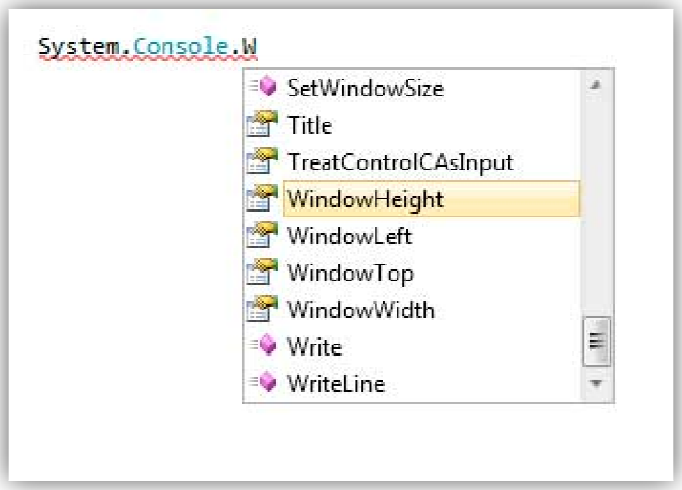
IntelliSense لیستی از کلمات به شما پیشنهاد می دهد که بیشترین تشابه را با نوشته شما دارند.شما می توانید با زدن دکمه **tab** گزینه مورد نظرتان را انتخاب کنید.

با تایپ نقطه **(.)** شما با لیست پیشنهادی دیگری مواجه می شوید.

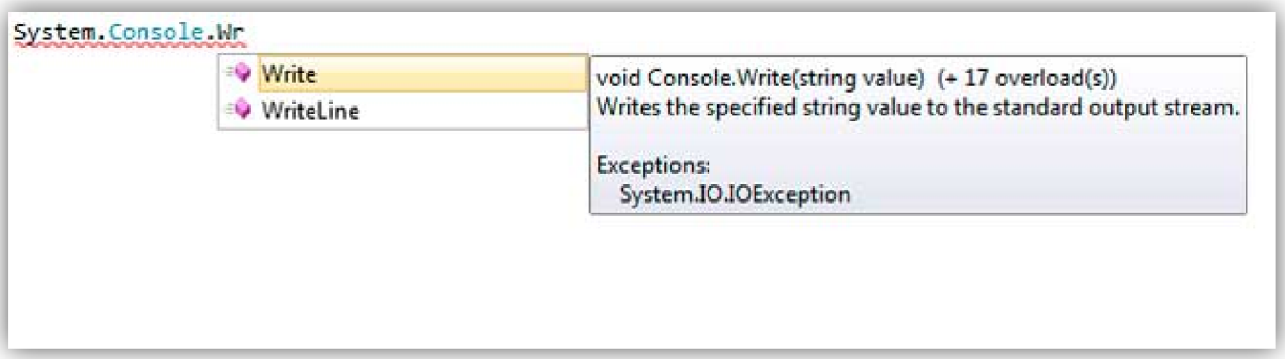


اگر بر روی گزینه ای که می خواهید انتخاب کنید لحظه ای مکث کنید توضیحی در رابطه با آن مشاهده خواهید کرد مانند شکل بالا :

هر چه که به پایان کد نزدیک می شوید لیست پیشنهادی محدود تر می شود.برای مثال با تایپ حرف W ، IntelliSense فقط کلماتی را که دارای حرف W هستند را نمایش می دهد.



با تایپ حرف های بیشتر لیست محدودتر شده و فقط دو کلمه را نشان می دهد.



اگر **IntelliSense** نتواند چیزی را که شما تایپ کرده اید پیدا کند هیچ چیزی را نمایش نمی دهدبرای ظاهر کردن **IntelliSense** کافیست دکمه ترکیبی **Ctrl+Space** را فشار دهید،برای انتخاب یکی از متدهایی که دارای چند حالتهستند، میتوان با استفاده از دکمه های مکان نما **(بالا و پایین)** یکی از حالت ها را انتخاب کرد. مثلا متد **Writeline** همانطور که در شکل زیر مشاهده می کنید دارای **19** حالت نمایش پیغام در صفحه است.

IntelliSense به طور هوشمند کدهایی را به شما پیشنهاد می دهد و در نتیجه زمان نوشتن کد را کاهش می دهد.

رفع خطاها

بیشتر اوقات هنگام برنامه نویسی با خطا مواجه می شویم. تقریبا همه برنامه هایی که امروزه می بینید حداقل از داشتن یک خطا رنج می برند.خطا ها می توانند برنامه شما را با مشکل مواجه کنند .

در سی شارپ سه نوع خطا وجود دارد :

- خطای کامپایلری –این نوع خطا از اجرای برنامه شما جلوگیری می کند. این خطا ها شامل خطای دستور زبان می باشد.این بدین معنی است که شما قواعد کد نویسی را رعایت نکرده اید.یکی دیگر از موارد وقوع این خطا هنگامی است که شما از چیزی استفاده

می کنید که نه وجود دارد و نه ساخته شده است. حذف فایلها یا اطلاعات ناقص در مورد پروژه ممکن است باعث به وجود آمدن خطای کامپایلری شود.استفاده از برنامه بوسیله برنامه دیگر نیز ممکن است باعث جلوگیری از اجرای برنامه و ایجاد خطای کامپایلری شود.

- **خطاهای منطقی**–این نوع خطا در اثر تغییر در یک منطق موجود در برنامه به وجود می آید.رفع این نوع خطاها بسیار سخت است چون شما برای یافتن آنها باید کد را تست کنید.نمونه ای از یک خطای منطقی برنامه ای است که دو عدد را جمع می کند ولی حاصل تفریق دو عدد را نشان می دهد.در این حالت ممکن ات برنامه نویس علامت ریاضی را اشتباه تایپ کرده باشد.

- **استثناء** - این نوع خطاها هنگامی رخ می دهند که برنامه در حال اجراست. این خطا هنگامی روی می دهد که کاربر یک ورودی نامعتبر به برنامه بدهد و برنامه نتواند آن را پردازش کند.

ویژوال استودیو و ویژوال سی شارپ دارای ابزارهایی برای پیدا کردن و برطرف کردن خطاها هستند.وقتی در محیط کدنویسی در حال تایپ کد هستیم یکی از ویژگیهای ویژوال استودیو تشخیص خطاهای ممکن قبل از اجرای برنامه است.

زیر کدهایی که دارای خطای کامپایلری هستند خط قرمز کشیده می شود.

```
namespace MyFirstProgram
{
    class Program
    {
        static voidx Main(string[] args)
        {
            System.ConSolle.WriTTe()
            System.Console.WriteLine("Hello")
            Demonstrating Syntax Errors
        }
    }
}
```

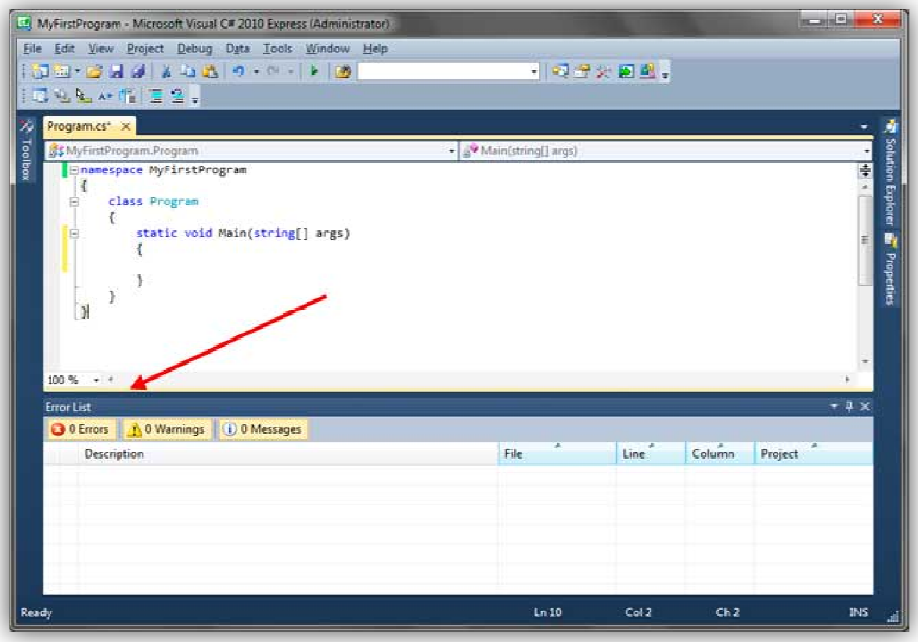
هنگامی که شما با موس روی این خطوط توقف کنید توضیحات خطا را مشاهده میکنید.شما ممکن است با خط سبز هم مواجه شوید که نشان دهنده اخطار در کد است ولی به شما اجازه اجرای برنامه را می دهند.به عنوان مثال ممکن است شما یک متغیر را تعریف کنید ولی در طول برنامه از آن استفاده نکنید. (در درس های آینده توضیح خواهیم داد).

```
namespace MyFirstProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            int number;
        }
    }
}
```

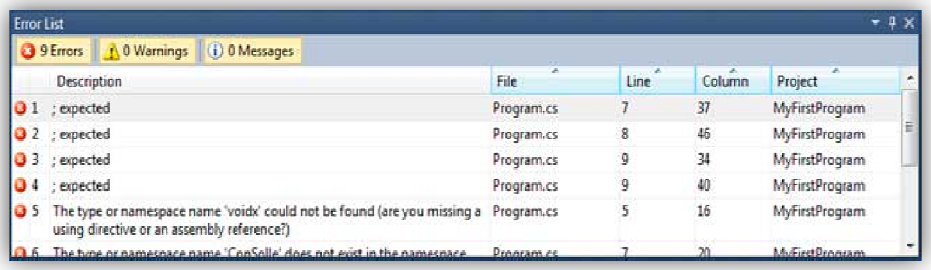
در باره رفع خطاها در آینده توضیح بیشتری می دهیم.

ErrorList (لیست خطاها) که در شکل زیر با فلش قرمز نشان داده شده است به شما امکان مشاهده خطاها ، هشدارها و رفع آنها را می دهد.

برای باز کردن **Error List** می توانید به مسیر **View > Other Windows > Error List** بروید.



همانطور که در شکل زیر مشاهده می کنید هرگاه برنامه شما با خطا مواجه شود لیست خطاها در **Error List**نمایش داده می شود.



در شکل بالا تعدادی خطا همراه با راه حل رفع آنها در **Error List** نمایش داده شده است.

Error List دارای چندین ستون است که به طور کامل جزییات خطاها را نمایش می دهند.

ستون	توضیحات
Description	توضیحی درباره خطا
File	فایلی که خطا در آن اتفاق افتاده است
Line	شماره خطی از فایل که دارای خطاست
Column	ستون یا موقعیت افقی خطا در داخل خط
Project	نام پروژه ای که دارای خطاست.

اگر برنامه شما دارای خطا باشد و آن را اجرا کنید با پنجره زیر روبرو می شوید :

مربع کوچک داخل پنجره بالا را تیک زنید چون دفعات بعد که برنامه شما با خطا مواجه شود دیگر این پنجره به عنوان هشدار نشان داده نخواهد شد.

با کلیک بر رو ی دکمه **Yes** برنامه با وجود خطا نیز اجرا می شود،اما با کلیک بر روی دکمه **NO** اجرای برنامه متوقف می شود و شما باید خطاهای موجود در پنجره **Error List** را بر طرف نمایید.

یکی دیگر از ویژگیهای مهم پنجره **Error List** نشان دادن قسمتی از برنامه است که دارای خطاست،با یک کلیک ساده بر روی هر کدام خطاهای موجود در پنجره **Error List** ، محل وقوع خطا نمایش داده می شود.

خطایابی و برطرف کردن آن

در جدول زیر لیست خطاهای معمول در پنجره Error List و نحوه برطرف کردن آنها آمده است :

کلمه Sample ،جانشین نامهای وابسته به خطاهایی است که شما با آنها مواجه می شوید و در کل یک کلمه اختیاری است:

خطا	توضیح	راه حل
; expected	**در پایان دستور علامت سیمیکان (;) قرار نداده اید**	**اضافه کردن یک سیمیکالن (;)**
The name ‘sample’ does not exist in the current context.	**کلمه sample در کد شما نه تعریف شده و نه وجود دارد**	**کلمه sample را حذف یا تعریف کنید.**
Only assignment, call, increment, decrement, and new object expressions can be used as a statement.	**کد جز دستورات سی شارپ نیست**	**دستور را حذف کنید**
Use of unassigned local variable ‘sample’	**متغیر sample مقدار دهی اولیه نشده**	**قبل از استفاده از متغیر آن را مقدار دهی اولیه کنید**
The type or namespace name ‘sample’ could not be found (are you missing a using directive or an assembly reference?)	**نوع یا فضای نام متغیر sample تعریف نشده است**	**باید یک کلاس یا فضای نام ،به نام sample ایجاد کنید**
‘MyMethod()’: not all code paths return a value	**بدین معنات که متد MyMethod() که به عنوان متدی با مقدار برگشتی در نظر گرفته شده در همه قسمتهای کد دارای مقدار برگشتی نیست.**	**مطمئن شوید که متد در همه قسمتهای کد دارای مقدار برگشتی است**
Cannot implicitly convert type ‘type1′ to ‘type2′	**متغیر type2 نمی تواند به متغیر type1 تبدیل شود.**	**با استفاده از متدهای تبدیل انواع به هم ،دو متغیر را یکسان کنید.**

نگران یادگیری کلمات به کار رفته در جدول بالا نباشید چون توضیح انها در درسهای آینده آمده است.

توضیحات

وقتی که کدی تایپ می کنید شاید بخواهید که متنی جهت یادآوری وظیفه آن کد به آن اضافه کنید.
در سی شارپ (و بیشتر زبانهای برنامه نویسی) می توان این کار را با استفاده از توضیحات انجام داد.
توضیحات متونی هستند که توسط کامپایلر نادیده گرفه می شوند و به عنوان بخشی از کد محسوب نمی شوند.
هدف اصلی از ایجاد توضیحات خوانایی و تشخیص نقش کدهای نوشته شده توسط شما ، برای دیگران است.
فرض کنید که می خواهید در مورد یک کد خاص، توضیح بدهید، می توانید توضیحات را در بالای کد یا کنار آن بنویسید.
از توضیحات برای مستند سازی برنامه هم استفاده می شود.
در برنامه زیر نقش توضیحات نشان داده شده است :

```
1   namespace CommentsDemo
2   {
3       class Program
4       {
5           public static void Main(string[] args)
6           {
7               // This line will print the message hello world
8               System.Console.WriteLine("Hello World!");
9           }
10      }
11  }
```

در خط 7 یک توضیح ساده (تک خطی) نشان داده شده است.
توضحات بر دو نوعند، توضیحات تک خطی و توضیحات چند خطی :

```
// single line comment

/* multi
   line
   comment */
```

توضیحات تک خطی همانگونه که از نامش پیداست برای توضیحاتی در حد یک خط به کار می روند.
این توضیحات با علامت // شروع می شوند و هر نوشته ای که در سمت راست آن قرار بگیرد جز توضیحات به حساب می آید.
این نوع توضیحات معمولا در بالا یا کنار کد قرار می گیرند.
اگر توضیح در باره یک کد به بیش از یک خط نیاز باشد از از توضیحات چند خطی استفاده می شود. توضیحات چند خطی با /* شروع و با */ پایان می یابند.
هر نوشته ای که بین این دو علامت قرار بگیرد جز توضیحات محسوب می شود.
نوع دیگری از توضیحات ، توضیحات XML نامیده می شوند.
این نوع با سه اسلش (///) نشان داده می شوند. از این نوع برای مستند سازی برنامه استفاده می شود و در درس های آینده در مورد آنها توضیح خواهیم داد.

کاراکترهای کنترلی

کاراکترهای کنترلی کاراکترهای ترکیبی هستند که با یک بک اسلش (\) شروع می شوند و به دنبال آنها یک حرف یا عدد می آید و یک رشته را با فرمت خاص نمایش می دهند. برای مثال برای ایجاد یک خط جدید و قرار دادن رشته در آن می توان از کاراکتر کنترلی \n استفاده کرد :

```
System.Console.WriteLine("Hello\nWorld!");
```

```
Hello
World
```

مشاهده کردید که کامپایلر بعد از مواجهه با کاراکتر کنترلی \nنشانگر موس را به خط بعد برده و بقیه رشته را در خط بعد نمایش می دهد.
متد **WriteLine()** هم مانند کاراکتر کنترلی \n یک خط جدید ایجاد می کند ، البته بدین صورت که در انتهای رشته یک کاراکتر کنترلی \n اضافه می کند :

```
System.Console.WriteLine("Hello World!");
```

کد بالا و کد زیر هیچ فرقی با هم ندارند :

```
System.Console.Write("Hello World!\n");
```

متد **Write()** کارکردی شبیه به **WriteLine()** دارد با این تفاوت که نشان گر موس را در همان خط نگه می دارد و خط جدید ایجاد نمی کند. جدول زیر لیست کاراکترهای کنترلی و کارکرد آنها را نشان می دهد :

عملکرد	کاراکتر کنترلی	عملکرد	کاراکتر کنترلی
Form Feed	**\f**	چاپ کوتیشن	\’
خط جدید	**\n**	چاپ دابل کوتیشن	\”
سر سطر رفتن	**\r**	چاپ بک اسلش	\\
حرکت به صورت افقی	**\t**	چاپ فضای خالی	**\0**
حرکت به صورت عمودی	**\v**	صدای بیپ	**\a**
چاپ کاراکتر یونیکد	**\u**	حرکت به عقب	**\b**

ما برای استفاده از کاراکترهای کنترلی از بک اسلش(\) استفاده می کنیم.از آنجاییکه \ معنای خاصی به رشته ها می دهد برای چاپ بک اسلش (\) باید از (\\) استفاده کنیم :

```
System.Console.WriteLine("We can print a \\ by using the \\\\ escape sequence.");
```

```
We can print a \ by using the \\ escape sequence.
```

یکی از موارد استفاده از \\ ،نشان دادن مسیر یک فایل در ویندوز است :

```
System.Console.WriteLine("C:\\Program Files\\Some Directory\\SomeFile.txt");
```

```
C:\Program Files\Some Directory\SomeFile.txt
```

از آنجاییکه از دابل کوتیشن (") برای نشان دادن رشته ها استفاده می کنیم برای چاپ آن از "\استفاده می کنیم :

```
System.Console.WriteLine("I said, \"Motivate yourself!\".");
```

```
I said, "Motivate yourself!".
```

همچنین برای چاپ کوتیشن (') از \' استفاده می کنیم :

```
System.Console.WriteLine("The programmer\'s heaven.");
```
```
The programmer's heaven.
```

برای ایجاد فاصله بین حروف یا کلمات از \t استفاده می شود :

```
System.Console.WriteLine("Left\tRight");
```
```
Left    Right
```

هر چند تعداد کاراکتر که بعد از کاراکتر کنترلی \r بیایند به اول سطر منتقل و جایگزین کاراکترهای موجود می شوند :

```
System.Console.WriteLine("Mitten\rK");
```
```
Kitten
```

مثلا در مثال بالا کاراکتر K بعد از کاراکتر کنترلی \r آمده است،کاراکتر کنترلی حرف K را به ابتدای سطر برده و جایگزین حرف M می کند.

برای چاپ کاراکترهای یونیکد می توان از \u استفاده کرد. برای استفاده از \u ،مقدار در مبنای 16 کاراکتر را درست بعد از علامت \u قرار می دهیم،برای مثال اگر بخواهیم علامت کپی رایت (©)را چاپ کنیم باید بعد از علامت \u مقدار 00A9 را قرار دهیم مانند :

```
System.Console.WriteLine("\u00A9");
```
```
©
```

برای مشاهده لیست مقادیر مبنای 16 برای کاراکترهای یونیکد به لینک زیر مراجعه نمایید :

http://www.ascii.cl/htmlcodes.htm

اگر کامپایلر به یک کاراکتر کنترلی غیر مجاز برخورد کند، برنامه پیغام خطا می دهد،بیشترین خطا زمانی اتفاق می افتد که برنامه نویس برای چاپ اسلش (\) از \\ استفاده می کند.

علامت @

علامت @به شما اجازه می دهد که کاراکترهای کنترلی را رد کرده و رشته ای خوانا تر و طبیعی تر ایجاد کنید.

وقتی از کاراکترهای کنترلی در یک رشته استفاده می شود ممکن است برای تایپ مثلا یک بک اسلش (\ ) به جای استفاده از دو علامت \\ از یک \ استفاده کرده و دچار اشتباه شوید.

این کار باعث به وجود آمدن خطای کامپایلری شده و چون کامپایلر فکر می کند که شما می خواهید یک کاراکتر کنترلی را تایپ کنید و کاراکتر بعد از علامت \ را پردازش می کند و چون کاراکتر کنترلی وجود ندارد خطا به وجود می آید.

به مثال زیر توجه کنید :

```
System.Console.WriteLine("I want to have a cat\dog as a birthday present."); //Error
```

با وجودیکه بهتر است در مثال بالا از اسلش (/) در cat/dog استفاده شود ولی عمدا از بک اسلش (\ ) برای اثبات گفته بالا استفاده کرده ایم.

کامپایلر خطا ایجاد می کند و به شما می گوید که کاراکتر کنترلی \d قابل تشخیص نیست چون همچین کاراکتر کنترلی وجود ندارد.

زمانی وضعیت بدتر خواهد شد که کاراکتر بعد از بک اسلش کاراکتری باشد که هم جز یک کلمه باشد و هم جز کاراکترهای کنترلی.

به مثال زیر توجه کنید :

```
System.Console.WriteLine("Answer with yes\no:");
Answer with yes
o
```

استفاده از علامت @ برای نادیده گرفتن کاراکترهای کنترلی

استفاده از علامت @ زمانی مناسب است که شما نمی خواهید از علامت بک اسلش برای نشان دادن یک کارکتر کنترلی استفاده کنید.

استفاده از این علامت بسیار ساده است و کافی است که قبل از رشته مورد نظر آن را قرار دهید.

```
System.Console.WriteLine(@"I want to have a cat\dog as a birthday present.");
I want to have a cat\dog as a birthday present.
```

از علامت @ معمولا زمانی استفاده می شود که شما بخواهید مسیر یک دایرکتوری را به عنوان رشته داشته باشید.

چون دایرکتوری ها دارای تعداد زیادی بک اسلش هستند و طبیعتا استفاده از علامت @ به جای دابل بک اسلش (\\) بهتر است.

```
System.Console.WriteLine(@"C:\Some Directory\SomeFile.txt");
C:\Some Directory\SomeFile.txt
```

اگر بخواهید یک دابل کوتیشن چاپ کنید به سادگی می توانید از دو دابل کوتیشن استفاده کنید.

```
System.Console.WriteLine(@"Printing ""double quotations""...");
Printing "double quotations"...
```

از به کار بردن علامت @ و کاراکترهای کنترلی به طور همزمان خودداری کنید چون باعث چاپ کاارکتر کنترلی در خروجی می شود.

استفاده از علامت @ برای نگهداری از قالب بندی رشته ها

یکی دیگر از موارد استفاده از علامت @ چاپ رشته های چند خطی بدون استفاده از کاراکتر کنترلی \n است. بع عنوان مثال برای چاپ پیغام زیر :

```
C# is a great programming language and
it allows you to create different
kinds of applications.
```

یکی از راه های چاپ جمله بالا به صورت زیر است :

```
Console.WriteLine("C# is a great programming language and\n" +
"it allows you to create different\n" +
"kinds of applications.");
```

به نحوه استفاده از \n در آخر هر جمله توجه کنید.

این کاراکتر همانطور که قبلا مشاهده کردید خط جدید ایجاد می کند و بدر مثال بالا باعث می شود که جمله به چند خط تقسیم شود.

از علامت + هم برای ترکیب رشته ها استفاده می شود.

راه دیگر برای نمایش مثال بالا در چندین خط استفاده از علامت @ است

```
Console.WriteLine(@"C# is a great programming language and
it allows you to create different
kinds of applications.");
```

در این حالت کافیست که در هر جا که می خواهید رشته در خط بعد نمایش داده شود دکمه Enter را فشار دهید

متغیرها

متغیر مکانی از حافظه است که شما می توانید مقادیری را در آن ذخیره کنید،می توان آن را به عنوان یک ظرف تصور کرد که داده های خود را در آن قرار داده اید،محتویات این ظرف می تواند پاک شود یا تغییر کند،هر متغیر دارای یک نام نیز هست،که از طریق آن میتوان متغیر را از دیگر متغیر ها تشخیص داد و به مقدار آن دسترسی پیدا کرد،همچنین دارای یک مقدار می باشد که می تواند توسط کاربر انتخاب شده باشد یا نتیجه یک محاسبه باشد،مقدار متغیر می تواند تهی نیز باشد،متغیر دارای نوع نیز هست بدین معنی که نوع آن با نوع داده ای که در آن ذخیره می شود یکی است، متغیر دارای عمر نیز هست که از روی آن می توان تشخیص داد که متغیر باید چقدر در طول برنامه مورد استفاده قرار گیرد،و در نهایت متغیر دارای محدوده استفاده نیز هست که به شما می گوید که متغیر در چه جای برنامه برای شما قابل دسترسی است،ما از متغیرها به عنوان یک انبار موقتی برای ذخیره داده استفاده می کنیم، هنگامی که یک برنامه ایجاد می کنیم احتیاج به یک مکان برای ذخیره داده ،مقادیر یا داده هایی که توسط کاربر وارد می شوند داریم،ایم مکان همان متغیر است، برای این از کلمه متغیر استفاده می شود چون ما می توانیم بسته به نوع شرایط هر جا که لازم باشد مقدار آن را تغییر دهیم، متغیرها موقتی هستند و فقط موقعی مورد استفاده قرار می گیرند که برنامه در حال اجراست و وقتی شما برنامه را می بندید محتویات متغیر ها نیز پاک می شود، قبلا ذکر شد که به وسیله نام متغیر می توان به آن دسترسی پیدا کرد،

برای نامگذاری متغیرها باید قوانین زیر را رعایت کرد :

- نام متغیر باید با یک از حروف الفبا **(a-z or A-Z)** شروع شود،
- نمی تواند شامل کاراکترهای غیرمجاز مانند .$, ^, ?, # باشد،
- نمی توان از کلمات رزرو شده در سی شارپ برای نام متغیر استفاده کرد،
- نام متغیر نباید دارای فضای خالی **(spaces)** باشد،

اسامی متغیرها نسبت به بزرگی و کوچکی حروف حساس هستند، در سی شارپ دو حرف مانند **a** و **A** دو کاراکتر مختلف به حساب می آیند،

دو متغیر با نامهای **myNumber** و **MyNumber** دو متغیر مختلف محسوب می شوند چون یکی از آنها با حرف کوچک **m** و دیگری با حرف بزرگ **M** شروع می شود، شما نمی توانید دو متغیر را که دقیق شبیه هم هستند را در یک **scope (محدوده)** تعریف کنید،**Scope** به معنای یک بلوک کد است که متغیر در آن قابل دسترسی و استفاده است، در مورد **Scope** در فصلهای آینده بیشتر توضیح خواهیم داد،

متغیر دارای نوع هست که نوع داده ای را که در خود ذخیره می کند را نشان می دهد،

معمولترین انواع داده **int, double, string, char, float, decimal** می باشند، برای مثال شما برای قرار دادن یک عدد صحیح در متغیر باید از نوع **int** استفاده کنید،

انواع ساده

انواع ساده انواعی از داده ها هستند که شامل اعداد، کاراکترها و رشته ها و مقادیر بولی می باشند.به انواع ساده انواع اصلی نیز گفته می شود چون از آنها برای ساخت انواع پیچیده تری مانند کلاس ها و ساختارها استفاده می شود. انواع ساده دارای مجموعه مشخصی از مقادیر هستند و محدوده خاصی از اعداد را در خود ذخیره می کنند.

در جدول زیر انواع ساده و محدود آنها آمده است :

نوع	دامنه
sbyte	-128 تا 127
byte	0 تا 255
short	-32768 تا 32767
ushort	0 تا 65535
int	-2147483648 تا 2147483647
uint	0 تا 4294967295
long	-9223372036854775808 تا 922337203685477807
ulong	0 تا 18446744073709551615

به حرف u در ابتدای برخی از انواع داده ها مثلا ushort توجه کنید.این بدان معناست که این نوع فقط شامل اعداد مثبت و صفر هستند.

جدول زیر انواعی که مقادیر با ممیز اعشار را می توانند در خود ذخیره کنند را نشان می دهد :

نوع	دامنه تقریبی	دقت
float	±1.5E-45تا±3.4E38	رقم 7
double	±5.0E-324تا ±1.7E308	رقم 16 – 15
decimal	$(-7.9 \times 10^{28}) / (10^{0 \text{ to } 28})$ تا $(7.9 \times 10^{28}) / (10^{0 \text{ to } 28})$	رقم معنادار 29 – 28

برای به خاطر سپردن آنها باید از نماد علمی استفاده شود.

نوع دیگری از انواع ساده برای ذخیره داده های غیر عددی به کار می روند و در جدول زیر نمایش داده شده اند :

نوع	مقادیر مجاز
char	کاراکترهای یونیکد
bool	مقدار true یا false
string	مجموعه ای از کاراکترهای

نوع charرای ذخیره کاراکترهای یونیکد استفاده می شود.کاراکترها باید داخل یک کوتیشن ساده قرار بگیرند مانند ('a').

نوع bool فقط می تواند مقادیر درست (true) یا نادرست (false) را در خود ذخیره کند و بیشتر در برنامه هایی که دارای ساختار تصمیم گیری هستند مورد استفاده قرار می گیرد.

نوع string برای ذخیره گروهی از کاراکترها مانند یک پیغام استفاده می شود.مقادیر ذخیره شده در یک رشته باید داخل دابل کوتیشن قرار گیرند تا توسط کامپایلر به عنوان یک رشته در نظر گرفته شوند. مانند ("massage")

استفاده از متغیرها

در مثال زیر نحوه تعریف و مقدار دهی متغیرها نمایش داده شده است :

```
using System;

public class Program
{
    public static void Main()
    {
        //Declare variables
        int num1;
        int num2;
        double num3;
        double num4;
        bool boolVal;
        char myChar;
        string message;

        //Assign values to variables
        num1 = 1;
        num2 = 2;
        num3 = 3.54;
        num4 = 4.12;
        boolVal = true;
        myChar = 'R';
        message = "Hello World!";

        //Show the values of the variables
        Console.WriteLine("num1 = {0}", num1);
        Console.WriteLine("num2 = {0}", num2);
        Console.WriteLine("num3 = {0}", num3);
        Console.WriteLine("num4 = {0}", num4);
        Console.WriteLine("boolVal = {0}", boolVal);
        Console.WriteLine("myChar = {0}", myChar);
        Console.WriteLine("message = {0}", message);
    }
}
```

```
num1 = 1
num2 = 2
num3 = 3.54
num4 = 4.12
boolVal = true
myChar = R
message = Hello World!
```

تعریف متغیر

در خطوط 8-14 متغیرهایی با نوع و نام متفاوت تعریف شده اند. ابتدا باید نوع داده هایی را که این متغیر ها قرار است در خود ذخیره کنند را مشخص کنیم و سپس یک نام برای آنها در نظر بگیریم و در آخر سیمیکولن بگذاریم.همیشه به یاد داشته باشید که قبل از مقدار دهی و استفاده از متغیر باید آن را تعریف کرد.

```
int num1;
int num2;
double num3;
double num4;
bool boolVal;
char myChar;
string message;
```

نحوه تعریف متغیر به صورت زیر است :

```
data_typeidentifier;
```

date_type همان نوع داده است مانند int ،double و.... .

Identifier نیز نام متغیر است که به ما امکان استفاده و دسترسی به مقدار متغیر را می دهد.

برای تعریف چند متغیر از یک نوع می توان به صورت زیر عمل کرد :

```
data_type identifier1, identifier2, ... indentifierN;
```

مثال

```
int num1, num2, num3, num4, num5;
string message1, message2, message3;
```

در مثال بالا 5 متغیر از نوع صحیح و 3 متغیر از نوع رشته تعریف شده است.توجه داشته باشید که بین متغیر ها باید علامت کاما (,) باشد.

نامگذاری متغیرها

- نام متغیر باید با یک حرف یا زیرخط و به دنبال آن حرف یا عدد شروع شود.
- نمی توان از کاراکترهای خاص مانند &, %, # یا عدد برای شروع نام متغیر استفاده کرد مانند 2numbers .
- نام متغیر نباید دارای فاصله باشدبرای نام های چند حرفی میتوان به جای فاصله از علامت زیرخط یا _ استفاده کرد.

نامهای مجاز :

```
num1  myNumber  studentCount  total       first_name    _minimum
num2  myChar    average      amountDue   last_name     _maximum
name  counter   sum          isLeapYear  color_of_car  _age
```

نامهای غیر مجاز :

```
123         #numbers#  #ofstudents  1abc2
123abc      $money     first name   ty.np
my number   this&that  last name    1:00
```

اگر به نامهای مجاز در مثال بالا توجه کنید متوجه قراردادهای به کار رفته در نامگذاری آنها خواهید شد.یکی از روشهای نامگذاری،نامگذاری کوهان شتری است.در این روش اولین که برای متغرهای دو کلمه ای به کار می رود،اولین حرف اولین کلمه با حرف کوچک و اولین حرف دومین کلمه با حرف بزرگ نمایش داده می شود مانند : **myNumber** توجه کنید که اولین حرف کلمه **Naumber** با حرف بزرگ شروع شده است. مثال دیگر کلمه **numberOfStudents** است.اگز توجه کنید بعد از اولین کلمه حرف اول سایر کلمات با حروف بزرگ نمایش داده شده است.

محدوده متغیر

متغیرها در داخل متد **Main** تعریف می شوند.این متغیرها فقط در داخل متد **Main** قابل دسترسی هستند.

محدوده یک متغیر مشخص می کند که متغیر در کجای کد قابل دسترسی است.

هنگامیکه برنامه به پایان متد **Main** می رسد متغیرها از محدوده خارج و بدون استفاده می شوند تا زمانی که برنامه در حال اجراست.محدوده متغیرها انواعی دارد که در درسهای بعدی با آنها آشنا می شوید.

تشخیص محدوده متغیر بسیار مهم است چون به وسیله آن می فهمید که در کجای کد می توان از متغیر استفاده کرد.

باید یاد آور شد که دو متغیر در یک محدوده نمی توانند دارای نام یکسان باشند.

مثلا کد زیر در برنامه ایجاد خطا می کند :

```
int num1;
int num1;
```

از آنجاییکه سی شارپ به بزرگی و کوچک بودن حروف حساس است می توان از این خاصیت برای تعریف چند متغیر هم نام ولی با حروف متفاوت(از لحاظ بزرگی و کوچکی) برای تعریف چند متغیر از یک نوع استفاده کرد مانند :

```
int num1;
int Num1;
int NUM1;
```

مقداردهی متغیرها

می توان فورا بعد از تعریف متغیرها مقادیری را به آنها اختصاص داد.این عمل را مقداردهی می نامند.

در زیر نحوه مقدار دهی متغیر ها نشان داده شده است :

```
data_type identifier = value;
```

به عنوان مثال :

```
int myNumber = 7;
```

همچنین می توان چندین متغیر را فقط با گذاشتن کاما بین آنها به سادگی مقدار دهی کرد :

```
data_type variable1 = value1, varaible2 = value2, ... variableN, valueN;
```

```
int num1 = 1, num2 = 2, num3 = 3;
```

تعریف متغیر با مقدار دهی متغیرها متفاوت است.

تعریف متغیر یعنی انخاب نوع و نام برای متغیر ولی مقدار دهی یعنی اختصاص یک مقدار به متغیر.

اختصاص مقدار به متغیر

در زیر نحوه اختصاص مقادیر به متغیرها نشان داده شده است:

```
num1 = 1;
num2 = 2;
num3 = 3.54;
num4 = 4.12;
boolVal = true;
myChar = 'R';
message = "Hello World!";
```

به این نکته توجه کنید که شما به مغیری که هنوز تعریف نشده نمی توانید مقدار بدهید.

شما فقط می توانید از متغیرهایی استفاده کنید که هم تعریف و هم مقدار دهی شده باشند.

مثلا متغیرهای بالا همه قابل استفاده هستند.

در این مثال **num1** و **num2** هر دو تعریف شده اند و مقادیری از نوع صحیح به آنها اختصاص داده شده است.

اگر نوع داده با نوع متغیر یکی نباشد برنامه پیغام خطا می دهد.

جانگهدار (Placeholders)

به متد **WriteLine()** در خطوط **(26-32)** توجه کنید.

این متد دو آرگومان قبول می کند.

آرگومانها اطلاعاتی هستند که متد با استفاده از آنها کاری انجام می دهد.

آرگومانها به وسیله کاما از هم جدا می شوند.

آرگومان اول یک رشته قالب بندی شده است و آرگومان دوم مقداری است که توسط رشته قالب بندی شده مورد استفاده قرار می گیرد.

اگر به دقت نگاه کنید رشته قالب بندی شده دارای عدد صفری است که در داخل دو آکولاد محصور شده است. البته عدد داخل دو آکولاد می تواند از صفر تا **n** باشد. به این اعداد جانگهدار می گویند.

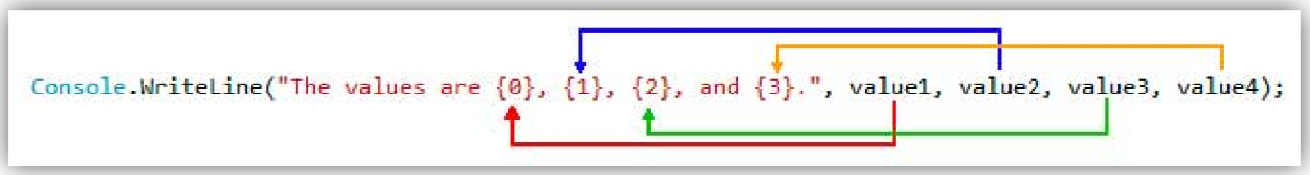
این اعداد بوسیله مقدار آرگومان بعد جایگزین می شوند.

به عنوان مثال جانگهدار **{0}** به این معناست که اولین آرگومان **(مقدار)** بعد از رشته قالب بندی شده در آن قرار می گیرد.

متد **WriteLine** عملا می تواند هر تعداد آرگومان قبول کند اولین آرگومان همان رشته قالب بندی شده است که جانگهدار در آن قرار دارد و دومئین آرگومان مقداری است که جایگزین جانگهدار می شود.

در مثال زیر از **4** جانگهدار استفاده شده است :

```
Console.WriteLine("The values are {0}, {1}, {2}, and {3}.", value1, value2, value3,
value4);
```



جانگهدارها از صفر شروع می شوند.

تعداد جانگهئدارها باید با تعداد آرگومانهای بعد از رشته قالب بندی شده برابر باشد.

برای مثال اگر شما چهار جانگهدار مثل بالا داشته باشید باید چهار مقدار هم برای آنها بعد از رشته قالب بندی شده در نظر بگیرید.

اولین جا نگهدار با دومین آرگومان و دومین جا نگهدار با سومین آرگومان جایگزین می شود.

در ابتدا فهمیدن این مفهوم برای کسانی که تازه برنامه نویسی را شروع کرده اند سخت است اما در درسهای آینده مثالهای زیادی در این مورد مشاهده خواهید کرد.

وارد کردن فضاهای نام

شاید به این نکته توجه کرده باشید که ما زمان فراخوانی متد **WriteLine()** و قبل از **Console** ،کلمه **System** را ننوشتیم چون در خط 1 و در ابتدای برنامه این کلمه را در قسمت تعریف فضای نام وارد کردیم.

```
using System;
```

این دستور بدین معناست که ما از تمام چیزهایی که در داخل فضای نام سیستم قرار دارند استفاده می کنیم.

پس به جای اینکه جمله زیر را به طور کامل بنویسیم :

```
System.Console.WriteLine("Hello World!");
```

می توانیم آن را ساده تر کرده و به صورت زیر بنویسیم :

```
Console.WriteLine("Hello World");
```

در مورد فضای نام در درسهای آینده توضیح خواهیم داد.

ثابت ها

ثابت ها انواعی از متغیرها هستند که مقدار آنها در طول برنامه تغییر نمی کند، ثابت ها حتما باید مقدار دهی اولیه شوند و اگر مقدار دهی آنها فراموش شود در برنامه خطا به وجود می آید، بعد از این که به ثابت ها مقدار اولیه اختصاص داده شد هرگز در زمان اجرای برنامه نمی توان آن را تغییر داد.

برای تعریف ثابت ها باید از کلمه کلیدی **const** استفاده کرد،معمولا نام ثابت ها را طبق قرارداد با حروف بزرگ می نویسند تا تشخیص آنها در برنامه راحت باشد، نحوه تعریف ثابت در زیر آمده است :

```
const data_type identifier = initial_value;
```

مثال :

```
class Program
{
    public static void Main()
    {
        const int NUMBER = 1;

        NUMBER = 10; //ERROR, Cant modify a constant
    }
}
```

در این مثال می بینید که مقدار دادن به یک ثابت ،که قبلا مقدار دهی شده برنامه را با خطا مواجه می کند،نکته ی دیگری که نباید فراموش شود این است که نباید مقدار ثابت را با مقدار دیگر متغیرهای تعریف شده در برنامه برابر قرار داد.

مثال :

```
int someVariable;
constint MY_CONST = someVariable
```

ممکن است این سوال برایتان پیش آمده باشد که دلیل استفاده از ثابت ها چیست؟ اگر مطمئن هستید که مقادیری در برنامه وجود دارند که هرگز در طول برنامه تغییر نمی کنند بهتر است که آنها را به صورت ثابت تعریف کنید،این کار هر چند کوچک کیفیت برنامه شما را بالا می برد.

تبدیل ضمنی

تبدیل ضمنی متغیرها یک نوع تبدیل ات که به طور خودکار توسط کامپایلر انجام می شود.

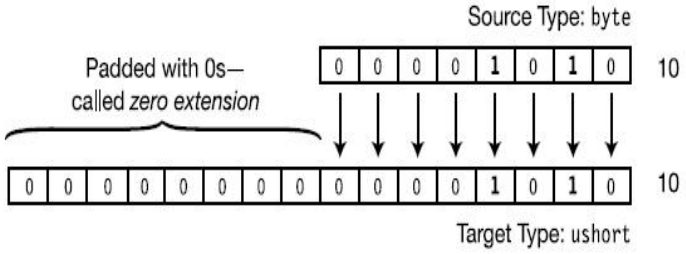
برای مثال شاید بخواهید داده ای 8 بیتی را به داده ای 16 بیتی تبدیل کنید ، بدون آنکه اصل داده دچار تغییر شود .

وقتی می خواهید داده ای را از نوع کوچکتر به نوع بزرگتر تبدیل کنید ، باید از تبدیل ضمنی استفاده کنید ، زبان سی شارپ این کار را بصورت اتوماتیک برای شما انجام می دهد . اما ....

1-هنگام تبدیل از نوع کوچکتر (نوع با تعداد بیت کمتر) به نوع بزرگتر (نوع با بیت بیشتر) بیتهای بیشتر باید با صفر و یک پر شوند.

2-هنگام تبدیل از نوع کوچکتر بدون علامت به نوع بزرگتر بدون علامت ، بیتهای بیشتر با صفر پر می شوند.

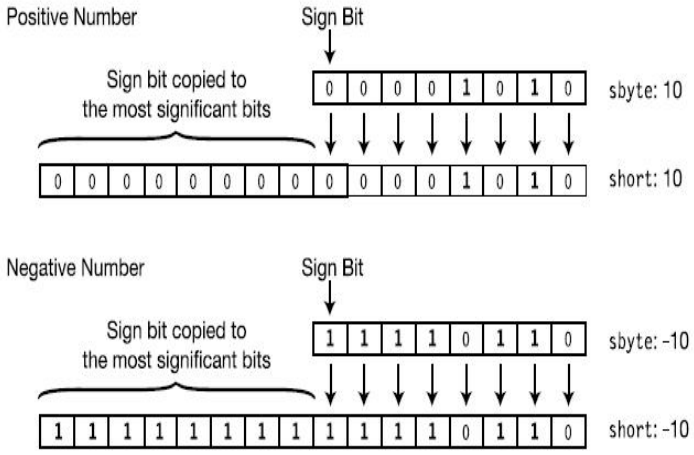
مانند شکل زیر



در شکل بالا ما عدد 10 که از نوع byte است را تبدیل به نوع بزرگتر از نوع ushort کردیم .

نکته : اگر هنگام تبدیل نوع کوچکتر به نوع بزرگتر ، داده ما دارای علامت مثبت یا منفی بود ، آنگاه بیتهای بیشتر با بیت علامت پر می شوند ، علامت مثبت ، یک علامت پیش فرض است لذا بیتهای آن صفر است اما علامت منفی بیتهای آن عدد یک است .

مانند شکل زیر



در شکل بالا دقت کنید که چگونه عدد منفی 10 جایگزین و تبدیل شده است.

در مثال زیر نیز یک داده از نوع **byte** که شامل مقادیری بین **255-0** است به یک عدد صحیح **(int)** تبدیل شده است.

```
byte number1 = 5;

int number2 = number1;
```

در مثال بالا مقدار **number1** برابر **5** است در نتیجه متغیر **number2** که یک متغیر از نوع صحیح است می تواند مقدار **number1** را در خود ذخیره کند چون نوع صحیح از نوع بایت بزرگتر است، پس متغیر **number1** که یک متغیر از نوع بایت است می تواند به طور ضمنی به **number2** که یک متغیر از نوع صحیح است تبدیل شود.

اما عکس مثال بالا صادق نیست.

```
int number1 = 5;

byte number2 = number1;
```

در این مورد ما با خطا مواجه می شویم

اگر چه مقدار **5** متغیر **number1** در محدوده مقادیر **byte** یعنی اعداد بین **255-0** قرار دارد اما متغیر ی از نوع بایت حافظه کمتری نسبت به متغیری از نوع صحیح اشغال می کند، نوع **byte** شامل **8** بیت یا **8** رقم دودویی است در حالی که نوع **int** شامل **32** بیت یا رقم باینری است،یک عدد باینری عددی متشکل از **0** و **1** است،برای مثال عدد **5** در کامپیوتر به عدد باینری **101** ترجمه می شود، بنابراین وقتی ما عدد **5** را در یک متغیر از نوع بایت ذخیره می کنیم عددی به صورت زیر نمایش داده می شود :

```
00000101
```

و وقتی آن را در یک متغیر از نوع صحیح ذخیره می کنیم به صورت زیر مایش داده می شود :

```
00000000000000000000000000000101
```

نکته دیگری که نباید فراموش شود این است که شما نمی توانید اعداد با ممیز اعشار را به یک نوع **int** تبدیل کنید.

```
double number1 = 5.25;

int number2 = number1; //Error
```

میتوان یک نوع کاراکتر را به نوع **ushort** تبدیل کرد چون هر دو دارای طیف مشابهی از اعداد هستند، گرچه هر یک از آنها کاملا متفاوت توسط کامپایلر ترجمه می شوند، نوع **char** به عنوان یک کاراکتر و نوع **ushort** به عنوان یک عدد ترجمه می شود.

```
char charVar = 'c';
ushort shortVar = charVar;

Console.WriteLine(charVar);
Console.WriteLine(shortVar);
```

```
c
99
```

تبدیلاتی که کامپایلر به صورت ضمنی می تواند انجام دهد در جدول زیر آمده است :

Source Type	Can Safely Be Converted To
byte	short, ushort, int, uint, long, ulong, float, double, decimal
sbyte	short, int, long, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
ulong	float, double, decimal
float	double
char	ushort, int, uint, long, ulong, float, double, decimal

نکته ای دیگر که معمولا ابهام بر انگیز است تعین نوع داده است، برای مثال ما چطور بدانیم که مثلا عدد **7** از نوع **int**، **uint**، **long** یا **ulong** است؟ برای این کار باید کاراکترهایی را به انتهای اعداد اضافه کنیم.

```
uint number1 = 7U;
long number2 = 7L;
```

```
ulong number3 = 7UL;
```

در حالت پیشفرض و بدون قرار دادن کاراکتر در انتهای عدد کامپایلر عدد را از نوع صحیح **(int)** در نظر می گیرد.

همچنین در حالت پیشفرض کامپایلر اعداد دسیمال **(decimal)** را اعداد **double** در نظر می گیرد.

شما می توانید برای نشان دادن اعداد اعشاری **float** از کاراکتر **F** و برای نشان دادن اعداد دسیمال از کاراکتر **M** استفاده کنید.

```
double number1 = 1.23;
float number2 = 1.23F;
decimal number3 = 1.23M
```

تبدیل صریح

تبدیل صریح نوعی تبدیل است که برنامه را مجبور می کند که یک نوع داده را به نوعی دیگر تبدیل کند اگر این نوع تبدیل از طریق تبدیل ضمنی انجام نشود،در هنگام استفاده از این تبدیل باید دقت کرد چون در این نوع تبدیل ممکن است مقادیر اصلاح یا حذف شوند،ما میتوانیم این عملیات را با استفاده از **Cast** انجام دهیم.

Cast فقط نام دیگر تبدیل صریح است و دستور آن به صورت زیر است :

```
datatypeA variableA = value;
datatypeB variableB = (datatypeB)variableA;
```

همانطور که قبلا مشاهده کردید نوع **int** را نتوانستیم به نوع **byte** تبدیل کنیم اما اکنون با استفاده از عمل **Cast** این تبدیل انجام خواهد شد :

```
int number1 = 5;

byte number2 = (byte)number1;
```

حال اگر برنامه را اجرا کنید با خطا مواجه نخواهید شد.

همانطور که پیشتر اشاره شد ممکن است در هنگام تبدیلات مقادیر اصلی تغییر کنند،برای مثال وقتی که یک عدد با ممیز اعشار مثلا از نوع **double** را به یک نوع **int** تبدیل می کنیم مقدار اعداد بعد از ممیز از بین می روند :

```
double number1 = 5.25;

int number2 = (int)number1;

Console.WriteLine(number2)
```

```
5
```

خروجی کد بالا عدد **5** است چون نوع داده ای **int** نمی تواند مقدار اعشار بگیرد.

حالت دیگر را تصور کنید،اگر شما بخواهید یک متغیر را که دارای مقداری بیشتر از محدوده متغیر مقصد هست تبدیل کنید چه اتفاقی می افتد؟ مانند تبدیل زیر که می خواهیم متغیر **number1** را که دارای مقدار **300** است را به نوع بایت تبدیل کنیم که محدود اعداد بین **255-** **0** را پوشش می دهد.

```
int number1 = 300;

byte number2 = (byte)number1;

Console.WriteLine("Value of number2 is {0}.", number2);
```

```
Value of number2 is 44.
```

خروجی کد بالا عدد **44** است. **Byte** فقط می تواند شامل اعداد **0** تا **255** باشد و نمی تواند مقدار **300** را در خود ذخیره کند،حال می خواهیم ببینیم که چرا به جای عدد **300** ما عدد **44** را در خروجی می گیریم. این کار به تعداد بیتها بستگی دارد،یک **byte** دارای **8** بیت است درحالی که **int** دارای **32** بیت است. حال اگر به مقدار باینری **2** عدد توجه کنید متوجه می شوید که چرا خروجی عدد **44** است.

```
300 = 00000000000000000000000100101100
255 =                         11111111
 44 =                         00101100
```

خروجی بالا نشان می دهد که بیشترین مقدار **byte** که عدد **255** است می تواند فقط شامل **8** بیت باشد **(11111111)** بنابراین فقط **8** بیت اول مقدار **int** به متغیر **byte** انتقال می یابد که شامل **(00101100)** یا عدد **44** در مبنای **10** است.

قرار ندادن یک مقدار مناسب در داخل یک متغیر باعث ایجاد یک سرریز **(overflow)** می شود. یک مورد آن سرریز ریاضی نام دارد که در مثال زیر مشاهده می کنید :

```
byte sum = (byte)(150 + 150);
```

گرچه در این تبدیل ما داده هایی را از دست می دهیماما کامپایلر کد مار را قبول می کند،برای اینکه برنامه هنگام وقوع سرریز پیغام خطا بدهد می توان از کلمه کلیدی **checked** استفاده کرد.

```
int number1 = 300;

byte number2 = checked((byte)number1);

Console.WriteLine("Value of number2 is {0}.", number2)
```

```
Unhandled Exception: System.OverflowException: Arithmetic operation resulted in an
overflow ...
```

برنامه پیغام **System.OverflowException** که به زبان ساده نشان دهند وقوع خطاست،در نتیجه شما می توانید از اجرای برنامه جلوگیری کنید.

تبدیل با استفاده از کلاس Convert

.NET Framework دارای یک کلاس استاتیک (در مورد کلمه استاتیک بعدا توضیح می دهیم)است که می توان از آن برای تبدیل مقادیر از نوعی به نوع دیگر استفاده کرد.این کلاس به نوبه خود دارای متدهایی برای تبدیل انواع داده به یکدیگر می باشد.

در جدول زیر متدها ذکر شده اند :

Command	Result
Convert.ToBoolean(val)	val converted to bool
Convert.ToByte(val)	val converted to byte
Convert.ToChar(val)	val converted to char
Convert.ToDecimal(val)	val converted to decimal
Convert.ToDouble(val)	val converted to double
Convert.ToInt16(val)	val converted to short
Convert.ToInt32(val)	val converted to int
Convert.ToInt64(val)	val converted to long
Convert.ToSByte(val)	val converted to ushort
Convert.ToSingle(val)	val converted to float
Convert.ToString(val)	val converted to string
Convert.ToUInt16(val)	val converted to ushort
Convert.ToUInt32(val)	val converted to uint
Convert.ToUInt64(val)	val converted to ulong

در برنامه زیر یک نمونه از تبدل متغیر ها با استفاده از کلاس Convert و متدهای آن نمایش داده شده است :

```
double x = 9.99;
int convertedValue = Convert.ToInt32(x);

Console.WriteLine("Original value is: " + x);
Console.WriteLine("Converted value is: " + convertedValue);
```

```
Original value is: 9.99
Converted value is: 10
```

مقدار val هر نوع داده ای می تواند باشد اما باید مطمئن شد که به نوع داده ای مورد نظر تبدیل شود.

عبارات و عملگرها

ابتدا با دو کلمه آشنا شوید :

عملگر : نمادهایی هستند که اعمال خاص انجام می دهند.

عملوند : مقادیری که عملگرها بر روی آنها عملی انجام می دهند.

مثلا X+Y: یک عبارت است که در آن X و Y عملوند و علامت + عملگر به حساب می آیند.

زبانهای برنامه نویسی جدید دارای عملگرهایی هستند که از اجزاء معمول زبان به حساب می آیند. سی شارپ دارای عملگرهای مختلفی از جمله عملگرهای ریاضی ،تخصیصی ، مقایسه ای ، منطقی و بیتی می باشد. از عملگرهای ساده ریاضی می توان به عملگر جمع و تفریق اشاره کرد.

سه نوع عملگر در سی شارپ وجود دارد :

- **یگانی (Unary)– به یک عملوند نیاز دارد**
- **دودویی(Binary) – به دو عمولوند نیاز دارد**
- **سه تایی (Ternary)– به سه عملوند نیاز دارد**

انواع مختلف عملگر که در ای بخش مورد بحث قرار می گیرند عبارتند از :

- **عملگرهای ریاضی**
- **عملگرهای تخصیصی**
- **عملگرهای مقایسه ای**
- **عملگرهای منطقی**
- **عملگرهای بیتی**

عملگرهای ریاضی

سی شارپ از عملگرهای ریاضی برای انجام محاسبات استفاده می کند.

جدول زیر عملگرهای ریاضی سی شارپ را نشان می دهد :

عملگر	دسته	مثال	نتیجه
+	Binary	var1 = var2 + var3;	Var1 برابر است با حاصل جمع var2 وvar3
-	Binary	var1 = var2 – var3;	Var1 برابر است با حاصل تفریق var2 وvar3
*	Binary	var1 = var2 * var3;	Var1 برابر است با حاصلضرب var2 در var3
/	Binary	var1 = var2 / var3;	Var1 برابر است با حاصل تقسیم var2 برvar3
%	Binary	var1 = var2 % var3;	Var1 برابر است با باقیمانده تقسیم var2 وvar3
+	Unary	var1 = +var2;	Var1 برابر است با مقدار var2
-	Unary	var1 = -var2	Var1 برابر است با مقدار var2 ضربدر 1-

در مثال بالا از نوع عددی استفاده شده است،اما استفاده از عملگرهای ریاضی برای نوع رشته ای نتیجه متفاوتی دارد،همچنین در جمع دو کاراکتر کامپایلر معادل عددی آنها را نشان می دهد. اگر از عملگر + برای رشته ها استفاده کنیم دو رشته را با هم ترکیب کرده و به هم می چسباند. دیگر عملگرهای سی شارپ عملگرهای کاهش و افزایش هستند،این عملگرها مقدار 1 را از متغیر ها کم یا به آنها اضافه می کنند.

از این متغیر ها اغلب در حلقه ها استفاده می شود :

عملگر	دسته	مثال	نتیجه
++	Unary	var1 = ++var2;	مقدار var1 برابر است با var2 بعلاوه 1
--	Unary	var1 = --var2;	مقدار var1 برابر است با var2 منهای 1
++	Unary	var1 = var2++;	مقدار var1 برابر است با var2 به متغیر var2 یک واحد اضافه می شود
--	Unary	var1 = var2--;	مقدار var1 برابر است با var2 از متغیر var2 یک واحد کم می شود

به این نکته توجه داشته باشید که محل قرار گیری عملگر در نتیجه محاسبات تاثیر دارد.اگر عملگر قبل از متغیر var2 بیاید افزایش یا کاهش var1 و var2اتفاق می افتند

```
using System;

    class Program
    {
        static void Main(string[] args)
        {
            int x, y;

            y = 5;

            x = --y;
            Console.WriteLine("x={0}{1}y={2}", x, "  ", y);
        }
    }
```

```
x=4  y=4
```

.چنانچه عملگرها بعد از متغیر var2 قرار بگیرند ابتدا var1 برابر var2 می شود و سپس متغیر var2 افزایش یا کاهش می یابد

```
using System;

    class Program
    {
        static void Main(string[] args)
        {
            int x, y;

            y = 5;

            x = y--;
            Console.WriteLine("x={0}{1}y={2}", x, "  ", y);
        }
    }
```

```
x=5  y=4
```

حال می توانیم با ایجاد یک برنامه نحوه عملکرد عملگرهای ریاضی در سی شارپ را یاد بگیریم :

```
using System;

public class Program
{
    public static void Main()
    {
        //Variable declarations
        int num1, num2;
        string msg1, msg2;

        //Assign test values
        num1 = 5;
        num2 = 3;

        //Demonstrate use of mathematical operators
        Console.WriteLine("The sum of {0} and {1} is {2}.", num1, num2, (num1 +
num2));
        Console.WriteLine("The difference of {0} and {1} is {2}.", num1, num2,
                          (num1 - num2));
        Console.WriteLine("The product of {0} and {1} is {2}.", num1, num2,
                          (num1 * num2));
        Console.WriteLine("The quotient of {0} and {1} is {2:F2}.", num1, num2,
                         ((double)num1 / num2));
        Console.WriteLine("The remainder of {0} divided by {1} is {2}", num1, num2,
                         (num1 % num2));

        //Demonstrate concatenation on strings using the + operator
        msg1 = "Hello ";
        msg2 = "World!";
        Console.WriteLine(msg1 + msg2);
    }
}
```

نتیجه

```
The sum of 5 and 3 is 8.
The difference of 5 and 3 is 2.
The product of 5 and 3 is 15.
The quotient of 5 and 3 is 1.67.
The remainder of 5 divided by 3 is 2
Hello World!
```

برنامه بالا نتیجه هر عبارت را نشان می دهد.

در این برنامه از متد **Writeline()** برای نشان دادن نتایج در سطرهای متفاوت استفاده شده است.

در این مثال با یک نکته عجیب مواجه می شویم و آن حاصل تقسیم دو عدد صحیح است،وقتی که دو عدد صحیح را بر هم تقسیم کنیم حاصل باید یک عدد صحیح و فاقد بخش کسری باشد،اما همانطور که مشاهده می کنید اگر فقط یکی از اعداد را به نوع اعشاری **double** تبدیل کنیم **(**در مثال می بینید**)** حاصل به صورت اعشار نشان داده می شود.

برای اینکه ارقام کسری بعد از عدد حاصل دو رقم باشند از {2:F2} استفاده می کنیم. **F** به معنای تعیین کردن می باشد و در این جا بدین معناست که عدد را تا دو رقم اعشار نمایش بده. چون خطوط کد طولانی هستند آنها را در دو خط می نویسیم. سی شارپ خط جدید و فاصله و فضای خالی را نادیده می گیرد.

در خط 29 مشاهده می کنید که دو رشته به وسیله عملگر + به هم متصل شده اند.
نتیجه استفاده از عملگر + برای چسباندن دو کلمه " Hello" و "!World" رشته "!Hello World" خواهد بود. به فاصله های خالی بعد از اولین کلمه توجه کنید اگر آنها را حذف کنید از خروجی برنامه نیز حذف می شوند.

عملگرهای تخصیصی (جایگزینی)

نوع دیگر از عملگرهای سی شارپ عملگرهای جایگزینی نام دارند.این عملگرها مقدار متغیر سمت راست خود را در متغیر سمت چپ قرار می دهند.

جدول زیر انواع عملگرهای تخصیصی در سی شارپ را نشان می دهد:

عملگر	مثال	نتیجه
=	var1 = var2;	مقدار var1 با است با مقدار var2
+=	var1 += var2;	مقدار var1 با است با حاصل جمع var1 وvar2
-=	var1 -= var2;	مقدار var1 با است با حاصل تفریق var1 وvar2
*=	var1 *= var2;	مقدار var1 با است با حاصل ضرب var1 درvar2
/=	var1 /= var2;	مقدار var1 با است با حاصل تقسیم var1 برvar2
%=	var1 %= var2;	مقدار var1 با است با باقیمانده تقسیم var1 برvar2

از عملگر **+=** برای اتصال دو رشته نیز می توان استفاده کرد.استفاده از این نوع عملگرها در واقع یک نوع خلاصه نویسی در کد است.

مثلا شکل اصلی کد var1 += var2 به صورت var1 = var1 + var2 می باشد. این حالت کدنویسی زمانی کارایی خود را نشان می دهد که نام متغیرها طولانی باشد.

برنامه زیر چگونگی استفاده از عملگرهای تخصیصی و تاثیر آنها را بر متغیر ها نشان می دهد.

```
using System;

public class Program
{
    public static void Main()
    {
        int number;

        Console.WriteLine("Assigning 10 to number...");
        number = 10;
        Console.WriteLine("Number = {0}", number);

        Console.WriteLine("Adding 10 to number...");
        number += 10;
        Console.WriteLine("Number = {0}", number);

        Console.WriteLine("Subtracting 10 from number...");
        number -= 10;
        Console.WriteLine("Number = {0}", number);
    }
}
```

```
Assigning 10 to number...
Number = 10
Adding 10 to number...
Number = 20
Subtracting 10 from number...
Number = 10
```

در برنامه از 3 عملگر تخصیصی استفاده شده است، ابتدا یک متغیر و مقدار 10 با استفاده از عملگر = به آن اختصاص داده شده است،سپس به آن با استفاده از عملگر =+ مقدار 10 اضافه شده است،و در آخر به وسیله عملگر =- عدد 10 از آن کم شده است.

عملگرهای مقایسه ای

از عملگرهای مقایسه ای برای مقایسه مقادیر استفاده می شود،نتیجه این مقادیر یک مقدار بولی(منطقی) است.
این عملگرها اگر نتیجه مقایسه دو مقدار درست باشد مقدار **true** و اگر نتیجه مقایسه شتباه باشد مقدار **false** را نشان می دهند.
این عملگرها به طور معمول در دستورات شرطی به کار می روند به ای ترتیب که باعث ادامه یا توقف دستور شرطی می شوند.

جدول زیر عملگر های مقایسه ای در سی شارپ را نشان می دهد:

عملگر	دسته	مثال	نتیجه
==	Binary	var1 = var2 == var3	var1 در صورتی true است که مقدار var2 با مقدار var3 برابر باشد در غیر اینصورت false است
!=	Binary	var1 = var2 != var3	var1 در صورتی true است که مقدار var2 با مقدار var3 برابر نباشد در غیر اینصورت false است
<	Binary	var1 = var2 < var3	var1 در صورتی true است که مقدار var2 کوچکتر از var3 مقدار باشد در غیر اینصورت false است
>	Binary	var1 = var2 > var3	var1 در صورتی true است که مقدار var2بزرگتر ازمقدارvar3 باشد در غیر اینصورت false است
<=	Binary	var1 = var2 <= var3	var1 در صورتی true است که مقدار var2 کوچکتر یا مساوی مقدار var3 باشد در غیر اینصورت false است
>=	Binary	var1 = var2 >= var3	var1 در صورتی true است که مقدار var2 بزرگتر یا مساوی var3 مقدار باشد در غیر اینصورت false است

برنامه زیر نحوه عملکرد ای عملگرها را نشان می دهد :

```
using System;

namespace ComparisonOperators
{
    class Program
    {
        static void Main()
        {
            int num1 = 10;
            int num2 = 5;

            Console.WriteLine("{0} == {1} : {2}", num1, num2, num1 == num2);
            Console.WriteLine("{0} != {1} : {2}", num1, num2, num1 != num2);
            Console.WriteLine("{0} <  {1} : {2}", num1, num2, num1 < num2);
            Console.WriteLine("{0} >  {1} : {2}", num1, num2, num1 > num2);
            Console.WriteLine("{0} <= {1} : {2}", num1, num2, num1 <= num2);
            Console.WriteLine("{0} >= {1} : {2}", num1, num2, num1 >= num2);
        }
    }
}
```

```
10 == 5 : False
10 != 5 : True
10 <  5 : False
10 >  5 : True
10 <= 5 : False
10 >= 5 : True
```

در مثال بالا ابتدا دو متغیر را که می خواهیم با هم مقایسه کنیم را ایجاد کرده و به آنها مقادیری اختصاص می دهیم،سپس با استفاده از یک عملگر مقایسه ای آنها را با هم مقایسه کرده و نتیجه را چاپ می کنیم.

به این نکته توجه کنید که هنگام مقایسه دو متغیر از عملگر == به جای عملگر = باید استفاده شود.

عملگر = عملگر تخصیصی است و در عبارتی مانند x = y مقدار y را در به x اختصاص می دهد.

عملگر == عملگر مقایسه ای است که دو مقدار را با هم مقایسه می کند مانند x==y و اینطور خوانده می شود x برابر است با y .

عملگرهای منطقی

عملگرهای منطقی بر روی عبارات منطقی عمل می کنند و نتیجه آنها نیز یک مقدار بولی است. از این عملگرها اغلب برای شرطهای پیچیده استفاده می شود. همانطور که قبلا یاد گرفتید مقادیر بولی می توانند false یا true باشند.

فرض کنید که var2 و var3 دو مقدار بولی هستند.

عملگر	نام	دسته	مثال
&&	AND منطقی	Binary	var1 = var2 && var3;
||	OR منطقی	Binary	var1 = var2 || var3;
!	NOT منطقی	Unary	var1 = !var1;

- **عملگر منطقی AND (&&)**

اگر مقادیر دو طرف عملگر AND ، true باشند عملگر AND مقدار true را بر می گرداند. در غیر اینصورت اگر یکی از مقادیر یا هر دوی آنها falseباشند مقدار false را بر می گرداند.

در زیر جدول درستی عملگر AND نشان داده شده است :

X	Y	X && Y
true	true	true
true	false	false
false	true	false
false	false	false

برای درک بهتر تاثیر عملگر AND یاد آوری می کنم که این عملگر فقط در صورتی مقدار true را نشان می دهد که هر دو عملوند مقدارشان true باشد.در غیر اینصورت نتیجه تمام ترکیبهای بعدی false خواهد شد.
استفاده از عملگر AND مانند استفاده از عملگرهای مقایسه ای است.
به عنوان مثال نتیجه عبارت زیر درست (true) است اگر سن (age) بزرگتر از 18 و salary کوچکتر از 1000 باشد.

```
result = (age > 18) && (salary < 1000);
```

عملگر AND زمانی کارامد است که ما با محدود خاصی از اعداد سرو کار داریم.

مثلا عبارت 100 => x => 10 بدین معنی است که x می تواند مقداری شامل اعداد 10 تا 100 را بگیرد. حال برای انتخاب اعداد خارج از این محدوده می توان از عملگر منطقی AND به صورت زیر استفاده کرد.

```
inRange = (number <= 10) && (number >= 100);
```

- عملگر منطقی (||) OR

اگر یکی یا هر دو مقدار دو طرف عملگر OR ،درست (true) باشد،عملگر OR مقدار true را بر می گرداند.

جدول درستی عملگر OR در زیر نشان داده شده است :

X	Y	X || Y
true	true	true
true	false	true
false	true	true
false	false	false

در جدول بالا مشاهده می کنید که عملگر OR در صورتی مقدار false را بر میگرداند که مقادیر دو طرف آن false باشند.
کد زیر را در نظر بگیرید،نتیجه این کد در صورتی درست (true) است که رتبه نهایی دانش آموز (finalGrade) بزرگتر از 75 یا یا نمره نهایی امتحان آن 100 باشد.

```
isPassed = (finalGrade >= 75) || (finalExam == 100);
```

- عملگر منطقی (!)NOT

برخلاف دو اپراتور OR و AND عملگر منطقی NOT یک عملگر یگانی است و فقط به یک عملوند نیاز دارد.
این عملگر یک مقدار یا اصطلاح بولی را نفی می کند.
مثلا اگر عبارت یا مقدار true باشد آنرا false و اگر false باشد آنرا true می کند
جدول زیر عملکرد اپراتور NOT را نشان می دهد :

X	!X
true	false
false	true

نتیجه کد زیر در صورتی درست است که age (سن) بزرگتر یا مساوی 18 نباشد.

```
isMinor = !(age >= 18);
```

عملگرهای بیتی

عملگرهای بیتی به شما اجازه می دهند که شکل باینری انواع داده ها را دستکاری کنید،برای درک بهتر این درس توصیه می شود که شما سیستم باینری و نحوه تبدیل اعداد اعشاری به باینری را یاد بگیرید،در سیستم باینری (دودویی) که کامپیوتر از آن استفاده می کند وضعیت هر چیز یا خاموش است یا روشن،برای نشان دادن حالت روشن از عدد 1 و برای نشان دادن حالت خاموش از عدد 0 استفاده می شود،بنابراین اعداد باینری فقط می توانند صفر یا یک باشند،اعداد باینری را اعداد در مبنای 2 و اعداد اعشاری را اعداد در مبنای 10 می گویند، یک بیت نشان دهنده یک رقم باینری است و هر بایت نشان دهنده 8 بیت است، به عنوان مثال برای یک داده از نوع int به 32 بیت یا 8 بایت فضا برای ذخیره آن نیاز داریم،این بدین معناست که اعداد از 32 رقم 0 و 1 برای ذخیره استفاده می کنند.
برای مثال عدد 100 وقتی به عنوان یک متغیر از نوع int ذخیره می شود در کامپیوتر به صورت زیر خوانده می شود :

```
00000000000000000000000001100100
```

عدد 100 در مبنای ده معادل عدد 1100100 در مبنای 2 است،در اینجا 7 رقم سمت راست نشان دهنده عدد 100 در مبنای 2 است و مابقی صفرهای سمت راست برای پر کردن بیتهایی است که عدد از نوع int نیاز دارد،به این نکته توجه کنید که اعداد باینری از سمت راست به چپ خوانده می شوند، عملگرهای بیتی سی شارپ در جدول زیر نشان داده شده اند :

عملگر	نام	دسته	مثال
&	بیتی AND	Binary	x = y & z;
|	بیتی OR	Binary	x = y | z;
^	بیتی XOR	Binary	x = y ^ z;
~	بیتی NOT	Unary	x = ~y;
&=	بیتی AND Assignment	Binary	x &= y;
|=	بیتی OR Assignment	Binary	x |= y;
^=	بیتی XOR Assignment	Binary	x ^= y;

- عملگر بیتی AND(&)

عملگر بیتی ANDمانند کاری شبیه عملگر منطقی AND انجام می دهد با این تفاوت که این عملگر بر روی بیتها کار می کند،اگر مقادیر دو طرف آن 1 باشد مقدار 1 را بر می گرداند و اگر یکی یا هر دو طرف آن صفر باشد مقدار صفر را بر می گرداند.

جدول درستی عمگر بیتی AND در زیر آمده است:

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

در زیر نحوه استفاده از عملگر بیتی AND آمده است :

```
int result = 5 & 3;

Console.WriteLine(result);
```

```
1
```

همانطور که در مثال بالا مشاهده می کنید نتیجه عملکرد عملگر AND بر روی دو مقدار 5 و 3 عدد یک می شود،اجازه بدهید ببینیم که چطور این نتیجه را به دست می آید:

```
5:  00000000000000000000000000000101
3:  00000000000000000000000000000011
-----------------------------------
1:  00000000000000000000000000000001
```

ابتدا دو عدد 5 و 3 به معادل باینری شان تبدیل می شوند، از آنجاییکه هر عدد صحیح (int) 32 بیت است از صفر برای پر کردن بیتهای خالی استفاده می کنیم، با استفاده از جدول درستی عملگر بیتی ANDمی توان فهمید که چرا نتیجه عدد یک می شود.

- عملگر بیتی (|)OR

اگر مقادیر دو طرف عملگر بیتی **OR** هر دو صفر باشند نتیجه صفر در غیر اینصورت **1** خواهد شد.جدول درستی این عملگر در زیر آمده است :

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

نتیجه عملگر بیتی **OR** در صورتی صفر است که عملوند های دو طرف آن صفر باشند.اگر فقط یکی از دو عملوند یک باشد نتیجه یک خواهد شد. به مثال زیر توجه کنید :

```
int result = 7 | 9;

Console.WriteLine(result);
```

```
15
```

وقتی که از عملگر بیتی **OR** برای دو مقدار در مثال بالا **(7 و 9)** استفاده می کنیم نتیجه **15** می شود.حال بررسی می کنیم که چرا این نتیجه به دست آمده است؟

```
7: 00000000000000000000000000000111
9: 00000000000000000000000000001001
---------------------------------
15:00000000000000000000000000001111
```

با استفاده از جدول درستی عملگر بیتی **OR** می توان نتیجه استفاده از این عملگر را تشخیص داد.عدد **1111** باینری معادل عدد **15** صحیح است.

- عملگر بیتی XOR(^)

جدول درستی این عملگر در زیر آمده است :

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

در صورتیکه عملوندهای دو طرف این عملگر هر دو صفر یا هر دو یک باشند نتیجه صفر در غیر اینصورت نتیجه یک می شود.در مثال زیر تاثیر عملگر بیتی XOR را بر روی دو مقدار مشاده می کنید :

```
int result = 5 ^ 7;

Console.WriteLine(result);
```

```
2
```

در زیر معادل باینری اعداد بالا (5 و 7) نشان داده شده است.

```
5: 00000000000000000000000000000101
7: 00000000000000000000000000000111
-----------------------------------
2: 00000000000000000000000000000010
```

با نگاه کردن به جدول درستی عملگر بیتی XOR می توان فهمید که چرا نتیجه عدد 2 می شود.

- **عملگر بیتی NOT(~)**

این عملگر یک عملگر یگانی است و فقط به یک عملوند نیاز دارد،در زیر جدول درستی این عملگر آمده است:

X	NOT X
1	0
0	1

عملگر بیتی NOT مقادیر بیتها را معکوس می کند،در زیر چگونگی استفاده از این عملگر آمده است :

```
int result = ~7;

Console.WriteLine(result);
```

به نمایش باینری مثال بالا که در زیر نشان داده شده است توجه نمایید.

```
7: 00000000000000000000000000000111
-----------------------------------
-8: 11111111111111111111111111111000
```

مثالهایی از عملگرهای بیتی

فرض کنید که از یک سبک خاص فونت در برنامه تان استفاده کنید،کدهای مربوط به هر سبک هم در جدول زیر آمده است :

سبک	کد
Regular	0
Bold	1
Italic	2
Underline	4
Strikeout	8

توجه کنید که مقدار اولیه صفر بدین معنی است که می خواهید از سبک regular (عادی) استفاده کنید.

```
int fontStyle = 0;
```

برای نشان دادن فونتها به صورت کلفت (Bold) از عملگر بیتی OR استفاده می شود.توجه کنید که برای فونت Bold باید کد 1 را به کار برید.

```
fontStyle = fontStyle | 1;
```

برای استفاده از سبک Italic باید از عملگر بیتی OR و کد 2 استفاده شود.

```
fontStyle |= 2;
```

برای استفاده از سایر سبک ها می توان به روش های ذکر شده در بالا عمل کرد و فقط کدها را جایگزین کنید.

اگر بخواهید یک سبک جدید ایجاد کنید که ترکیبی از چند سبک باشد می توانید به سادگی عملگر بیتی OR را در بین هر سبک فونت قرار دهید مانند مثال زیر :

```
fontStyle = 1 | 2 | 4 | 8;
```

- **عملگر بیتی تغییر مکان (shift)**

این نوع عملگرها به شما اجازه می دهند که بیتها را به سمت چپ یا راست جا به جا کنید.دو نوع عملگر بیتی تغییر مکان وجود دارد که هر کدام دو عملوند قبول می کنند.عملوند سمت چپ این عملگرها حالت باینری یک مقدار و عملوند سمت راست تعداد جابه جاییبیت ها را نشان می دهد.

عملگر	نام	دسته	مثال
<<	تغییر مکان به سمت چپ	Binary	x = y << 2;
>>	تغییر مکان به سمت راست	Binary	x = y >> 2;

- عملگر تغییر مکان به سمت چپ

این عملگر بیتهای عملوند سمت چپ را به تعداد n مکان مشخص شده توسط عملوند سمت راست، به سمت چپ منتقل می کند.
به عنوان مثال :

```
int result = 10 << 2;

Console.WriteLine(result);
```

```
40
```

در مثال بالا ما بیتهای مقدار 10 را دو مکان به سمت چپ منتقل کرده ایم، حال بیایید تاثیر این انتقال را بررسی کنیم :

```
10: 00000000000000000000000000001010
----------------------------------
40: 00000000000000000000000000101000
```

مشاهده می کنید که همه بیت ها به اندازه دو واحد به سمت چپ منتقل شده اند.در این انتقال دو صفر از صفرهای سمت چپ کم می شود و در عوض دو صفر به سمت راست اضافه می شود.

- عملگر تغییر مکان به سمت راست

این عملگر شبیه به عمگر تغییر مکان به سمت چپ است با این تفاوت که بیت ها را به سمت راست جا به جا می کند.

به عنوان مثال :

```
int result = 100 >> 4;

Console.WriteLine(result);
```

```
6
```

با استفاده از عملگرتغییر مکان به سمت راست بیت های مقدار 100 را به اندازه 4 واحد به سمت چپ جا به جا می کنیم.اجازه بدهید تاثیر این جا به جایی را مورد بررسی قرار دهیم :

```
100: 00000000000000000000000001100100
----------------------------------
  6: 00000000000000000000000000000110
```

هر بیت به اندازه 4 واحد به سمت راست منتقل می شود،بنابراین 4 بیت اول سمت راست حذف شده و چهار صفر به سمت چپ اضافه می شود.

تقدم عملگرها

تقدم عملگرها مشخص می کند که در محاسباتی که بیش از دو عملوند دارند ابتدا کدام عملگر اثرش را اعمال کند،عملگر ها در سی شارپ در محاسبات دارای حق تقدم هستند،به عنوان مثال :

```
number = 1 + 2 * 3 / 1;
```

اگر ما حق تقدم عملگرها را رعایت نکنیم و عبارت بالا را از سمت چپ به راست انجام دهیم نتیجه 9 خواهد شد (1+2=3 سپس 3×3=9 و در آخر 9/1=9)،اما کامپایلر با توجه به تقدم عملگرها محاسبات را انجام می دهد،برای مثال عمل ضرب و تقسیم نسبت به جمع و تفریق تقدم دارند،بنابراین در مثال فوق ابتدا عدد 2 ضربدر 3 و سپس نتیجه آنها تقسیم بر 1 می شود که نتیجه 6 به دست می آید،در آخر عدد 6 با 1 جمع می شود و عدد 7 حاصل می شود.

در جدول زیر تقدم برخی از عملگرهای سی شارپ آمده است :

Precedence	Operators
بالاترین	++, –, (used as prefixes); +, – (unary)
	*, /, %
	+, -
	<<, >>
	<, >, <=, >=
	==, !=
	&
	^
	|
	&&
	||
	=, *=, /=, %=, +=, -=
پایین ترین	++, – (used as suffixes)

ابتدا عملگرهای با بالاترین و سپس عملگرهای با پایین ترین حق تقدم در محاسبات تاثیر می گذارند،به این نکته توجه کنید که تقدم عملگرها ++ و –به مکان قرارگیری آنها بستگی دارد (در سمت چپ یا راست عملوند باشند).

به عنوان مثال :

```
int number = 3;

number1 = 3 + ++number; //results to 7
number2 = 3 + number++; //results to 6
```

در عبارت اول ابتدا به مقدار number یک واحد اضافه شده و 4 می شود و سپس مقدار جدید با عدد 3 جمع می شود و در نهایت عدد 7 به دست می آید،در عبارت دوم مقدار عددی 3 به مقدار number اضافه می شود و عدد 6 به دست می آید،سپس این مقدار در متغیر number2 قرار می گیرد، و در نهایت مقدار number به 4 افزایش می یابد.

برای ایجاد خوانایی در تقدم عملگرها و انجام محاسباتی که در آنها از عملگرهای زیادی استفاده می شود از پرانتز استفاده می کنیم :

```
number = ( 1 + 2 ) * ( 3 / 4 ) % ( 5 - ( 6 * 7 ));
```

در مثال بالا ابتدا هر کدام از عباراتی که داخل پرانتز هستند مورد محاسبه قرار می گیرند،به نکته ای در مورد عبارتی که در داخل پرانتز سوم قرار دارد توجه کنید،در این عبارت ابتدا مقدار داخلی ترین پرانتز مورد محاسبه قرار می گیرد یعنی مقدار **6** ضربدر **7** شده و سپس از **5** کم می شود.

اگر دو یا چند عملگر با حق تقدم یکسان موجود باشد ابتدا باید هر کدام از عملگرها را که در ابتدای عبارت می آیند مورد ارزیابی قرار دهید.

به عنوان مثال :

```
number = 3 * 2 + 8 / 4;
```

هر دو عملگر * و / دارای حق تقدم یکسانی هستند، بنابر این شما باید از چپ به راست آنها را در محاسبات تاثیر دهید،یعنی ابتدا **3** را ضربدر **2** می کنید و سپس عدد **8** را بر **4** تقسیم می کنید، در نهایت نتیجه دو عبارت را جمع کرده و در متغیر **number** قرار می دهید.

گرفتن ورودی از کاربر

چارچوب دات نت تعدادی متد برای گرفتن ورودی از کاربر در اختیار شما قرار می دهد،حال می خواهیم در باره متد **ReadLine()** یکی دیگر از متدهای کلاس **Console** بحث کنیم که یک مقدار رشته ای را از کاربر دریافت می کند،متد **ReadLine()** فقط مقدار رشته ای را که توسط کاربر نوشته می شود را بر می گرداند،همانطور که از نام این متد پیداست،تمام کاراکترهایی را که شما در محیط کنسول تایپ می کنید تا زمانی که دکمه **enter** را می زنید می خواند،هر چه که در محیط کنسول تایپ می شود از نوع رشته است،برای تبدیل نوع رشته به انواع دیگر می توانید از کلاس **convert** و متدهای آن استفاده کنید.

به برنامه زیر توجه کنید :

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           string name;
8           int age;
9           double height;
10
11          Console.Write("Enter your name: ");
12          name = Console.ReadLine();
13          Console.Write("Enter your age: ");
14          age = Convert.ToInt32(Console.ReadLine());
15          Console.Write("Enter your height: ");
16          height = Convert.ToDouble(Console.ReadLine());
17
18          //Print a blank line
19          Console.WriteLine();
20
21          //Show the details you typed
22          Console.WriteLine("Name is {0}.", name);
23          Console.WriteLine("Age is {0}.", age);
24          Console.WriteLine("Height is {0}.", height);
25      }
26  }
```

نتیجه :

```
Enter your name: John
Enter your age: 18
Enter your height: 160.5

Name is John.
Age is 18.
Height is 160.5.
```

ابتدا **3** متغیر را برای ذخیره داده در برنامه تعریف می کنیم. **(خطوط 7 و 8 و 9)**برنامه از کاربر می خواهد که نام خود را وارد کند.**(خط 11)** در خط **12** شما به عنوان کاربر نام خود را وارد می کنید،مقدار متغیر نام، برابر مقداری است که توسط متد **ReadLine()** خوانده می شود،از آنجاییکه نام از نوع رشته است و مقداری که از متد **ReadLine()** خوانده می شود هم از نوع رشته است در نتیجه نیازی به تبدیل انواع نداریم.

سپس برنامه از ما سن را سوال می کند.(خط 13). سن ،متغیری از نوع صحیح (int) است، پس نیاز است که ما تبدیل از نوع رشته به صحیح را انجام دهیم،بنابراین از کلاس و متد Convert.ToInt32() برای این تبدیل استفاده می کنیم(خط 14).مقدار بازگشتی از این متد در متغیر سن قرار می گیرد،چون متغیر قد (height) را از نوع double تعریف کرده ایم برای تبدیل رشته دریافتی از محیط کنسول به نوع double باید از متد وکلاس Convert.ToDouble() استفاده کنیم.(خط 16)علاوه بر آنچه گفته شد شما می توانید از متد parse برای تبدیل های بالا استفاده کنید ، مانند:

```
age = int.Parse(Console.ReadLine());
height = double.Parse(Console.ReadLine());
```

توجه داشته باشد که این متد برای تبدیل رشته به رقم استفاده می شود یعنی رشته ای که توسط کاربر تایپ می شود باید فقط عدد باشد.

ساختارهای تصمیم

تقریبا همه زبانهای برنامه نویسی به شما اجازه اجرای کد را در شرایط مطمئن می دهند،حال تصور کنید که یک برنامه دارای ساختار تصمیم گیری نباشد و همه کدها را اجرا کند، این حالت شاید فقط برای چاپ یک پیغام در صفحه مناسب باشد ولی فرض کنید که شما بخواهید اگر مقدار یک متغیر با یک عدد برابر باشد سپس یک پیغام چاپ شود ان وقت با مشکل مواجه خواهید شد. سی شارپ راه های مختلفی برای رفع این نوع مشکلات ارائه می دهد.

در این بخش با مطالب زیر آشنا خواهید شد :

- **دستور if**
- **دستور if...else**
- **عملگر سه تایی**
- **دستور if چندگانه**
- **دستور if تو در تو**
- **عملگرهای منطقی**
- **دستور switch**

دستور if

می توان با استفاده از دستور **if** و یک شرط خاص که باعث ایجاد یک کد می شود یک منطق به برنامه خود اضافه کنید.دتور **if** ساده ترین دستور شرطی است که برنامه می گوید اگر شرطی برقرار است کد معینی را انجام بده

ساختار دستور **if** به صورت زیر است :

```
if(condition)
code to execute;
```

قبل از اجرای دستور **if** ابتدا شرط بررسی می شود،اگر شرط برقرار باشد یعنی درست باشد سپس کد اجرا می شود.شرط یک عبارت مقایسه ای است.
می توان از عملگرهای مقایسه ای برای تست درست یا اشتباه بودن شرط استفاده کرد.اجازه بدهید که نگاهی به نحوه استفاده از دستور **if** در داخل برنامه بیندازیم.

برنامه زیر پیغام **Hello World** را اگر مقدار **number** کمتر از **10** و **Goodbye World** را اگر مقدار **number** از **10** بزرگتر باشد در صفحه نمایش می دهد.

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           //Declare a variable and set it a value less than 10
8           int number = 5;
9
10          //If the value of number is less than 10
11          if (number < 10)
12              Console.WriteLine("Hello World.");
13
14          //Change the value of a number to a value which
15          // is greater than 10
16          number = 15;
17
18          //If the value of number is greater than 10
19          if (number > 10)
20              Console.WriteLine("Goodbye World.");
21      }
22  }
```

نتیجه :

```
Hello World.
Goodbye World.
```

در خط **8** یک متغیر با نام **number** تعریف و مقدار **5** به آن اختصاص داده شده است.
وقتی به اولین دستور **if** در خط **11** می رسیم برنامه تشخیص می دهد که مقدار **number** از **10** کمتر است یعنی **5** کوچکتر از **10** است.
منطقی است که نتیجه مقایسه درست می باشد بنابراین دستور **if** دستور را اجرا می کند**(خط 12)** و پیغام **Hello World** چاپ می شود.

حال مقدار **number** را به **15** تغییر می دهیم **(خط 16)**.وقتی به دومین دستور **if** در خط **19** می رسیم برنامه مقدار **number** را با **10** مقایسه می کند و چون مقدار **number** یعنی **15** از **10** بزرگتر است برنامه پیغام **Goodbye World** را چاپ می کند.**(خط 20)**
به این نکته توجه کنید که دستور **if** را می توان در یک خط نوشت :

```
if (number > 10) Console.WriteLine("Goodbye World.");
```

شما می توانید چندیندستور را در داخل دستور **if** بنویسید. کافیست که از یک آکولاد برای نشان دادن ابتدا و انتهای دستورات استفاده کنید.
همه دستورات داخل بین آکولاد جز بدنه دستور **if** هستند. نحوه تعریف چند دستور در داخل بدنه **if** به صورت زیر است :

```
if(condition)
{
       statement1;
       statement2;
           .
           .
           .
       statementN;
}
```

این هم یک مثال ساده :

```
if (x > 10)
{
Console.WriteLine("x is greater than 10.");
Console.WriteLine("This is still part of the if statement.");
}
```

در مثال بالا اگر مقدار **x** از **10** بزرگتر باشد دو پیغام چاپ می شود.

حال اگر به عنوان مثال آکولاد را حذف کنیم و مقدار **x** از **10** بزرگتر نباشد مانند کد زیر :

```
if (x > 10)
Console.WriteLine("x is greater than 10.");
Console.WriteLine("This is still part of the if statement. (Really?)");
```

کد بالا در صورتی بهتر خوانده می شود که بین دستورات فاصله بگذاریم.

```
if (x > 10)
Console.WriteLine("x is greater than 10.");

Console.WriteLine("This is still part of the if statement. (Really?)");
```

می بینید که دستور دوم**(خط 3)** در مثال بالا جز دتور **if** نیست.اینجاست که چون ما فرض را بر این گذاشته ایم که مقدار **x** از **10** کوچکتر است پس خط **This is still part of the if statement. (Really?)** چاپ می شود. در نتیجه اهمیت وجود آکولاد مشخص می شود.به عنوان تمرین همیشه حتی اگر فقط یک دستور در بدنه **if** داشتید برای آن یک آکولاد بگذارید.فراموش نکنید که از قلم انداختن یک آکولاد

باعث به وجود آمدن خطا شده و یافتن آن را سخت می کند. یکی از خطاهای معمول کسانی که برنامه نویسی را تازه شروع کرده اند قرار دادن سیمیکولن در سمت راست پرانتز **if** است.

به عنوان مثال :

```
if (x > 10); ◄——
Console.WriteLine("x is greater than 10");
```

به یاد داشته باشید که **if** یک مقایسه را انجام میدهد و دستور اجرایی نیست.بنابراین برنامه شما با یک خطای منطقی مواجه می شود.

همیشه به یاد داشته باشید که قرار گرفتن سیمیکولن در سمت راست پرانتز **if** به منزله این است که بلوک کد در اینجا به پایان رسیده است.

مثالی دیگر در مورد دستور **if** :

```
using System;

public class Program
{
    public static void Main()
    {
        int firstNumber;
        int secondNumber;

        Console.Write("Enter a number: ");
        firstNumber = Convert.ToInt32(Console.ReadLine());

        Console.Write("Enter another number: ");
        secondNumber = Convert.ToInt32(Console.ReadLine());

        if (firstNumber == secondNumber)
        {
            Console.WriteLine("{0} == {1}", firstNumber, secondNumber);
        }
        if (firstNumber != secondNumber)
        {
            Console.WriteLine("{0} != {1}", firstNumber, secondNumber);
        }
        if (firstNumber < secondNumber)
        {
            Console.WriteLine("{0} < {1}", firstNumber, secondNumber);
        }
        if (firstNumber > secondNumber)
        {
            Console.WriteLine("{0} > {1}", firstNumber, secondNumber);
        }
        if (firstNumber <= secondNumber)
        {
            Console.WriteLine("{0} <= {1}", firstNumber, secondNumber);
        }
        if (firstNumber >= secondNumber)
        {
            Console.WriteLine("{0} >= {1}", firstNumber, secondNumber);
        }
    }
}
```

نتیجه

```
Enter a number: 2
Enter another number: 5
2 != 5
2 < 5
2 <= 5
Enter a number: 10
Enter another number: 3
10 != 3
10 > 3
10 >= 3
Enter a number: 5
Enter another number: 5
5 == 5
5 <= 5
5 >= 5
```

ما از عملگرهای مقایسه ای در دستور **if** استفاده کرده ایم،ابتدا دو عدد که قرار است با هم مقایسه شوند را به عنوان ورودی از کاربر می گیریم،اعداد با هم مقایسه می شوند و اگر شرط درست بود پیغامی چاپ می شود،به این نکته توجه داشته باشید که شرطها مقادیر بولی هستند،بنابراین شما می توانید نتیجه یک عبارت را در داخل یکمتغیر بولی ذخیره کنید و سپس از متغیر به عنوان شرط در دستور **if** استفاده کنید.

```
bool isNewMillenium = year == 2000;

if (isNewMillenium)
{
Console.WriteLine("Happy New Millenium!");
}
```

اگر مقدار **year** برابر **2000**باشد سپس حاصل عبارت در متغیر **isNewMillenium** ذخیره می شود،می توان از متغیر برای تشخیص کد اجرایی بدنه دستور **if** استفاده کرد خواه مقدار متغیر درست باشد یا نادرست.

دستور if...else

دستور **if** فقط برای اجرای یک حالت خاص به کار می رود یعنی اگر حالتی برقرار بود کارخاصی انجام شود،اما زمانی که شما بخواهید اگر شرط خاصی برقرار شد یک دستور و اگر برقرار نبود دستور دیگر اجرا شود باید از دستور **if...else** استفاده کنید.

ساختار دستور **if...else** در زیر آمده است :

```
if(condition)
{
code to execute if condition is true;
}
else
{
code to execute if condition is false;
}
```

از کلمه کلیدی **else** نمی توان به تنهایی استفاده کرد بلکه حتما باید با **if** به کار برده شود،اگر فقط یک کد اجرایی در داخل بدنه **if** و بدنه **else** دارید استفاده از آکولاد اختیاری است .کد داخل بلوک **else** فقط در صورتی اجرا می شود که شرط داخل دستور **if** نادرست باشد.

در زیر نحوه استفاده از دستور **if...else** آمده است.

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           int number = 5;
8
9           //Test the condition
10          if (number < 10)
11          {
12              Console.WriteLine("The number is less than 10.");
13          }
14          else
15          {
16              Console.WriteLine("The number is either greater than or equal to 10.");
17          }
18
19          //Modify value of number
20          number = 15;
21
22          //Repeat the test to yield a different result
23          if (number < 10)
24          {
25              Console.WriteLine("The number is less than 10.");
26          }
27          else
28          {
29              Console.WriteLine("The number is either greater than or equal to 10.");
30          }
31      }
32  }
```

```
The number is less than 10.
The number is either greater than or equal to 10.
```

وقتی مقدار **number** از 10 کمتر باشد کد داخل بلوک **if** اجرا می شود و اگر مقدار **number** را تغییر دهیم و به مقداری بزرگتر از 10 تغییر دهیم شرط نادرست می شودو کد داخل بلوک **else** اجرا می شود. مانند بلوک **if** نباید به آخر کلمه کلیدی **else** سیمیکولن اضافه شود.

عملگر شرطی

عملگر شرطی (:?) در سی شارپ مانند دستور شرطی **if…else** عمل می کند.

در زیر نحوه استفاده از این عملگر آمده است:

```
<condition> ? <result if true> : <result if false>
```

عملگر شرطی تنها عملگر سه تایی سی شارپ است که نیاز به سه عملوند دارد، شرط، یک مقدار زمانی که شرط درست باشد و یک مقدار زمانی که شرط نادرست باشد.

اجازه بدهید که نحوه استفاده این عملگر را در داخل برنامه مورد بررسی قرار دهیم.

```
public class Program
{
    public static void Main()
    {
        string pet1 = "puppy";
        string pet2 = "kitten";
        string type1;
        string type2;

        type1 = (pet1 == "puppy") ? "dog" : "cat";
        type2 = (pet2 == "kitten") ? "cat" : "dog";
    }
}
```

برنامه بالا نحوه استفاده از این عملگر شرطی را نشان می دهد.

خط یک به صورت زیر ترجمه می شود : اگر مقدار **pet1** برابر با **puppy** سپس مقدار **dog** را در **type1** قرار بده در غیر این صورت مقدار **cat** را **type1** قرار بده.

خط دو به صورت زیر ترجمه می شود : اگر مقدار **pet2** برابر با **kitten** سپس مقدار **cat** را در **type2** قرار بده در غیر این صورت مقدار **.dog**

حال برنامه بالا را با استفاده از دستور **if else** می نویسیم:

```
if (pet1 == "puppy")
    type1 = "dog";
else
    type1 = "cat";
```

هنگامی که چندین دستور در داخل یک بلوک **if** یا **else** دارید از عملگر شرطی استفاده نکنید چون خوانایی برنامه را پایین می آورد.

دستور if چندگانه

اگر بخواهید چند شرط را بررسی کنید چکار می کنید؟

می توانید از چندین دستور **if** استفاده کنید و بهتر است که این دستورات **if** را به صورت زیر بنویسید :

```
if (condition)
{
   code to execute;
}
else
{
   if (condition)
   {
      code to execute;
   }
   else
   {
      if (condition)
      {
         code to execute;
      }
      else
      {
         code to execute;
      }
   }
}
```

خواندن کد بالا سخت است.

بهتر است دستورات را به صورت تو رفتگی در داخل بلوک **else** بنویسید.

میتوانید کد بالا را ساده تر کنید :

```
if(condition)
{
   code to execute;
}
else if(condition)
{
   code to execute;
}
else if(condition)
{
   code to execute;
}
else
{
   code to execute;
}
```

حال که نحوه استفده از دستور **else if** را یاد گرفتید باید بدانید که مانند **else** ، **else if** نیز به دتور **if** وابسته است.دستور **else if** وقتی اجرا می شود که اولیندستور **if** اشتباه باشد حال اگر **else if** اشتباه باشد دستور **else if** بعدی اجرا می شود. و اگر آن نیز اجرا نشود در نهایت دستور **else** اجرا می شود.

برنامه زیر نحوه استفاده از دستور**if else** را نشان می دهد :

```
using System;

public class Program
{
    public static void Main()
    {
        int choice;

        Console.WriteLine("What's your favorite color?");
        Console.WriteLine("[1] Black");
        Console.WriteLine("[2] White");
        Console.WriteLine("[3] Blue");
        Console.WriteLine("[4] Red");
        Console.WriteLine("[5] Yellow\n");

        Console.Write("Enter your choice: ");
        choice = Convert.ToInt32(Console.ReadLine());

        if (choice == 1)
        {
            Console.WriteLine("You might like my black t-shirt.");
        }
        else if (choice == 2)
        {
            Console.WriteLine("You might be a clean and tidy person.");
        }
        else if (choice == 3)
        {
            Console.WriteLine("You might be sad today.");
        }
        else if (choice == 4)
        {
            Console.WriteLine("You might be inlove right now.");
        }
        else if (choice == 5)
        {
            Console.WriteLine("Lemon might be your favorite fruit.");
        }
        else
        {
            Console.WriteLine("Sorry, your favorite color is " +
                              "not in the choices above.");
        }
    }
}
```

نتیجه :

```
What's your favorite color?
[1] Black
[2] White
[3] Blue
[4] Red
[5] Yellow

Enter your choice: 1
You might like my black t-shirt.
```

```
What's your favorite color?
[1] Black
[2] White
[3] Blue
[4] Red
[5] Yellow

Enter your choice: 999
Sorry, your favorite color is not in the choices above.
```

خروجی برنامه بالا به متغیر choice وابسته است. بسته به اینکه شما چه چیزی انتخاب می کنید پیغامهای مختلفی چاپ می شود.اگر عددی که شما تایپ می کنید در داخل حالتهای انتخاب نباشد کد مربوط به بلوک else اجرا می شود.

دستور if تو در تو

می توان از دستور **if** تو در تو در سی شارپ استفاده کرد. یک دستور ساده **if** در داخل دستور **if** دیگر.

```
if (condition)
{
    code to execute;

    if (condition)
    {
         code to execute;
    }
    else if (condition)
    {
        if (condition)
        {
           code to execute;
        }
    }
}
else
{
    if (condition)
    {
       code to execute;
    }
}
```

اجازه بدهید که نحوه استفاده از دستور **if** تو در تو را نشان دهیم :

```
using System;
public class Program
{
    public static void Main()
    {
        int age;
        string gender;

        Console.Write("Enter your age: ");
        age = Convert.ToInt32(Console.ReadLine());

        Console.Write("Enter your gender (male/female): ");

        gender = Console.ReadLine();

        if (age > 12)
        {
            if (age < 20)
            {
                if (gender == "male")
                {
                    Console.WriteLine("You are a teenage boy.");
                }
                else
                {
                    Console.WriteLine("You are a teenage girl.");
                }
            }
            else
            {
                Console.WriteLine("You are already an adult.");
            }
        }
        else
        {
            Console.WriteLine("You are still too young.");
        }
    }
}
```

نتیجه

```
Enter your age: 18
Enter your gender: male
You are a teenage boy.
```

```
Enter your age: 12
Enter your gender: female
You are still too young.
```

اجازه بدهید که برنامه را کالبد شکافی کنیم.ابتدا برنامه از شما درباره سن تان سوال می کند (خط 11).

در خط 14 درباره جنس تان از شما سوال می کند،سپس به اولین دستور if می رسد.(خط 16)

در این قسمت اگر سن شما بیشتر از 12 سال باشد برنامه وارد بدنه دستور if می شود در غیر اینصورت وارد بلوک else(خط 34)مربوط به همین دستور if می شود.

حال فرض کنیم که ن شما بیشتر از 12 سال است و شما وارد بدنه اولین ifشده اید،در بدنه اولین if دو دستور if دیگر را مشاهده می کنید،اگر سن کمتر 20 باشد شما وراد بدنه if دوم می شوید و اگر نباشد به قمت else متناظر با آن می روید.(خط 29)

دوباره فرض می کنیم که سن شما کمتر از 20 باشد،در اینصورت وارد بدنه if دوم شده و با یک if دیگر مواجه می شوید.(خط 20)

در اینجا جنسیت شما مورد بررسی قرار می گیرد که گر برابر "male" باشد کدهای اخل بدنه سومین if اجرا می شود در غیر اینصورت قسمت else مربوط به این if اجرا می شود(خط 24). پیشنهاد می شود که از if تو در تو در برنامه کمتر استفاده کنید چون خوانایی برنامه را پایین می آورد.

استفاده از عملگرهای منطقی

عملگرهای منطقی به شما اجازه می دهند که چندین شرط را با هم ترکیب کنید.

این عملگرها حداقل دو شرط را در گیر می کنند و در آخر یک مقدار بولی را بر می گردانند.

در جدول زیر برخی از عملگرهای منطقی آمده است :

عملگر	تلفظ	مثال	تاثیر
&&	And	z = (x > 2) && (y < 10)	مقدار Z در صورتی true است که هر دو شرط دو طرف عملگر مقدارشان true باشد. اگر فقط مقدار یکی از شروط false باشد مقدار z، false خواهد شد.
||	Or	z = (x > 2) || (y < 10)	مقدار Z در صورتی true است که یکی از دو شرط دو طرف عملگر مقدارشان true باشد. اگر هر دو شرط مقدارشانfalse باشد مقدار z، false خواهد شد.
!	Not	z = !(x > 2)	مقدار Z در صورتی true است که مقدار شرط false باشد و در صورتی false است که مقدار شرط true باشد.

به عنوان مثال جمله z = (x > 2) && (y < 10) را به این صورت بخوانید : "در صورتی مقدار z برابر true است که مقدار x بزرگتر از 2 و مقدار y کوچکتر از 10 باشد در غیر اینصورت false است". این جمله بدین معناست که برای اینکه مقدار کل دستور true باشد باید مقدار همه شروط true باشد.

عملگر منطقی OR (||) تاثیر متفاوتی نسبت به عملگر منطقی AND (&&) دارد. نتیجه عملگر منطقی OR برابر true است اگر فقط مقدار یکی از شروط true باشد. و اگر مقدار هیچ یک از شروط true نباشد نتیجه false خواهد شد.

می توان عملگرهای منطقی AND و OR را با هم ترکیب کرده و در یک عبارت به کار برد مانند :

```
if ( (x == 1) && ( (y > 3) || z < 10) ) )
{
//do something here
}
```

در اینجا استفاده از پرانتز مهم است چون از آن در گروه بندی شرطها استفاده می کنیم.

در اینجا ابتدا عبارت(z < 10) || (y > 3) مورد بررسی قرار می گیرد.(به علت تقدم عملگرها)سپس نتیجه آن بوسیله عملگر AND با نتیجه (x == 1) مقایسه می شود.

حال بیایید نحوه استفاده از عملگرهای منطقی در برنامه را مورد بررسی قرار دهیم :

```
using System;

public class Program
{
    public static void Main()
    {
        int age;
        string gender;

        Console.Write("Enter your age: ");
        age = Convert.ToInt32(Console.ReadLine());

        Console.Write("Enter your gender (male/female): ");
        gender = Console.ReadLine();

        if (age > 12 && age < 20)
        {
            if (gender == "male")
            {
                Console.WriteLine("You are a teenage boy.");
            }
            else
            {
                Console.WriteLine("You are a teenage girl.");
            }
        }
        else
        {
            Console.WriteLine("You are not a teenager.");
        }
    }
}
```

نتیجه :

```
Enter your age: 18
Enter your gender (male/female): female
You are a teenage girl.
```

```
Enter you age: 10
Enter your gender (male/female): male
You are not a teenager.
```

برنامه بالا نحوه استفاده از عملگر منطقی **AND** را نشان می دهد (خط 16).

وقتی به دستور **if** می رسید (خط 16) برنامه سن شما را چک می کند،اگر سن شما بزرگتر از 12 و کوچکتر از 20 باشد (سن تان بین 12 و20باشد) یعنی مقدار هر دو **true** باشد سپس کدهای داخل بلوک **if** اجرا می شوند،اگر نتیجه یکی از شروط **false** باشد کدهای داخل بلوک **else** اجرا می شود.

عملگر **AND** عملوند سمت چپ را مرود بررسی قرار می دهد،اگر مقدار آن **false** باشد دیگر عملوند سمت راست را بررسی نمی کند و مقدار **false** را بر می گرداند. بر عکس عملگر || عملوند سمت چپ را مورد بررسی قرار می دهد و اگر مقدار آن **true** باشد سپس عملوند سمت راست را نادیده می گیرد و مقدار **true** را بر می گرداند.

نکته مهم اینجاست که شما می توانید از عملگرهای & و | به عنوان عملگر بیتی استفاده کنید.

```
if (x == 2 & y == 3)
{
//Some code here
}

if (x == 2 | y == 3)
{
//Some code here
}
```

تفاوت جزئی این عملگرها وقتی که به عنوان عملگر بیتی به کار می روند این است که دو عملوند را بدون در نظر گرفتن مقدار عملوند سمت چپ مورد بررسی قرار می دهند.

به عنوان مثال حتی اگر مقدار عملوند سمت چپ **false** باشد عملوند سمت چپ به وسیله عملگر بیتی **AND (&)** ارزیابی می شود.

اگر شرطها را در برنامه ترکیب کنید استفاده از عملگرهای منطقی **AND (&&)** و **OR (||)** به جای عملگرهای بیتی **AND(&)** و **OR (|)**بهتر خواهد بود.

یکی دیگر از عملگرهای منطقی عملگر **NOT (!)** است که نتیجه یک عبارت را خنثی یا منفی می کند.به مثال زیر توجه کنید:

```
if (!(x == 2))
{
Console.WriteLine("x is not equal to 2.");
}
```

اگر نتیجه عبارت **x == 2** برابر **false** باشد عملگر **!** آن را **True** می کند.

دستور Switch

در سی شارپ ساختاری به نام **switch** وجود دارد که به شما اجازه می دهد که با توجه به مقدار ثابت یک متغیر چندین انتخاب داشته باشید. دستور **switch** معادل دستور **if** تو در تو است با این تفاوت که در دستور **switch** متغیر فقط مقادیر ثابتی از اعداد ، رشته ها و یا کاراکترها را قبول می کند. مقادیر ثابت مقادیری هستند که قابل تغیر نیستند.

در زیر نحوه استفاده از دستور **switch** آمده است :

```
switch (testVar)
{
   case compareVal1:
       code to execute if testVar == compareVa11;
       break;
   case compareVa12:
       code to execute if testVar == compareVa12;
       break;
   .
   .
   .
   case compareVa1N:
       code to execute if testVer == compareVa1N;
       break;
   default:
       code to execute if none of the values above match the testVar;
       break;
}
```

ابتدا یک مقدار در متغیر **switch** که در مثال بالا **testVar** است قرار می دهید.این مقدار با هر یک از عبارتهای **case** داخل بلوک **switch** مقایسه می شود. اگر مقدار متغیر با هر یک از مقادیر موجود در دستورات **case** برابر بود کد مربوط به آن **case** اجرا خواهد شد. به این نکته توجه کنید که حتی اگر تعداد خط کدهای داخل دستور **case** از یکی بیشتر باشد نباید از آکولاد استفاده کنیم. آخر هر دستور **case** با کلمه کلیدی **break** تشخیص داده می شود که باعث می شود برنامه از دستور **switch** خارج شده ودستورات بعد از آن اجرا شوند. اگر این کلمه کلیدی از قلم بیوفتد برنامه با خطا مواجه می شود.

دستور **switch** یک بخش **default** دارد. این دستور در صورتی اجرا می شود که مقدار متغیر با هیچ یک از مقادیر دستورات **case** برابر نباشد.

دستور **default** اختیاری است و اگر از بدنه **switch** حذف شود هیچ اتفاقی نمی افتد. مکان این دستور هم مهم نیست اما بر طبق تعریف آن را در پایان دستورات می نویسند.

به مثالی در مورد دستور **switch** توجه کنید :

```
using System;

public class Program
{
    public static void Main()
    {
        int choice;

        Console.WriteLine("What's your favorite pet?");
        Console.WriteLine("[1] Dog");
        Console.WriteLine("[2] Cat");
        Console.WriteLine("[3] Rabbit");
        Console.WriteLine("[4] Turtle");
        Console.WriteLine("[5] Fish");
        Console.WriteLine("[6] Not in the choices");
        Console.Write("\nEnter your choice: ");

        choice = Convert.ToInt32(Console.ReadLine());

        switch (choice)
        {
            case 1:
                Console.WriteLine("Your favorite pet is Dog.");
                break;
            case 2:
                Console.WriteLine("Your favorite pet is Cat.");
                break;
            case 3:
                Console.WriteLine("Your favorite pet is Rabbit.");
                break;
            case 4:
                Console.WriteLine("Your favorite pet is Turtle.");
                break;
            case 5:
                Console.WriteLine("Your favorite pet is Fish.");
                break;
            case 6:
                Console.WriteLine("Your favorite pet is not in the choices.");
                break;
            default:
                Console.WriteLine("You don't have a favorite pet.");
                break;
        }
    }
}
```

نتیجه

```
What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

Enter your choice: 2
Your favorite pet is Cat.




What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

Enter your choice: 99
You don't have a favorite pet.
```

برنامه بالا به شما اجازه انتخاب حیوان مورد علاقه تان را می دهد. به اسم هر حیوان یک عدد نسبت داده شده است. شما عدد را وارد می کنید و این عدد در دستور **switch** با مقادیر **case** مقایسه می شود و با هر کدام از آن مقادیر که برابر بود پیغام مناسب نمایش داده خواهد شد. اگر هم با هیچ کدام از مقادیر **case** ها برابر نبود دستور **default** اجرا می شود.

یکی دیگر از ویژگیهای دستور **switch** این است که شما می توانید از دو یا چند **case** برای نشان داده یک مجموعه کد استفاده کنید.

در مثال زیر اگر مقدار **number** ، **1**، **2** یا **3** باشد یک کد اجرا می شود. توجه کنید که **case** ها باید پشت سر هم نوشته شوند.

```
switch (number)
{
   case 1:
   case 2:
   case 3:
       Console.WriteLine("This code is shared by three values.");
       break;
}
```

همانطور که قبلا ذکر شد دستور **switch** معادل دستور **if** تو در تو است.

برنامه بالا را به صورت زیر نیز می توان نوشت :

```
if (choice == 1)
    Console.WriteLine("Your favorite pet is Dog.");
else if (choice == 2)
    Console.WriteLine("Your favorite pet is Cat.");
else if (choice == 3)
    Console.WriteLine("Your favorite pet is Rabbit.");
else if (choice == 4)
    Console.WriteLine("Your favorite pet is Turtle.");
else if (choice == 5)
    Console.WriteLine("Your favorite pet is Fish.");
else if (choice == 6)
    Console.WriteLine("Your favorite pet is not in the choices.");
else
    Console.WriteLine("You don't have a favorite pet.");
```

کد بالا دقیقا نتیجه ای مانند دستور **switch** دارد. دستور **default** معادل دستور **else** می باشد.

حال از بین این دو دستور **(switch** و **if else)** کدامیک را انتخاب کنیم.

از دستور **switch** موقعی استفاده می کنیم که مقداری که می خواهیم با دیگر مقادیر مقایسه شود ثابت باشد.

مثلا در مثال زیر هیچگاه از **switch** استفاده نکنید.

```
int myNumber = 5;
int x = 5;

switch (myNumber)
{
   case x:
       Console.WriteLine("Error, you can't use variables as a value" +
                                  " to be compared in a case statment.");
       break;
}
```

مشاهده می کنید که با اینکه مقدار **x** عدد **5** است و به طور واضح با متغیر **myNumber** مقایسه شده است برنامه خطا می دهد چون **x** یک ثابت نیست بلکه یک متغیر است یا به زبان ساده تر ،قابلیت تغییر را دارد. اگر بخواهید از **x** استفاده کنید و برنامه خطا ندهد باید از کلمه کلیدی **const** به صورت زیر استفاده کنید.

```
int myNumber = 5;
const int x = 5;

switch (myNumber)
{
  case x:
      Console.WriteLine("Error has been fixed!");
  break;
}
```

از کلمه کلیدی **const** برای ایجاد ثابتها استفاده می شود.توجه کنید که بعد از تعریف یک ثابت نمی توان مقدار آن را در طول برنامه تغییر داد. به یاد داشته باشید که باید ثابتها را حتما مقداردهی کنید. دستور **switch** یک مقدار را با مقادیر **Case** ها مقایسه می کند و شما لازم نیست که به شکل زیر مقادیر را با هم مقایسه کنید :

```
switch (myNumber)
{
  case x > myNumber:
      Console.WriteLine("switch staments can't test if a value is less than " +
                        "or greater than the other value.");
  break;
}
```

تکرار

ساختارهای تکرار به شما اجازه می دهند که یک یا چند دستور کد را تا زمانی که یک شرط برقرار است تکرار کنید،بدون ساختارهای تکرار شما مجبورید همان تعداد کدها را بنویسید که بسیار خسته کننده است،مثلا شما مجبورید 10 بار جمله **"Hello World."** را تایپ کنید مانند مثال زیر :

```
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
```

البته شما می توانید با کپی کردن این تعداد کد را راحت بنویسید ولی این کار در کل کیفیت کدنویسی را پایین می اورد،را بهتر برای نوشتن کدهای بالا استفاده از حلقه ها است.

ساختارهای تکرار در سی شارپ عبارتند از :

- **while**
- **do while**
- **for**

حلقه While

ابتدایی ترین ساختار تکرار در سی شارپ حلقه **While** است.ابتدا یک شرط را مورد بررسی قرار می دهد و تا زمانیکه شرط برقرار باشد کدهای درون بلوک اجرا می شوند.

ساختار حلقه **while** به صورت زیر است :

```
while(condition)
{
code to loop;
}
```

می بینید که ساختار **while** مانند ساختار **if** بسیار ساده است.

ابتدا یک شرط را که نتیجه آن یک مقدار بولی است مینویسیم اگر نتیجه درست یا **true** باشد سپس کدهای داخل بلوک **while** اجرا می شوند.
اگر شرط غلط یا **false** باشد وقتی که برنامه به حلقه **while** برسد هیچکدام از کدها را اجرا نمی کند. برای متوقف شدن حلقه باید مقادیر داخل حلقه **while** اصلاح شوند.
به یک متغیر شمارنده در داخل بدنه حلقه نیاز داریم.این شمارنده برای آزمایش شرط مورد استفاده قرار می گیرد و ادامه یا توقف حلقه به نوعی به آن وابسته است.این شمارنده را در داخل بدنه باید کاهش یا افزایش دهیم.

در برنامه زیر نحوه استفاده از حلقه **while** آمده است :

```
1  using System;
2
3  public class Program
4  {
5      public static void Main()
6      {
7          int counter = 1;
8
9          while (counter <= 10)
10         {
11             Console.WriteLine("Hello World!");
12             counter++;
13         }
14
15     }
16 }
```

نتیجه

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

برنامه بالا 10 بار پیغام Hello World! را چاپ می کند.

اگر از حلقه در مثال بالا استفاده نمی کردیم مجبور بودیم تمام 10 خط را تایپ کنیم.

اجازه دهید که نگاهی به کدهای برنامه فوق بیندازیم.

ابتدا در خط 7 یک متغیر تعریف و از آن به عنوان شمارنده حلقه استفاده شده است،سپس به آن مقدار 1 را اختصاص می دهیم چون اگر مقدار نداشته باشد نمی توان در شرط از آن استفاده کرد.

در خط 9 حلقه While را وارد می کنیم. در حلقه while ابتدا مقدار اولیه شمارنده با 10 مقایسه می شود که آیا از 10 کمتر است یا با آن برابر است. نتیجه هر بار مقایسه ورود به بدنه حلقه while و چاپ پیغام است.

همانطور که مشاهده می کنید بعد از هر بار مقایسه مقدار شمارنده یک واحد اضافه می شود (خط 12). حلقه تا زمانی تکرار می شود که مقدار شمارنده از 10 کمتر باشد.

اگر مقدار شمارنده یک بماند و آن را افزایش ندهیم و یا مقدار شرط هرگز false نشود یک حلقه بینهایت به وجود می آید.

به این نکته توجه کنید که در شرط بالا به جای علامت > از => استفاده شده است. اگر از علامت > استفاده می کردیم کد ما 9 بار تکرار می شد چون مقدار اولیه 1 است و هنگامی که شرط به 10 برسد false می شود چون 10 > 10 نیست.

اگر می خواهید یک حلقه بی نهایت ایجاد کنید که هیچگاه متوقف نشود باید یک شرط ایجاد کنید که همواره درست (true) باشد.

```
while(true)
{
//code to loop
}
```

این تکنیک در برخی موارد کارایی دارد و آن زمانی است که شما بخواهید با استفاده از دستورات break و return که در آینده توضیح خواهیم داد از حلقه خارج شوید.

حلقه do while

حلقه **do while** یکی دیگر از ساختارهای تکرار است. این حلقه بسیار شبیه حلقه **while** است با این تفاوت که در این حلقه ابتدا کد اجرا می شودو سپس شرط مورد بررسی قرار می گیرد.

ساختار حلقه **do while** به صورت زیر است :

```
do
{
code to repeat;

} while(condition);
```

همانطور که مشاهده می کنید شرط در آخر ساختار قرار دارد. این بدین معنی است که کدهای داخل بدنه حداقل یکبار اجرا می شوند.برخلاف حلقه **while** که اگر شرط نادرست باشد دستورات داخل بدنه اجرا نمی شوند.یکی از موراد برتری استفاده از حلقه **do while** نسبت به حلقه **while** زمانی است که شما بخواهید اطلاعاتی از کاربر دریافت کنید. به مثال زیر توجه کنید :

استفاده از **while**

```
//while version
Console.WriteLine("Enter a number greater than 10: ");
number = Convert.ToInt32(Console.ReadLine());

while (number < 10)
{
   Console.WriteLine("Enter a number greater than 10: ");
   number = Convert.ToInt32(Console.ReadLine());
}
```

استفاده از **do while**

```
//do while version

do
{
   Console.WriteLine("Enter a number greater than 10: ");
   number = Convert.ToInt32(Console.ReadLine());
} while (number < 10);
```

مشاهده می کنید که از کدهای کمتری در بدنه **do while** نسبت به **while** استفاده شده است.

حلقه for

یکی دیگر از ساختارهای تکرار حلقه **for** است،این حلقه عملی شبیه به حلقه **while** انجام می دهد و فقط دارای چند خصوصیت اضافی است.

ساختار حلقه **for** به صورت زیر است :

```
for(initialization; condition; operation)
{
   code to repeat;
}
```

مقدار دهی اولیه (initialization) اولین مقداری است که به شمارنده حلقه می دهیم،شمارنده فقط در داخل حلقه **for** قابل دسترسی است.

شرط (condition) در اینجا مقدار شمارنده را با یک مقدار دیگر مقایسه می کند و تعیین می کند که حلقه ادامه یابد یا نه.

عملگر (operation) که مقدار اولیه متغیر را کاهش یا افزایش می دهد.

در زیر یک مثال از حلقه **for** آمده است:

```
using System;

namespace ForLoopDemo
{
    public class Program
    {
        public static void Main()
        {
            for (int i = 1; i <= 10; i++)
            {
                Console.WriteLine("Number " + i);
            }
        }
    }
}
```

نتیجه

```
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
Number 7
Number 8
Number 9
Number 10
```

برنامه بالا اعداد 1 تا 10 را با استفاده از حلقه **for** می شمارد.

ابتدا یک متغیر به عنوان شمارنده تعریف می کنیم و آن را با مقدار 1 مقدار دهی اولیه می کنیم.

سپس با استفاده از شرط آن را با مقدار 10 مقایسه می کنیم که آیا کمتر است یا مساوی؟

توجه کنید که قسمت سوم حلقه (i++) فورا اجرا نمی شود.
کد اجرا می شود و ابتدا رشته Number و سپس مقدار جاری i یعنی 1 را چاپ می کند. آنگاه یک واحد به مقدار i اضافه شده و مقدار i برابر 2 می شود و بار دیگر i با عدد 10 مقایسه می شود و این حلقه تا زمانی که مقدار شرط true شود ادامه می یابد.
حال اگر بخواهید معکوس برنامه بالا را پیاده سازی کنید یعنی اعداد از بزرگ به کوچک چاپ شوند باید به صورت یزر عمل کنید :

```
for (int i = 10; i > 0; i--)
{
//code omitted
}
```

کد بالا اعداد را از 10 به 1 چاپ می کند (از بزرگ به کوچک).
مقدار اولیه شمارنده را 10 می دهیم و با استفاده از عملگر کاهش (--) برنامه ای که شمارش معکوس را انجام می دهد ایجاد می کنیم.
می توان قسمت شرط و عملگر را به صورت های دیگر نیز تغییر داد.
به عنوان مثال می توان از عملگرهای منطقی در قسمت شرط و از عملگرهای تخصیصی در قسمت عملگر افزایش یا کاهش استفاده کرد.
همچنین می توانید از چندین متغیر در ساختار حلقه for استفاده کنید.

```
for (int i = 1, y = 2; i < 10 && y > 20; i++, y -= 2)
{
//some code here
}
```

به این نکته توجه کنید که اگر از چندین متغیر شمارنده یا عملگر در حلقه for استفاده می کنید باید آنها را با استفاده از کاما از هم جدا کنید.

خارج شدن از حلقه با استفاده از break و continue

گاهی اوقات با وجود درست بودن شرط می خواهیم حلقه متوقف شود. سوال اینجاست که چطور این کار را انجام دهید؟ با استفاده از کلمه کلیدی **break** حلقه را متوقف کرده و با استفاده از کلمه کلیدی **continue** می توان بخشی از حلقه را رد کرد و به مرحله بعد رفت.

برنامه زیر نحوه استفاده از **continue** و **break** را نشان می دهد :

```
1   using System;
2
3   namespace BreakContinueDemo
4   {
5       public class Program
6       {
7           public static void Main()
8           {
9               Console.WriteLine("Demonstrating the use of break.\n");
10
11              for (int x = 1; x < 10; x++)
12              {
13                  if (x == 5)
14                      break;
15
16                  Console.WriteLine("Number " + x);
17              }
18
19              Console.WriteLine("\nDemonstrating the use of continue.\n");
20
21              for (int x = 1; x < 10; x++)
22              {
23                  if (x == 5)
24                      continue;
25
26                  Console.WriteLine("Number " + x);
27              }
28          }
29      }
30  }
```

```
Demonstrating the use of break.

Number 1
Number 2
Number 3
Number 4

Demonstrating the use of continue.

Number 1
Number 2
Number 3
Number 4
Number 6
Number 7
Number 8
Number 9
```

در این برنامه از حلقه **for** برای نشان دادن کاربرد دو کلمه کلیدی فوق استفاده شده است اگر به جای **for** از حلقه های **while** و **do…while** استفاده می شد نتیجه یکسانی به دست می امد.

همانطور که در شرط برنامه (خط 11) آمده است وقتی که مقدار x به عدد 5 رسید سپس دستور break اجرا شود(خط 12).
حلقه بلافاصله متوقف می شود حتی اگر شرط 10 > x برقرار باشد.
از طرف دیگر در خط 22 حلقه for فقط برای یک تکرار خاص متوقف شده و سپس ادامه می یابد.(وقتی مقدار x برابر 5 شود حلقه از 5 رد شده و مقدار 5 را چاپ نمی کند و بقیه مقادیر چاپ می شوند.)

آرایه ها

آرایه نوعی متغیر است که لیستی از آدرسهای مجموعه ای از داده های هم نوع را در خود ذخیره می کند،تعریف چندین متغیر از یک نوع برای هدفی یکسان بسیار خسته کننده است،مثلا اگر بخواهید صد متغیر از نوع اعداد صحیح تعریف کرده و از آنها استفاده کنید، مطمئنا تعریف این همه متغیر بسیار کسالت آور و خسته کننده است،اما با استفاده از آرایه می توان همه آنها را در یک خط تعریف کرد.

در زیر راهی ساده برای تعریف یک آرایه نشان داده شده است :

```
datatype[] arrayName = new datatype[length];
```

Datatype نوع داده هایی را نشان می دهد که آرایه در خود ذخیره می کند،کروشه که بعد از نوع داده قرار می گیرد و نشان دهنده استفاده از آرایه است.**arrayName** که نام آرایه را نشان می دهد،هنگام نامگذاری آرایه بهتر است که نام آرایه نشان دهنده نوع آرایه باشد،به عنوان مثال برای نامگذاری آرایه ای که اعداد را در خود ذخیره می کند از کلمه **number** استفاده کنید،طول آرایه که به کامپایلر می گوید شما قصد دارید چه تعداد داده یا مقدار را در آرایه ذخیره کنید،از کلمه کلیدی **new** هم برای اختصاص فضای حافظه به اندازه طول آرایه استفاده می شود،برای تعریف یک آرایه که **5** مقدار از نوع اعداد صحیح در خود ذخیره می کند باید به صورت زیر عمل کنیم :

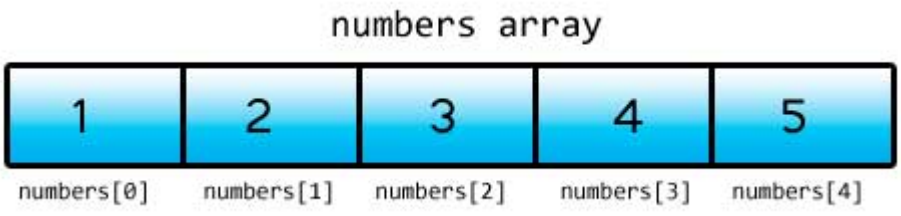
```
int[] numbers = new int[5];
```

در این مثال **5** آدرس از فضای حافظه کامپیوتر شما برای ذخیره **5** مقدار رزرو می شود،حال چطور مقادیرمان را در هر یک از این آدرسها ذخیره کنیم؟ برای دسترسی و اصلاح مقادیر آرایه از اندیس یا مکان آنها استفاده می شود.

```
numbers[0] = 1;
numbers[1] = 2;
numbers[2] = 3;
numbers[3] = 4;
numbers[4] = 5;
```

اندیس یک آرایه از صفر شروع شده و به یک واحد کمتر از طول آرایه ختم می شود،به عنوان مثال شما یک آرایه **5** عضوی دارید،اندیس آرایه از **0** تا **4** می باشد چون طول آرایه **5** است پس **5-1** برابر است با **4**،این بدان معناست که اندیس **0** نشان دهنده اولین عضو آرایه است و اندیس **1** نشان دهنده دومین عضو و الی آخر.

برای درک بهتر مثال بالا به شکل زیر توجه کنید :



به هر یک از اجزاء آرایه و اندیسهای داخل کروشه توجه کنید،کسانی که تازه شروع به برنامه نویسی کرده اند معمولا در گذاشتن اندیس دچار اشتباه می شوند و مثلا ممکن است در مثال بالا اندیسها را از **1** شروع کنند.

اگر بخواهید به یکی از اجزائ آرایه با استفاده از اندیسی دسترسی پیدا کنید که در محدوده اندیسهای آرایه شما نباشد با پیغام خطای **IndexOutOfRangeException** مواجه می شوید و بدین معنی است که شما آدرسی را می خواهید که وجود ندارد.

یکی دیگر از راه های تعریف سریع و مقدار دهی یک آرایه به صورت زیر است :

```
datatype[] arrayName =new datatype[length] { val1, val2, ... valN };
```

در این روش شما می توانید فورا بعد از تعریف اندازه آرایه مقادیر را در داخل آکولاد قرار دهید. به یاد داشته باشید که هر کدام از مقادیر را با استفاده از کاما از هم جدا کنید. همچنین تعداد مقادیر داخل آکولاد باید با اندازه آرایه تعریف شده برابر باشد.

به مثال زیر توجه کنید :

```
int[] numbers = new int[5] { 1, 2, 3, 4, 5 };
```

این مثال با مثال قبل هیچ تفاوتی ندارد و تعداد خطهای کدنویسی را کاهش می دهد.شما می توانید با استفاده از اندیس به مقدار هر یک از اجزاء آرایه دسترسی یابید و آنها را به دلخواه تغییر دهید.تعداد اجزاء آرایه در مثال بالا **5** است و ما **5** مقدار را در آن قرار می دهیم. اگر تعداد مقادیری که در آرایه قرار می دهیم کمتر یا بیشتر از طول آرایه باشد با خطا مواجه می شویم.

یکی دیگر از راه های تعریف آرایه در زیر امده است. شما می توانید هر تعداد عنصر را که خواستید در آرایه قرار دهید بدون اینکه اندازه آرایه را مشخص کنید.

به عنوان مثال :

```
int[] numbers = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

در این مثال ما **10** مقدار را به آرایه اختصاص داده ایم.نکته اینجاست که طول آرایه را تعریف نکرده ایم. در این حالت کامپایلر بعد از شمردن تعداد مقادیر داخل آکولاد طول آرایه را تشخیص می دهد.

به یاد داشته باشید که اگر برای آرایه طولی در نظر نگیرید باید برای آن مقدار تعریف کنید در عیر این صورت با خطا مواجه می شوید :

```
int[] numbers = new int[]; //not allowed
```

یک راه بسیار ساده تر برای تعریف آرایه به صورت زیر است :

```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

به سادگی و بدون احتیاج به کلمه کلیدی **new** می توان مقادیر را در داخل آکولاد قرار داد. کامپایلر به صورت اتوماتیک با شمارش مقادیر طول آرایه را تشخیص می دهد.

دستیابی به مقادیر آرایه با استفاده از حلقه for

در زیر مثالی در مورد استفاده از آرایه ها آمده است. در این برنامه 5 مقدار از کاربر گرفته شده و میانگین آنها حساب می شود:

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           int[] numbers = new int[5];
8           int total = 0;
9           double average;
10
11          for (int i = 0; i < numbers.Length; i++)
12          {
13              Console.Write("Enter a number: ");
14              numbers[i] = Convert.ToInt32(Console.ReadLine());
15          }
16
17          for (int i = 0; i < numbers.Length; i++)
18          {
19              total += numbers[i];
20          }
21
22          average = total / (double)numbers.Length;
23
24          Console.WriteLine("Average = {0}", average);
25      }
26  }
```

نتیجه

```
Enter a number: 90
Enter a number: 85
Enter a number: 80
Enter a number: 87
Enter a number: 92
Average = 86
```

در خط 7 یک آرایه تعریف شده است که می تواند 5 عدد صحیح را در خود ذخیره کند.خطوط 8 و 9 متغیرهایی تعریف شده اند که از آنها برای محاسبه میانگین استفاده می شود.توجه کنید که مقدار اولیه total صفر است تا از بروز خطا هنگام اضافه شدن مقدار به آن جلوگیری شود.در خطوط 11 تا 15 حلقه for برای تکرار و گرفتن ورودی از کاربر تعریف شده است.از خاصیت طول (Length) ارایه برای تشخیص تعداد اجزای آرایه استفاده می شود.اگر چه می توانستیم به سادگی در حلقه for مقدار 5 را برای شرط قرار دهیم ولی استفاده از خاصیت طول آرایه کار راحت تری است و می توانیم طول آرایه را تغییر دهیم و شرط حلقه for با تغییر جدید هماهنگ می شود.در خط 14 ورودی دریافت شده از کاربر به نوع int تبدیل و در آرایه ذخیره می شود.اندیس استفاده شده در number (خط 14) مقدار i جاری در حلقه است.برای مثال در ابتدای حلقه مقدار i صفر است بنابراین وقتی در خط 14 اولین داده از کاربر گرفته می شود اندیس آن برابر صفر می شود. در تکرار بعدی i یک واحد اضافه می شود و در نتیجه در خط 14 و بعد از ورود دومین داده توسط کاربر اندیس آن برابر یک می شود. این حالت تا زمانی که شرط در حلقه for برقرار است ادامه می یابد. در خطوط 17-20 از حلقه for دیگر برای دسترسی به مقدار هر یک از داده های آرایه استفاده شده است.در این حلقه نیز مانند حلقه قبل از مقدار متغیر شمارنده به عنوان اندیس استفاده می کنیم.

هر یک از اجزای عددی آرایه به متغیر **total** اضافه می شوند،بعد از پایان حلقه می توانیم میانگین اعداد را حساب کنیم **(خط 22)**. مقدار **total** را بر تعداد اجزای آرایه **(تعداد عدد ها)** تقسیم می کنیم،برای دسترسی به تعداد اجزای آرایه می توان از خاصیت **length** آرایه استفاده کرد،توجه کنید که در اینجا ما مقدار خاصیت **length** را به نوع **double** تبدیل کرده ایم بنابراین نتیجه عبارت یک مقدار از نوع **double** خواهد شد و دارای بخش کسری می باشد، حال اگر عملوند های تقسیم را به نوع **double** تبدیل نکنیم نتیجه تقسیم یک عدد از نوع صحیح خواهد شد و دارای بخش کسری نیست، خط **24** مقدار میانگین را در صفحه نمایش چاپ می کند، طول آرایه بعد از مقدار دهی نمی تواند تغیر کند، به عنوان مثال اگر یک آرایه را که شامل **5** جز است مقدار دهی کنید دیگر نمی توانید آن را مثلا به **10** جز تغییر اندازه دهید،البته تعداد خاصی از کلاسها مانند آرایه ها عمل می کنند و توانایی تغییر تعداد اجزای تشکیل دهنده خود را دارند،آرایه ها در برخی شرایط بسیار پر کاربرد هستند و تسلط شما بر این مفهوم و اینکه چطور از آنها استفاده کنید بسیار مهم است.

حلقه foreach

حلقه foreachیکی دیگر از ساختارهای تکرار در سی شارپ می باشد که مخصوصا برای آرایه ها ،لیستها و مجموعه ها طراحی شده است.حلقه **foreach** با هر بار گردش در بین اجزاء ، مقادیر هر یک از آنها را در داخل یک متغیر موقتی قرار می دهد و شما می توانید بواسطه این متغیر به مقادیر دسترسی پیدا کنید.

در زیر نحوه استفاده از حلقه **foreach** آمده است :

```
foreach (datatype temporaryVar in array)
{
   code to execute;
}
```

temporaryVar متغیری است که مقادیر اجزای آرایه را در خود نگهداری می کند.**temporaryVar** باید دارای نوع باشد تا بتواند مقادیر آرایه را در خود ذخیره کند.به عنوان مثال آرایه شما دارای اعدادی از نوع صحیح باشد باید نوع متغیر موقتی از نوع اعداد صحیح باشد یا هر نوع دیگری که بتواند اعداد صحیح را در خود ذخیره کند مانند **double** یا **long**.سپس کلمه کلیدی **in** و بعد از آن نام آرایه را می نویسیم.

در زیر نحوه استفاده از حلقه **foreach** امده است :

```
using System;

public class Program
{
    public static void Main()
    {
        int[] numbers = { 1, 2, 3, 4, 5 };

        foreach (int n in numbers)
        {
            Console.WriteLine("Number {0}", n);
        }
    }
}
```

نتیجه

```
Number 1
Number 2
Number 3
Number 4
Number 5
```

در برنامه آرایه ای با **5** جزء تعریف شده و مقادیر **1** تا **5** در آنها قرار داده شده است **(خط 7)**، در خط **9** حلقه **foreach** شروع می شود.ما یک متغیر موقتی تعریف کرده ایم که اعداد آرایه را در خود ذخیره می کند. در هر بار تکرار از حلقه **foreach** متغیر موقتی **n** ، مقادیر عددی را از آرایه استخراج می کند. حلقه **foreach** مقادیر اولین تا آخرین جزء آرایه را در اختیار ما قرار می دهد.

حلقه **foreach** برای دریافت هر یک از مقادیر آرایه کاربرد دارد،بعد از گرفتن مقدار یکی از اجزای آرایه ،مقدار متغیر موقتی را چاپ می کنیم.حلقه **foreach** یک ضعف دارد و آن اینست که این حلقه ما را قادر می سازد که به داده ها دسترسی یابیم و یا آنها را بخوانیم ولی اجازه اصلاح اجزاء آرایه را نمی دهد.

برای درک این مطلب در مثال زیر سعی شده است که مقدار هر یک از اجزا آرایه افزایش یابد :

```
int[] numbers = { 1, 2, 3 };

foreach(int number in numbers)
{
   number++;
}
```

اگر برنامه را اجرا کنید با خطا مواجه می شوید .برای اصلاح هر یک از اجزا آرایه می توان از حلقه **for** استفاده کرد.

```
int[] numbers = { 1, 2, 3 };

for (int i = 0; i < number.Length; i++)
{
   numbers[i]++;
}
```

آرایه های چند بعدی

آرایه های چند بعدی آرایه هایی هستند که برای دسترسی به هر یک از عناصر آنها باید از چندین اندیس استفاده کنیم.یک آرایه چند بعدی را می توان مانند یک جدول با تعدای ستون و ردیف تصور کنید.با افزایش اندیسها اندازه ابعاد آرایه نیز افزایش می یابد و آرایه های چند بعدی با بیش از دو اندیس به وجود می آیند.

نحوه ایجاد یک آرایه با دو بعد به صورت زیر است :

```
datatype[,] arrayName = new datatype[lengthX, lengthY];
```

و یک آرایه سه بعدی به صورت زیر ایجاد می شود :

```
datatype[, ,] arrayName = new datatype[lengthX, lengthY, lengthZ];
```

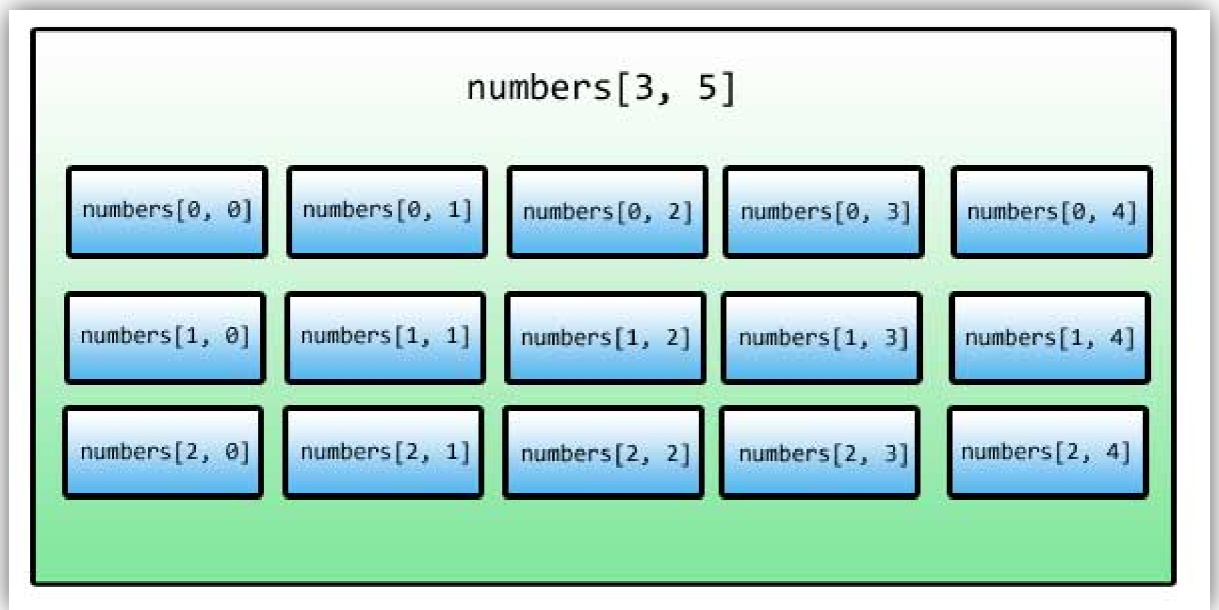
می توان یک آرایه با تعداد زیادی بعد ایجاد کرد به شرطی که هر بعد دارای طول مشخصی باشد.

به دلیل اینکه آرایه های سه بعدی یا آرایه های با بیشتر از دو بعد بسیار کمتر مورد استفاده قرار می گیرند اجازه بدهید که در این درس بر روی آرایه های دو بعدی تمرکز کنیم.در تعریف این نوع آرایه ابتدا نوع آرایه یعنی اینکه آرایه چه نوعی از انواع داده را در خود ذخیره می کند را مشخص می کنیم.سپس یک جفت کروشه و در داخل کروشه ها یک کاما قرار می دهیم. به تعداد کاماهایی که در داخل کروشه می گذارید توجه کنید.اگر آرایه ما دو بعدی است باید 1 کاما و اگر سه بعدی است باید 2 کاما قرار دهیم.سپس یک نام برای آرایه انتخاب کرده و بعد تعریف آنرا با گذاشتن کلمه new ، نوع داده و طول آن کامل می کنیم.در یک آرایه دو بعدی برای دسترسی به هر یک از عناصر به دو مقدار نیاز داریم یکی مقدار X و دیگری مقدار Y که مقدار x نشان دهنده ردیف و مقدار Y نشان دهنده ستون آرایه است البته اگر ما آرایه دو بعدی را به صورت جدول در نظر بگیریم.یک آرایه سه بعدی را می توان به صورت یک مکعب تصور کرد که دارای سه بعد است و x طول، Y عرض و z ارتفاع آن است.

یک مثال از آرایه دو بعدی در زیر آمده است :

```
int[,] numbers = new int[3, 5];
```

کد بالا به کامپایلر می گوید که فضای کافی به عناصر آرایه اختصاص بده (در این مثال 15 خانه).در شکل زیر مکان هر عنصر در یک آرایه دو بعدی نشان داده شده است.



مقدار **3** را به **X** اختصاص می دهیم چون **3** سطر و مقدار **5** را به **Y** چون **5** ستون داریم اختصاص می دهیم.

چطور یک آرایه چند بعدی را مقدار دهی کنیم؟ چند راه برای مقدار دهی به آرایه ها وجود دارد.

```
datatype[,] arrayName = new datatype[x, y] { { r0c0, r0c1, ... r0cX },
                                             { r1c0, r1c1, ... r1cX },
                                                       .
                                                       .
                                                       .
                                             { rYc0, rYc1, ... rYcX } };
```

برای راحتی کار می توان از نوشتن قسمت **new dataype[,]** صرف نظر کرد.

```
datatype[,] arrayName = { { r0c0, r0c1, ... r0cX },
                          { r1c0, r1c1, ... r1cX },
                                    .
                                    .
                                    .
                          { rYc0, rYc1, ... rYcX } };
```

به عنوان مثال :

```
int[,] numbers = { { 1, 2, 3, 4, 5 },
                   { 6, 7, 8, 9, 10 },
                   { 11, 12, 13, 14, 15 } };
```

و یا می توان مقدار دهی به عناصر را به صورت دستی انجام داد مانند :

```
array[0, 0] = value;
array[0, 1] = value;
array[0, 2] = value;
array[1, 0] = value;
array[1, 1] = value;
array[1, 2] = value;
```

```
array[2, 0] = value;
array[2, 1] = value;
array[2, 2] = value;
```

همانطور که مشاهده می کنید برای دسترسی به هر یک از عناصر در یک آرایه دو بعدی به سادگی می توان از اندیسهای X و Y و یک جفت کروشه مانند مثال استفاده کرد.

گردش در میان عناصر آرایه های چند بعدی

گردش در میان عناصر آرایه های چند بعدی نیاز به کمی دقت دارد. یکی از راههای آسان استفاده از حلقه **foreach** و یا حلقه **for** تو در تو است.
اجازه دهید ابتدا از حلقه **foreach** استفاده کنیم.

```
using System;

public class Program
{
    public static void Main()
    {
        int[,] numbers = { { 1, 2, 3, 4, 5 },
                           { 6, 7, 8, 9, 10 },
                           { 11, 12, 13, 14, 15 }
                         };

        foreach (int number in numbers)
        {
            Console.Write(number + " ");
        }
    }
}
```

نتیجه

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

مشاهده کردید که گردش در میان مقادیر عناصر یک آرایه چند بعدی چقدر راحت است. به وسیله حلقه **foreach** نمی توانیم انتهای ردیفها را مشخص کنیم. برنامه زیر نشان می دهد که چطور از حلقه **for** برای خواندن همه مقادیر آرایه و تعیین انتهای ردیف ها استفاده کنید.

```
using System;

public class Program
{
    public static void Main()
    {
        int[,] numbers = { { 1, 2, 3, 4, 5 },
                           { 6, 7, 8, 9, 10 },
                           { 11, 12, 13, 14, 15 }
                         };

        for (int row = 0; row < numbers.GetLength(0); row++)
        {
            for (int col = 0; col < numbers.GetLength(1); col++)
            {
                Console.Write(numbers[row, col] + " ");
            }

            //Go to the next line
            Console.WriteLine();
        }
    }
}
```

نتیجه

```
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
```

همانطور که در مثال بالا نشان داده شده است با استفاده از یک حلقه ساده **for** نمی توان به مقادیر دسترسی یافت بلکه به یک حلقه **for** تو در تونیاز داریم. در اولین حلقه **for** (خط **12**) یک متغیر تعریف شده است که در میان ردیف های آرایه **(row)** گردش می کند. این حلقه تا زمانی ادامه می یابد که مقدار ردیف کمتر از طول اولین بعد باشد.

در این مثال از متد **()GetLength** کلاس **Array** استفاده کرده ایم.این متد طول آرایه را در یک بعد خاص نشان می دهد و دارای یک پارامتر است که همان بعد آرایه می باشد.

به عنوان مثال برای به دست آوردن طول اولین بعد آرایه مقدار صفر را به این متد ارسال می کنیم چون شمارش ابعاد یک آرایه از صفر تا یک واحد کمتر از تعداد ابعاد انجام می شود.

در داخل اولین حلقه **for** حلقه **for** دیگری تعریف شده است (خط **14**).

در این حلقه یک شمارنده برای شمارش تعداد ستونهای **(columns)** هر ردیف تعریف شده است و در شرط داخل آن بار دیگر از متد **()GetLength** استفاده شده است، ولی این بار مقدار 1 را به آن ارسال می کنیم تا طول بعد دوم آرایه را به دست آوریم.

پس به عنوان مثال وقتی که مقدار ردیف **(row)** صفر باشد ، حلقه دوم از [0, 0] تا [0, 4] اجرا می شود.

سپس مقدار هر عنصر از آرایه را با استفاده از حلقه نشان می دهیم، اگر مقدار ردیف**(row)** برابر 0 و مقدار ستون**(col)**برابر 0 باشد مقدار عنصری که در ستون 1 و ردیف 1 **(**numbers[0, 0]**)**قرار دارد نشان داده خواهد شد که در مثال بالا عدد 1 است.

بعد از اینکه دومین حلقه تکرار به پایان رسید، فورا دستورات بعد از آن اجرا خواهند شد، که در اینجا دستور **()Console.WriteLine**که به برنامه اطلاع می دهد که به خط بعد برود.

سپس حلقه با اضافه کردن یک واحد به مقدار **row** این فرایند را دوباره تکرار می کند.

سپس دومین حلقه **for** اجرا شده و مقادیر دومین ردیف نمایش داده می شود.

این فرایند تا زمانی اجرا می شود که مقدار **row** کمتر از طول اولین بعد باشد.

حال بیایید انچه را از قبل یاد گرفته ایم در یک برنامه به کار بریم.این برنامه نمره چهار درس مربوط به سه دانش آموز را از ما می گیرد و معدل سه دانش آموز را حساب می کند.

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           double[,] studentGrades = new double[3, 4];
8           double total;
9
10          for (int student = 0; student < studentGrades.GetLength(0); student++)
11          {
12              total = 0;
13
14              Console.WriteLine("Enter grades for Student {0}", student + 1);
15
16              for (int grade = 0; grade < studentGrades.GetLength(1); grade++)
17              {
18                  Console.Write("Enter Grade #{0}: ", grade + 1);
19                  studentGrades[student, grade] =
20  Convert.ToDouble(Console.ReadLine());
21                  total += studentGrades[student, grade];
22              }
23
24              Console.WriteLine("Average is {0:F2}",
25                                (total / studentGrades.GetLength(1)));
26              Console.WriteLine();
27          }
28
29      }
    }
```

نتیجه

```
Enter grades for Student 1
Enter Grade #1: 92
Enter Grade #2: 87
Enter Grade #3: 89
Enter Grade #4: 95
Average is 90.75

Enter grades for Student 2
Enter Grade #1: 85
Enter Grade #2: 85
Enter Grade #3: 86
Enter Grade #4: 87
Average is 85.75

Enter grades for Student 3
Enter Grade #1: 90
Enter Grade #2: 90
Enter Grade #3: 90
Enter Grade #4: 90
Average is 90.00
```

در برنامه بالا یک آرایه چند بعدی از نوع **double** تعریف شده است **(خط 7)**.

همچنین یک متغیر به نام **total** تعریف می کنیم که مقدار محاسبه شده معدل هر دانش آموز را در آن قرار دهیم. حال وارد حلقه **for** تو در تو می شویم (خط 10). در اولین حلقه **for** یک متغیر به نام **sudent** برای تشخیص پایه درسی هر دانش آموز تعریف کرده ایم.

از متد **()GetLength** هم برای تشخیص تعداد دانش آموزان استفاده شده است. وارد بدنه حلقه **for** می شویم.در خط **12** مقدار متغیر **total** را برابر صفر قرار می دهیم.

بعدا مشاهده می کنید که چرا این کار را انجام دادیم.

سپس برنامه یک پیغام را نشان می دهد و از شما می خواهد که شماره دانش آموز را وارد کنید **(student + 1)**.

عدد 1 را به **student** اضافه کرده ایم تا به جای نمایش **Student 0**، با **Student 1** شروع شود، تا طبیعی تر به نظر برسد.

سپس به دومین حلقه **for** در خط **16** می رسیم.

در این حلقه یک متغیر شمارنده به نام **grade** تعریف می کنیم که طول دومین بعد آرایه را با استفاده از فراخوانی متد **GetLength(1)** به دست می آورد.

این طول تعداد نمراتی را که برنامه از سوال می کند را نشان می دهد.

برنامه چهار نمره مربوط به دانش آموز را می گیرد.

هر وقت که برنامه یک نمره را از کاربر دریافت می کند، نمره به متغیر **total** اضافه می شود.

وقتی همه نمره ها وارد شدند، متغیر **total** هم جمع همه نمرات را نشان می دهد.

در خطوط **24-23** معدل دانش آموز نشان داده می شود.

به فرمت **{0:F2}** توجه کنید.این فرمت معدل را تا دو رقم اعشار نشان می دهد.

معدل از تقسیم کردن **total** (جمع) بر تعداد نمرات به دست می آید.

از متد **GetLength(1)** هم برای به دست آوردن تعداد نمرات استفاده می شود.

آرایه های دندانه دار

آرایه های دندانه دار نوعی از آرایه های چند بعدی هستند که شامل ردیف هایی با طول های مختلفند. آرایه چند بعدی ساده آرایه ای به شکل مستطیل است چون تعداد ستونهای آن یکسان است ولی آرایه دندانه دار دارای ردیفهای با طول های متفاوت است. بنابر این می توان یک آرایه دندانه دار را آرایه ای از آرایه ها فرض کرد.

در زیر نحوه تعریف آرایه های چند بعدی آمده است.

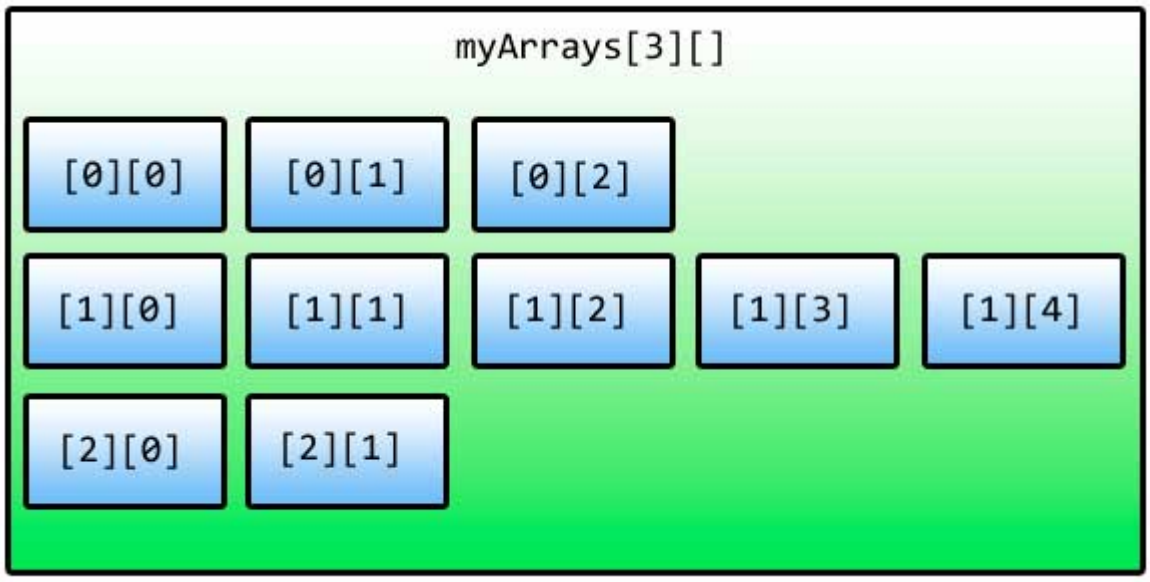
```
datatype[][] arrayName;
```

مقدار دهی به آرایه های دندانه دار بسیار گیج کننده است.

در زیر نحوه مقدار دهی به یک آرایه دندانه دار نشان داده شده است :

```
int[][] myArrays = new int[3][];

myArrays[0] = new int[3];
myArrays[1] = new int[5];
myArrays[2] = new int[2];
```



ابتدا تعداد ردیفهای آرایه را به وسیله کلمه کلیدی **new** تعریف می کنیم، و بعد نوع داده ای آرایه و سپس دو جفت کروشه که در جفت کروشه اول تعداد ردیف ها قرار دارد را تعریف می کنیم.

حال تعداد ستونها هر ردیف را با استفاده از سه ردیفی که در دسترس است و با استفاده از اندیسهای آنها مانند یک آرایه ساده مقدار دهی می کنیم. می توان به ستونهای هر ردیف مجموعه ای از مقادیر اختصاص داد :

```
int[][] myArrays = new int[3][];

myArrays[0] = new int[3] { 1, 2, 3 };
myArrays[1] = new int[5] { 5, 4, 3, 2, 1 };
myArrays[2] = new int[2] { 11, 22 };
```

یک راه بهتر برای مقداردهی به آرایه های دندانه دار به شکل زیر است :

```
int[][] myArrays = new int[3][] { new int[3] { 1, 2, 3 },
                                  new int[5] { 5, 4, 3, 2, 1 },
                                  new int[2] { 11, 22 } };
```

همچنین می توان از ذکر طول ردیفهای آرایه صرف نظر کرد :

```
int[][] myArrays = new int[][] { new int[] { 1, 2, 3 },
                                 new int[] { 5, 4, 3, 2, 1 },
                                 new int[] { 11, 22 } };
```

کد بالا را باز هم می توان ساده تر نوشت :

```
int[][] myArrays = { new int[] { 1, 2, 3 },
                     new int[] { 5, 4, 3, 2, 1 },
                     new int[] { 11, 22 } };
```

برای دسترسی به عناصر یک آرایه دندانه دار می توان از ستون ها و ردیف های آن استفاده کرد :

```
array[row][column]
```

```
Console.WriteLine(myArrays[1][2]);
```

از یک حلقه **foreach** ساده نمی توان برای دسترسی به اجزای این آرایه ها استفاده کرد.

```
foreach (int array in myArrays)
{
      Console.WriteLine(array);
}
```

اگر از حلقه **foreach** استفاده کنیم با خطا مواجه می شویم چون عناصر این نوع آرایه ها ، آرایه هستند نه عدد یا رشته یا... .برای حل این مشکل باید نوع متغیر موقتی **(array)** را تغییر داده و از حلقه **foreach** دیگری برای دسترسی به مقادیر استفاده کرد.

مثال :

```
foreach (int[] array in myArrays)
{
    foreach (int number in array)
    {
         Console.WriteLine(number);
    }
}
```

این کار با استفاده از یک حلقه **for** تو در تو قابل اجراست :

```
for (int row = 0; row < myArray.Length; row++)
{
    for (int col = 0; col < myArray[row].Length; col++)
    {
        Console.WriteLine(myArray[row][col]);
    }
}
```

در اولین حلقه **for** با استفاده از خاصیت **Length** ، **myArray** تعداد ردیف های آرایه را به دست می آوریم.در حلقه **for** دوم نیز با استفاده از خاصیت **Length** عنصر ردیف جاری تعداد ستونها را به دست می آوریم. سپس با استفاده از اندیس عناصر آرایه را چاپ می کنیم.

متدها

متدها به شما اجازه می دهند که یک رفتار یا وظیفه را تعریف کنید و مجموعه ای از کدها هستند که در هر جای برنامه می توان از آنها استفاده کرد.متدها دارای آرگومانهایی هستند که وظیفه متد را مشخص می کنند.متد در داخل کلاس تعریف می شود. نمی توان یک متد را در داخل متد دیگر تعریف کرد. وقتی که شما در برنامه یک متد را صدا می زنید برنامه به قسمت تعریف متد رفته و کدهای آن را اجرا می کند. در سی شارپ متدی وجود دارد که نقطه آغاز هر برنامه است و بدون آن برنامه ها نمی دانند با ید از کجا شروع شوند ، این متد **Main** نام دارد

پارامتر ها همان چیزهایی هستند که متد منتظر دریافت آنها است.

آرگومانها مقادیری هستند که به پارامترها ارسال می شوند.

گاهی اوقات دو کلمه پارامتر و آرگومان به یک منظور به کار می روند.

ساده ترین ساختار یک متد به صورت زیر است :

```
returnType MethodName()
{
   code to execute;
}
```

به برنامه ساده زیر توجه کنید.در این برنامه از یک متد برای چاپ یک پیغام در صفحه نمایش استفاده شده است :

```
1   using System;
2
3   public class Program
4   {
5       static void PrintMessage()
6       {
7           Console.WriteLine("Hello World!");
8       }
9
10      public static void Main()
11      {
12          PrintMessage();
13      }
14  }
```

نتیجه

```
Hello World!
```

در خطوط **5-8** یک متد تعریف کرده ایم.

مکان تعریف آن در داخل کلاس مهم نیست.

به عنوان مثال می توانید آن را زیر متد **Main** تعریف کنید.

می توان این متد را در داخل متد دیگر صدا زد **(فراخوانی کرد)**.

متد دیگر ما در اینجا متد **Main** است که می توانیم در داخل آن نام متدی که برای چاپ یک پیغام تعریف کرده ایم **(یعنی متد PrintMessage)** را صدا بزنیم. متد **Main** به صورت **static** تعریف شده است. برای اینکه بتوان از متد **()PrintMessage** در داخل متد **()Main** استفاده کنیم باید آن را به صورت **static** تعریف کنیم.

کلمه **static** به طور ساده به این معناست که می توان از متد استفاده کرد بدون اینکه از کلاس نمونه ای ساخته شود.

متد **Main** همواره باید به صورت **static** تعریف شود چون برنامه فورا و بدون نمونه سازی از کلاس از آن استفاده می کند.

وقتی به مبحث برنامه نویسی شی گرا رسیدید به طور دقیق کلمه **static** مورد بحث قرار می گیرد.

برنامه **class** (مثال بالا) زمانی اجرا می شود که برنامه دو متدی را که تعریف کرده ایم را اجرا کند و متد **main** به صورت **static** تعریف شود.

در باره این کلمه کلیدی در درسهای آینده مطالب بیشتری می آموزیم.

در تعریف متد بالا بعد از کلمه **static** کلمه کلیدی **void** آمده است که نشان دهنده آن است که متد مقدار برگشتی ندارد.

در درس آینده در مورد مقدار برگشتی از یک متد و استفاده از آن برای اهداف مختلف توضیح داده خواهد شد.

نام متد ما **PrintMessage** است.

به این نکته توجه کنید که در نامگذاری متد از روش پاسکال **(حرف اول هر کلمه بزرگ نوشته می شود)** استفاده کرده ایم.

این روش نامگذاری قراردادی است و می توان از این روش استفاده نکرد، اما پیشنهاد می شود که از این روش برای تشخیص متدها استفاده کنید.

بهتر است در نامگذاری متدها از کلماتی استفاده شود که کار ان متد را مشخص می کند مثلا نام هایی مانند **GoToBed** یا **OpenDoor**.

همچنین به عنوان مثال اگر مقدار برگشتی متد یک مقدار بولی باشد می توانید اسم متد خود را به صورت یک کلمه سوالی انتخاب کنید مانند **IsLeapyear** یا **IsTeenager** ...ولی از کذاشتن علامت سوال در آخر اسم متد خودداری کنید.

دو پرانتزی که بعد از نام می آید نشان دهنده آن است که نام متعلق به یک متد است. در این مثال در داخل پرانتز ها هیچ چیزی نوشته نشده چون پارامتری ندارد.در درسهای آینده در مورد متدها بیشتر توضیح می دهیم.

بعد از پرانتزها دو آکولاد قرار می دهیم که بدنه متد را تشکیل می دهد و کدهایی را که می خواهیم اجرا شوند را در داخل این آکولاد ها می نویسیم.

در داخل متد **Main** متدی را که در خط **12** ایجاد کرده ایم را صدا می زنیم.

برای صدا زدن یک متد کافیست نام آن را نوشته و بعد از نام پرانتزها را قرار دهیم.

اگر متد دارای پارامتر باشد باید شما آراگومانها را به ترتیب در داخل پرانتزها قرار دهید. در این مورد نیز در درسهای آینده توضیح بیشتری می دهیم.

با صدا زدن یک متد کدهای داخل بدنه آن اجرا می شوند.

برای اجرای متد PrintMessage() برنامه از متد Main به محل تعریف متد PrintMessage() می رود.

مثلا وقتی ما متد PrintMessage() را در خط 12 صدا می زنیم برنامه از خط 12 به خط 7 ، یعنی جایی که متد تعریف شده می رود.

اکنون ما یک متد در برنامه class داریم و همه متدهای این برنامه می توانند آن را صدا بزنند.

مقدار برگشتی از یک متد

متدها می توانند مقدار برگشتی از هر نوع داده ای داشته باشند. این مقادیر می توانند در محاسبات یا به دست آوردن یک داده مورد استفاده قرار بگیرند.
در زندگی روزمره فرض کنید که کارمند شما یک متد است و شما او را صدا می زنید و از او می خواهید که کار یک سند را به پایان برساند.
سپس از او میخواهید که بعد از اتمام کارش سند را به شما تحویل دهد.
سند همان مقدار برگشتی متد است.
نکته مهم در مورد یک متد، مقدار برگشتی و نحوه استفاده شما از آن است.
برگشت یک مقدار از یک متد آسان است. کافیست در تعریف متد به روش زیر عمل کنید :

```
returnType MethodName()
{
   return value;
}
```

returnType در اینجا نوع داده ای مقدار برگشتی را مشخص می کند **(int،bool،...)**.

در داخل بدنه متد کلمه کلیدی **return** و بعد از آن یک مقدار یا عبارتی که نتیجه آن یک مقدار است را می نویسیم.

نوع این مقدار برگشتی باید از انواع ساده بوده و در هنگام نامگذاری متد و قبل از نام متد ذکر شود.

اگر متد ما مقدار برگشتی نداشته باشد باید از کلمه **void** قبل از نام متد استفاده کنیم.

مثال زیر یک متد که دارای مقدار برگشتی است را نشان می دهد.

```
1  using System;
2
3  public class Program
4  {
5      static int CalculateSum()
6      {
7          int firstNumber = 10;
8          int secondNumber = 5;
9
10         int sum = firstNumber + secondNumber;
11
12         return sum;
13     }
14
15     public static void Main()
16     {
17         int result = CalculateSum();
18
19         Console.WriteLine("Sum is {0}.", result);
20     }
21 }
Sum is 15.
```

همانطور که در خط **5** مثال فوق مشاهده می کنید هنگام تعریف متد از کلمه **int** به جای **void** استفاده کرده ایم که نشان دهنده آن است که متد ما دارای مقدار برگشتی از نوع اعداد صحیح است.

در خطوط **7** و **8** دو متغیر تعریف و مقدار دهی شده اند.

توجه کنید که این متغیرها، متغیرهای محلی هستند. و این بدان معنی است که این متغیرها در سایر متدها مانند متد **Main** قابل دسترسی نیستند و فقط در متدی که در آن تعریف شده اند قابل استفاده هستند.

در خط **10** جمع دو متغیر در متغیر **sum** قرار می گیرد.

در خط **12** مقدار برگشتی **sum** توسط دستور **return** فراخوانی می شود.

در داخل متد **Main** یک متغیربه نام **result** در خط **17** تعریف می کنیم و متد **()CalculateSum** را فراخوانی می کنیم.

متد **()CalculateSum** مقدار **158** را بر می گرداند که در داخل متغیر **result** ذخیره می شود.

در خط **19** مقدار ذخیره شده در متغیر **result** چاپ می شود.

متدی که در این مثال ذکر شد متد کاربردی و مفیدی نیست،با وجودیکه کدهای زیادی در متد بالا نوشته شده ولی همیشه مقدار برگشتی **15** است، در حالیکه می توانستیم به راحتی یک متغیر تعریف کرده و مقدار **15** را به آن اختصاص دهیم. این متد در صورتی کارامد است که پارامترهایی به آن اضافه شود که در درسهای آینده توضیح خواهیم داد. هنگامی که می خواهیم در داخل یک متد از دستور **if** یا **switch** استفاده کنیم باید تمام کدها دارای مقدار برگشتی باشند.

برای درک بهتر این مطلب به مثال زیر توجه کنید :

```
1   using System;
2
3   public class Program
4   {
5       static int GetNumber()
6       {
7           int number;
8
9           Console.Write("Enter a number greater than 10: ");
10          number = Convert.ToInt32(Console.ReadLine());
11
12          if (number > 10)
13          {
14              return number;
15          }
16          else
17          {
18              return 0;
19          }
20      }
21
22      public static void Main()
23      {
24          int result = GetNumber();
25
26          Console.WriteLine("Result = {0}.", result);
27      }
28  }
```

```
Enter a number greater than 10: 11
Result = 11
Enter a number greater than 10: 9
Result = 0
```

در خطوط 5-20 یک متد با نام GetNumber() تعریف شده است که از کاربر یک عدد بزرگتر از 10 را می خواهد.

اگر عدد وارد شده توسط کاربر درست نباشد متد مقدار صفر را بر می گرداند.

و اگر قسمت else دستور if و یا دستور return را از آن حذف کنیم در هنگام اجرای برنامه با پیغام خطا مواجه می شویم.

چون اگر شرط دستور if نادرست باشد (کاربر مقداری کمتر از 10 را وارد کند) برنامه به قسمت else می رود تا مقدار صفر را بر گرداند و چون قسمت else حذف شده است برنامه با خطا مواجه می شود و همچنین اگر دستور return حذف شود چون برنامه نیاز به مقدار برگشتی دارد پیغام خطا می دهد.

و آخرین مطلبی که در این درس می خواهیم به شما آموزش دهیم این است که شما می توانید از یک متد که مقدار برگشتی ندارد خارج شوید.

حتی اگر از نوع داده ای void در یک متد استفاده می کنید باز هم می توانید کلمه کلیدی return را در آن به کار ببرید. استفاده از return باعث خروج از بدنه متد و اجرای کدهای بعد از آن می شود.

```
1   using System;
2
3   public class Program
4   {
5       static void TestReturnExit()
6       {
7           Console.WriteLine("Line 1 inside the method TestReturnExit()");
8           Console.WriteLine("Line 2 inside the method TestReturnExit()");
9
10          return;
11
12          //The following lines will not execute
13          Console.WriteLine("Line 3 inside the method TestReturnExit()");
14          Console.WriteLine("Line 4 inside the method TestReturnExit()");
15      }
16
17      public static void Main()
18      {
19          TestReturnExit();
20          Console.WriteLine("Hello World!");
21      }
22  }
```

```
Line 1 inside the method TestReturnExit()
Line 2 inside the method TestReturnExit()
Hello World!
```

در برنامه بالا نحوه خروج از متد با استفاده از کلمه کلیدی return و نادیده گرفتن همه کدهای بعد از این کلمه کلیدی نشان داده شده است. در پایان برنامه متد تعریف شده (TestReturnExit()) در داخل متد Main فراخوانی و اجرا می شود.

پارامترها و آرگومان ها

پارامترها داده های خامی هستند که متد آنها را پردازش می کند و سپس اطلاعاتی را که به دنبال آن هستید در اختیار شما قرار می دهد.

فرض کنید پارامترها مانند اطلاعاتی هستند که شما به یک کارمند می دهید که بر طبق آنها کارش را به پایان برساند. یک متد می تواند هر تعداد پارامتر داشته باشد. هر پارامتر می تواند از انواع مختلف داده باشد.

در زیر یک متد با **N** پارامتر نشان داده شده است :

```
returnType MethodName(datatype param1, datatype param2, ... datatype paramN)
{
   code to execute;

}
```

پارامترها بعد از نام متد و بین پرانتزها قرار می گیرند. بر اساس کاری که متد انجام می دهد می توان تعداد پارامترهای زیادی به متد اضافه کرد. بعد از فراخوانی یک متد باید آرگومانهای آن را نیز تامین کنید. آرگومانها مقادیری هستند که به پارامترها اختصاص داده می شوند. ترتیب ارسال آرگومانها به پارامترها مهم است. عدم رعایت ترتیب در ارسال آرگومانها باعث به وجود آمدن خطای منطقی و خطای زمان اجرا می شود.

اجازه بدهید که یک مثال بزنیم :

```
using System;

public class Program
{
    static int CalculateSum(int number1, int number2)
    {
        return number1 + number2;
    }

    public static void Main()
    {
        int num1, num2;

        Console.Write("Enter the first number: ");
        num1 = Convert.ToInt32(Console.ReadLine());
        Console.Write("Enter the second number: ");
        num2 = Convert.ToInt32(Console.ReadLine());

        Console.WriteLine("Sum = {0}", CalculateSum(num1, num2));
    }
}
```

نتیجه

```
Enter the first number: 10
Enter the second number: 5
Sum = 15
```

در برنامه بالا یک متد به نام **CalculateSum** (خطوط **5-8**) تعریف شده است که وظیفه آن جمع مقدار دو عدد است. چون این متد مقدار دو عدد صحیح را با هم جمع می کند پس نوع برگشتی ما نیز باید **int** باشد. متد دارای دو پارامتر است که اعداد را به آنها ارسال می کنیم. به نوع داده ای پارامترها توجه کنید. هر دو پارامتر یعنی **number1** و **number2** مقادیری از نوع اعداد صحیح **(int)** دریافت می کنند. در بدنه متد دستور **return** نتیجه جمع دو عدد را بر می گرداند. در داخل متد **Main** برنامه از کاربر دو مقدار را درخواست می کند و آنها را داخل متغیر ها قرار می دهد. حال متد را که آرگومانهای آن را آماده کرده ایم فراخوانی می کنیم. مقدار **num1** به پارامتر اول و مقدار **num2** به پارامتر دوم ارسال می شود. حال اگر مکان دو مقدار را هنگام ارسال به متد تغییر دهیم **(یعنی مقدار num2** به پارامتر اول و مقدار **num1** به پارامتر دوم ارسال شود**)** هیچ تغییری در نتیجه متد ندارد چون جمع خاصیت جابه جایی دارد.

فقط به یاد داشته باشید که باید ترتیب ارسال آرگومانها هنگام فراخوانی متد دقیقا با ترتیب قرار گیری پارامترها تعریف شده در متد مطابقت داشته باشد. بعد از ارسال مقادیر **10** و **5** به پارامترها ، پارامترها آنها را دریافت می کنند. به این نکته نیز توجه کنید که نام پارامترها طبق قرارداد به شیوه کوهان شتری یا **camelCasing** (حرف اول دومین کلمه بزرگ نوشته می شود**)** نوشته می شود. در داخل بدنه متد **(خط 7)** دو مقدار با هم جمع می شوند و نتیجه به متد فراخوان (متدی که متد **()CalculateSum** را فراخوانی می کند**)** ارسال می شود.

در درس آینده از یک متغیر برای ذخیره نتیجه محاسبات استفاده می کنیم ولی در اینجا مشاهده می کنید که میتوان به سادگی نتیجه جمع را نشان داد **(خط 7)**. در داخل متد **Main** از ما دو عدد که قرار است با هم جمع شوند درخواست می شود.

در خط **19** متد **()CalculateSum** را فراخوانی می کنیم و دو مقدار صحیح به آن ارسال می کنیم.

دو عدد صحیح در داخل متد با هم جمع شده و نتیجه آنها برگردانده می شود. مقدار برگشت داده شده از متد به وسیله متد **()WriteLine** از کلاس **console** نمایش داده می شود. **(خط 19)**

در برنامه زیر یک متد تعریف شده است که دارای دو پارامتر از دو نوع داده ای مختلف است:

```
1   using System;
2
3   public class Program
4   {
5       static void ShowMessageAndNumber(string message, int number)
6       {
7           Console.WriteLine(message);
8           Console.WriteLine("Number = {0}", number);
9       }
10
11      public static void Main()
12      {
13          ShowMessageAndNumber("Hello World!", 100);
14      }
15  }
```

نتیجه

```
Hello World!
Number = 100
```

در مثال بالا یک متدی تعریف شده است که اولین پارامتر آن مقداری از نوع رشته و دومین پارامتر آن مقداری از نوع **int** دریافت می کند. متد به سادگی دو مقداری که به آن ارسال شده است را نشان می دهد. در خط **13** متد را اول با یک رشته و سپس یک عدد خاص فراخوانی می کنیم . حال اگر متد به صورت زیر فراخوانی می شد :

```
ShowMessageAndNumber(100, "Welcome to Gimme C#!");
```

در برنامه خطا به وجود می آمد چون عدد **100** به پارامتری از نوع رشته و رشته **Hello World!** به پارامتری از نوع اعداد صحیح ارسال می شد.این نشان می دهد که ترتیب ارسال آرگومانها به پارامترها هنگام فراخوانی متد مهم است.

به مثال **1** توجه کنید در آن مثال دو عدد از نوع **int** به پارامترها ارسال کردیم که ترتیب ارسال آنها چون هردو پارامتر از یک نوع بودند مهم نبود، ولی اگر پارامترهای متد دارای اهداف خاصی باشند ترتیب ارسال آرگومانها مهم است.

```
void ShowPersonStats(int age, int height)
{
    Console.WriteLine("Age = {0}", age);
    Console.WriteLine("Height = {0}", height);
}

//Using the proper order of arguments
ShowPersonStats(20, 160);

//Acceptable, but produces odd results
ShowPersonStats(160, 20);
```

در مثال بالا نشان داده شده است که حتی اگر متد دو آرگومان با یک نوع داده ای قبول کند باز هم بهتر است ترتیب بر اساس تعریف پارامترها رعایت شود. به عنوان مثال در اولین فراخوانی متد بالا اشکالی به چشم نمی آید چون سن شخص **20** و قد او **160** سانتی متر است. اگر آرگومانها را به ترتیب ارسال نکنیم سن شخص **160** و قد او **20** سانتی متر می شود که به واقعیت نزدیک نیست.

دانستن مبانی مقادیر برگشتی و ارسال آرگومانها باعث می شود که شما متدهای کارامد تری تعریف کنید. تکه کد زیر نشان می دهد که شما حتی می توانید مقدار برگشتی از یک متد را به عنوان آرگومان به متد دیگر ارسال کنید.

```
int MyMethod()
{
   return 5;
}

void AnotherMethod(int number)
{
   Console.WriteLine(number);
}

// Codes skipped for demonstration

AnotherMethod(MyMethod());
```

چون مقدار برگشتی متد **MyMethod()** عدد **5** است و به عنوان آرگومان به متد **AnotherMethod()** ارسال می شود خروجی کد بالا هم عدد **5** است.

نامیدن آرگومان ها

یکی دیگر از راه های ارسال آرگومانها استفاده از نام آنهاست. استفاده از نام آرگومانها شما را از به یاد آوری و رعایت ترتیب پارامترها هنگام ارسال آرگومان ها راحت می کند. در عوض شما باید نام پارامترهای متد را به خاطر بسپارید (ولی از آن جاییکه ویژوال استودیو Intellisense دارد نیازی به این کار نیست.)

استفاده از نام آرگومانها خوانایی برنامه را بالا می برد چون شما می توانید ببینید که چه مقادیری به چه پارامترهایی اختصاص داده شده است.

نامیدن آرگومانها در سی شارپ 2010 مطرح شده است و اگر شما از نسخه های قبلی مانند سی شارپ 2008 استفاده می کنید نمی توانید از این خاصیت استفاده کنید.

در زیر نحوه استفاده از نام آرگومانها وقتی که متد فراخوانی می شود نشان داده شده است :

```
MethodToCall( paramName1: value, paramName2: value, ... paramNameN: value);
```

حال به مثال زیر توجه کنید :

```
using System;

public class Program
{
    static void SetSalaries(decimal jack, decimal andy, decimal mark)
    {
        Console.WriteLine("Jack's salary is {0:C}.", jack);
        Console.WriteLine("Andy's salary is {0:C}.", andy);
        Console.WriteLine("Mark's salary is {0:C}.", mark);
    }

    public static void Main()
    {
        SetSalaries(jack: 120, andy: 30, mark: 75);

        //Print a newline
        Console.WriteLine();

        SetSalaries(andy: 60, mark: 150, jack: 50);

        Console.WriteLine();

        SetSalaries(mark: 35, jack: 80, andy: 150);
    }
}
```

نتیجه

```
Jack's salary is $120.
Andy's salary is $30.
Mark's salary is $75.

Jack's salary is $50.
Andy's salary is $60.
Mark's salary is $150.

Jack's salary is $80.
Andy's salary is $150.
Mark's salary is $35.
```

متد **()WriteLine** در خطوط **9-7** از فرمت پول رایج که با **{0:C}** نشان داده می شود استفاده کرده است که یک داده عددی را به نوع پولی تبدیل می کند.

خروجی نشان می دهد که حتی اگر ما ترتیب آرگومانها در سه متد فراخوانی شده را تغییر دهیم مقادیر مناسب به پارامترهای مربوطه شان اختصاص داده می شود.

همچنین می توان از آرگومانهای دارای نام و آرگومانهای ثابت **(مقداری)** به طور همزمان استفاده کرد به شرطی که آرگومانهای ثابت قبل از آرگومانهای دارای نام قرار بگیرند.

```
//Assign 30 for Jack's salary and use named arguments for
// the assignment of the other two

SetSalary(30, andy: 50, mark: 60);

// or

SetSalary(30, mark: 60, andy: 50);

//The following codes are wrong and will lead to errors

SetSalary(mark: 60, andy: 50, 30);

// and

SetSalary(mark: 60, 30, andy: 50);
```

همانطور که مشاهده می کنید ابتدا باید آرگومانهای ثابت هنگام فراخوانی متد ذکر شوند.

در اولین و دومین فراخوانی در کد بالا ، مقدار **30** را به عنوان اولین آرگومان به اولین پارامتر متد یعنی **jack** اختصاص می دهیم.

سومین و چهارمین خط کد بالا اشتباه هستند چون آرگومانهای دارای نام قبل از آرگومانهای ثابت قرار گرفته اند.

قرار گرفتن آرگومانهای دارای نام بعد از آرگومانها ثابت از بروز خطا جلوگیری می کند.

ارسال آرگومان ها به روش ارجاع

آرگومانها را می توان به کمک ارجاع ارسال کرد. این بدان معناست که شما آدرس متغیری را ارسال می کنید نه مقدار آن را.

ارسال با ارجاع زمانی مفید است که شما بخواهید یک آرگومان که دارای مقدار بزرگی است (مانند یک آبجکت) را ارسال کنید.

در این حالت وقتی که آرگومان ارسال شده را در داخل متد اصلاح می کنیم مقدار اصلی آرگومان در خارج از متد هم تغییر می کند.

در زیر دستورالعمل پایه ای تعریف پارامتر ها که در آنها به جای مقدار از آدرس استفاده شده است نشان داده شده :

```
returnType MethodName(ref datatype param1)
{
   code to execute;
}
```

فراموش نشود که باید از کلمه کلیدی **ref** استفاده کنید. وقتی یک متد فراخوانی می شود و آرگومانها به آنها ارسال می شود هم باید از کلمه کلیدی **ref** استفاده شود.

```
MethodName(ref argument);
```

اجازه دهید که تفاوت بین ارسال با ارجاع و ارسال با مقدار آرگومان را با یک مثال توضیح دهیم.

```
1   using System;
2
3   public class Program
4   {
5       static void ModifyNumberVal(int number)
6       {
7           number += 10;
8           Console.WriteLine("Value of number inside method is {0}.", number);
9       }
10
11      static void ModifyNumberRef(ref int number)
12      {
13          number += 10;
14          Console.WriteLine("Value of number inside method is {0}.", number);
15      }
16
17      public static void Main()
18      {
19          int num = 5;
20
21          Console.WriteLine("num = {0}\n", num);
22
23          Console.WriteLine("Passing num by value to method ModifyNumberVal() ...");
24          ModifyNumberVal(num);
25          Console.WriteLine("Value of num after exiting the method is {0}.\n", num);
26
27          Console.WriteLine("Passing num by ref to method ModifyNumberRef() ...");
28          ModifyNumberRef(ref num);
```

```
29          Console.WriteLine("Value of num after exiting the method is {0}.\n", num);
30      }
31  }
```

```
num = 5

Passing num by value to method ModifyNumberVal() ...
Value of number inside method is 15.
Value of num after exiting the method is 5.

Passing num by ref to method ModifyNumberRef() ...
Value of number inside method is 15.
Value of num after exiting the method is 15.
```

در برنامه بالا دو متد که دارای یک هدف یکسان هستند تعریف شده اند و آن اضافه کردن عدد 10 به مقداری است که به آنها ارسال می شود.

اولین متد (خطوط 9-5) دارای یک پارامتر است که نیاز به یک مقدار آرگومان (از نوع int) دارد.

وقتی که متد را صدا می زنیم و آرگومانی به آن اختصاص می دهیم (خط 24) ،کپی آرگومان به پارامتر متد ارسال می شود.

بنابراین مقدار اصلی متغیر خارج از متد هیچ ارتباطی به پارامتر متد ندارد،سپس مقدار 10 را به متغیر پارامتر(number) اضافه کرده و نتیجه را چاپ می کنیم.

برای اثبات اینکه متغیر num هیچ تغییری نکرده است مقدار آن را یکبار دیگر چاپ کرده و مشاهده می کنیم که تغییر ی نکرده است.

دومین متد (خطوط 15-11) نیاز به یک مقدار با ارجاع دارد.

در این حالت به جای اینکه یک کپی از مقدار به عنوان آرگومان به ان ارسال شود آدرس متغیر به آن ارسال می شود.

حال پارامتر به مقدار اصلی متغیر که زمان فراخوانی متد به آن ارسال می شود دسترسی دارد.

وقتی که ما مقدار متغیر پارامتری که شامل آدرس متغیر اصلی است را تغییر می دهیم (خط 13) در واقع مقدار متغیر اصلی در خارج از متد را تغییر داده ایم.

در نهایت مقدار اصلی متغیر را وقتی که از متد خارج شدیم را نمایش می دهیم و مشاهده می شود که مقدار آن واقعا تغییر کرده است.

پارامترهای out

پارامترهای **out** پارامترهایی هستند که متغیرهایی را که مقدار دهی اولیه نشده اند را قبول می کنند.

کلمه کلیدی **out** زمانی مورد استفاده قرار می گیرد که بخواهیم یک متغیر بدون مقدار را به متد ارسال کنیم.

متغیر بدون مقدار اولیه ، متغیری است که مقداری به آن اختصاص داده نشده است.

در این حالت متد یک مقدار به متغیر می دهد.

ارسال متغیر مقداردهی نشده به متد زمانی مفید است که شما بخواهید از طریق متد متغیر را مقدار دهی کنید.

استفاده از کلمه کلیدی **out** باعث ارسال آرگومان به روش ارجاع می شود نه مقدار.

به مثال زیر توجه کنید :

```
using System;

public class Program
{
    static void GiveValue(out int number)
    {
        number = 10;
    }

    public static void Main()
    {
        //Uninitialized variable
        int myNumber;

        GiveValue(out myNumber);

        Console.WriteLine("myNumber = {0}", myNumber);
    }
}
```

نتیجه

```
myNumber = 10
```

از کلمه کلیدی **out** برای پارامترهای متد استفاده شده است بنابراین می توانند متغیرهای مقداردهی نشده را قبول کنند.

در متد **Main** ،خط **15** متد را فراخوانی می کنیم و قبل از آرگومان کلمه کلیدی **out** را قرار می دهیم، متغیر مقداردهی نشده **(myNumber)** به متد ارسال می شود و در آنجا مقدار **10** به آن اختصاص داده می شود **(خط 7)**، مقدار **myNumber**در خط **17** نمایش داده می شود و مشاهده می کنید که مقدارش برابر مقداری است که در داخل متد به آن اختصاص داده شده است **(یعنی 10)**. استفاده از پارامترهای **out** بدین معنا نیست که شما همیشه نیاز دارید که آرگومانهای مقداردهی نشده را به متد ارسال کنید بلکه آرگومانهایی که شامل مقدار هستند را هم می توان به متد ارسال کرد، این کار درحکم استفاده از کلمه کلیدی **ref** است.

ارسال آرایه به عنوان آرگومان

می توان آرایه ها را به عنوان آرگومان به متد ارسال کرد. ابتدا شما باید پارامترهای متد را طوری تعریف کنید که آرایه دریافت کنند.

به مثال زیر توجه کنید.

```
1   using System;
2
3   namespace ArraysAsArgumentsDemo1
4   {
5       public class Program
6       {
7           static void TestArray(int[] numbers)
8           {
9               foreach (int number in numbers)
10              {
11                  Console.WriteLine(number);
12              }
13          }
14
15          public static void Main()
16          {
17              int[] array = { 1, 2, 3, 4, 5 };
18
19              TestArray(array);
20          }
21      }
22  }
```

نتیجه

```
1
2
3
4
5
```

مشاهده کردید که به سادگی می توان با گذاشتن کروشه بعد از نوع داده ای پارامتر یک متد ایجاد کرد که پارامتر آن ، آرایه دریافت می کند. وقتی متد در خط 17 فراخوانی می شود، آرایه را فقط با استفاده از نام آن و بدون استفاده از اندیس ارسال می کنیم. پس آرایه ها هم به روش ارجاع به متدها ارسال می شوند. در خطوط 10-7 از حلقه foreach برای دسترسی به اجزای اصلی آرایه که به عوان آرگومان به متد ارسال کرده ایم استفاده می کنیم. در زیر نحوه ارسال یک آرایه به روش ارجاع نشان داده شده است.

```
using System;

namespace ArraysAsArgumentsDemo2
{
    public class Program
    {
        static void IncrementElements(int[] numbers)
        {
            for (int i = 0; i < numbers.Length; i++)
            {
                numbers[i]++;
            }
        }

        public static void Main()
        {
            int[] array = { 1, 2, 3, 4, 5 };

            IncrementElements(array);

            foreach (int num in array)
            {
                Console.WriteLine(num);
            }
        }
    }
}
```

نتیجه

```
2
3
4
5
6
```

برنامه بالا یک متد را نشان می دهد که یک آرایه را دریافت می کند و به هر یک از عناصر آن یک واحد اضافه می کند. به این نکته توجه کنید که از حلقه foreach نمی توان برای افزایش مقادیر آرایه استفاده کنیم چون این حلقه برای خواندن مقادیر آرایه مناسب است نه اصلاح آنها. در داخل متد ما مقادیر هر یک از اجزای آرایه را افزایش داده ایم..سپس از متد خارج شده و نتیجه را نشان می دهیم. مشاهده می کنید که هر یک از مقادیر اصلی متد هم اصلاح شده اند. راه دیگر برای ارسال آرایه به متد ، مقدار دهی مستقیم به متد فراخوانی شده است.

به عنوان مثال :

```
IncrementElements(new int[] { 1, 2, 3, 4, 5 });
```

در این روش ما آرایه ای تعریف نمی کنیم بلکه مجموعه ای از مقادیر را به پارامتر ارسال می کنیم که آنها را مانند آرایه قبول کند. از آنجاییکه در این روش آرایه ای تعریف نکرده ایم نمی توانیم در متد Main نتیجه را چاپ کنیم.

اگر از چندین پارامتر در متد استفاده می کنید همیشه برای هر یک از پارامترهایی که آرایه قبول می کنند از یک جفت کروشه استفاده کنید.

به عنوان مثال :

```
void MyMethod(int[] param1, int param2)
{
//code here
}
```

به پارامترهای متد بالا توجه کنید ، پارامتر اول (param1) آرگومانی از جنس آرایه قبول می کند ولی پارامتر دوم (param2) یک عدد صحیح.

حال اگر پارامتر دوم(param2) هم آرایه قبول می کرد باید برای آن هم از کروشه استفاده می کردیم:

```
void MyMethod(int[] param1, int[] param2)
{
//code here
}
```

کلمه کلیدی params

کلمه کلیدی **params** امکان ارسال تعداد دلخواه پارامترهای همنوع و ذخیره آنها در یک آرایه ساده را فراهم می آورد.

کد زیر طریقه استفاده از کلمه کلیدی **params** را نشان می دهد :

```
using System;

public class Program
{
    static int CalculateSum(params int[] numbers)
    {
        int total = 0;

        foreach (int number in numbers)
        {
            total += number;
        }

        return total;
    }

    public static void Main()
    {
        Console.WriteLine("1 + 2 + 3 = {0}", CalculateSum(1, 2, 3));

        Console.WriteLine("1 + 2 + 3 + 4 = {0}", CalculateSum(1, 2, 3, 4));

        Console.WriteLine("1 + 2 + 3 + 4 + 5 = {0}", CalculateSum(1, 2, 3, 4, 5));

    }
}
```

```
1 + 2 + 3 = 6
1 + 2 + 3 + 4 = 10
1 + 2 + 3 + 4 + 5 = 15
```

از کلمه کلیدی **params** قبل از نوع داده ای آرایه پارامتر استفاده می شود **(مثال بالا)**. حال متد را سه بار با تعداد مختلف آرگومانها فراخوانی می کنیم.

این آرگومانها در داخل یک پارامتر از نوع آرایه ذخیره می شوند. با استفاده از حلقه **foreach** این آرگومانها را جمع و به متد فراخوان برگشت داده می شود.

وقتی از چندین پارامتر در یک متد استفاده می کنید فقط یکی از آنها باید دارای کلمه کلیدی **params** بوده و همچنین از لحاظ مکانی باید آخرین پارامتر باشد.

اگر این پارامتر**(**پارامتری که دارای کلمه کلیدی **params** است**)** در آخر پارامترهای دیگر قرار نگیرد و یا از چندین پارامتر **params** دار استفاده کنید با خطا مواجه می شوید.

به مثالهای اشتباه و درست زیر توجه کنید :

```
void SomeFunction(params int[] x, params int[] y) //ERROR

void SomeFunction(params int[] x, int y, int z) //ERROR

void SomeFunction(int x, int y, params int[] z) //Correct
```

محدود متغیر

متدها در سی شارپ دارای محدوده هستند، محدوده یک متغیر به شما می گوید که در کجای برنامه می توان از متغیر استفاده کرد و یا متغیر قابل دسترسی است، به عنوان مثال متغیری که در داخل یک متد تعریف می شود فقط در داخل بدنه متد قابل دسترسی است، می توان دو متغیر با نام یکسان در دو متد مختلف تعریف کرد.

برنامه زیر این ادعا را اثبات می کند :

```
using System;

public class Program
{
    static void DemonstrateScope()
    {
        int number = 5;

        Console.WriteLine("number inside method DemonstrateScope() = {0}", number);
    }

    public static void Main()
    {
        int number = 10;

        DemonstrateScope();

        Console.WriteLine("number inside the Main method = {0}", number);
    }
}
```

نتیجه

```
number inside method DemonstrateScope() = 5
number inside the Main method = 10
```

مشاهده می کنید که حتی اگر ما دو متغیر با نام یکسان تعریف کنیم که دارای محدوده های متفاوتی هستند، می توان به هر کدام از آنها مقادیر مختلفی اختصاص داد، متغیر تعریف شده در داخل متد **Main** هیچ ارتباطی به متغیر داخل متد **DemonstrateScope()** ندارد، وقتی به مبحث کلاسها رسیدیم در این باره بیشتر توضیح خواهیم داد.

پارامترهای اختیاری

پارامترهای اختیاری همانگونه که از اسمشان پیداست اختیاری هستند و می توان به آنها آرگومان ارسال کرد یا نه. این پارامترها دارای مقادیر پیشفرضی هستند.اگر به اینگونه پارامترها آرگومانی ارسال نشود از مقادیر پیشفرض استفاده می کنند.

به مثال زیر توجه کنید :

```
1  using System;
2
3  public class Program
4  {
5      static void PrintMessage(string message = "Welcome to Visual C# Tutorials!")
6      {
7          Console.WriteLine(message);
8      }
9
10     public static void Main()
11     {
12         PrintMessage();
13
14         PrintMessage("Learn C# Today!");
15     }
16 }
```

نتیجه :

```
Welcome to Visual C# Tutorials!
Learn C# Today!
```

متد **PrintMessage()** (خطوط **5-8**)یک پارامتر اختیاری دارد. برای تعریف یک پارامتر اختیاری می توان به آسانی و با استفاده از علامت **=** یک مقدار را به یک پارامتر اختصاص داد (مثال بالا خط **5**). دو بار متد را فراخوانی می کنیم. در اولین فراخوانی (خط **12**) ما آرگومانی به متد ارسال نمی کنیم بنابراین متد از مقدار پیشفرض (**Welcome to Visual C# Tutorials!**) استفاده می کند. در دومین فراخوانی(خط **14**) یک پیغام (آرگومان) به متد ارسال می کنیم که جایگزین مقدار پیشفرض پارامتر می شود. اگر از چندین پارامتر در متد استفاده می کنید همه پارامترهای اختیاری باید در آخر بقیه پارامترها ذکر شوند.

به مثالهای زیر توجه کنید.

```
void SomeMethod(int opt1 = 10, int opt2 = 20, int req1, int req2) //ERROR

void SomeMethod(int req1, int opt1 = 10, int req2, int opt2 = 20) //ERROR

void SomeMethod(int req1, int req2, int opt1 = 10, int opt2 = 20) //Correct
```

وقتی متدهای با چندین پارامتر اختیاری فراخوانی می شوند باید به پارامترهایی که از لحاظ مکانی در آخر بقیه پارامترها نیستند مقدار اختصاص داد.

به یاد داشته باشید که نمی توان برای نادیده گرفتن یک پارامتر به صورت زیر عمل کرد :

```
void SomeMethod(int required1, int optional1 = 10, int optional2 = 20)
{
   //Some Code
}

// ... Code omitted for demonstration

SomeMethod(10, , 100); //Error
```

اگر بخواهید از یک پارامتر اختیاری که در آخر پارامترهای دیگر نیست رد شوید و آن را نادیده بگیرید باید به از نام پارامترها استفاده کنید.

```
SomeMethod(10, optional2: 100);
```

برای استفاده از نام آرگومانها شما به راحتی می توانید نام مخصوص پارامتر و بعد از نام علامت کالن (:) و بعد مقدار اختصاص شده به آن را نوشت مانند (`optional2: 100`**) .**

متد بالا هیچ آرگومانی برای پارامتر اختیاری optional1 ندارد بنابراین این پارامتر از مقدار پیشفرضی که در زمان تعریف متد به آن اختصاص داده شده است استفاده می کند.

سربارگذاری متدها

سر بارگذاری متدها به شما اجازه می دهد که دو متد با نام یکسان تعریف کنید که دارای امضا و تعداد پارامترهای مختلف هستند.برنامه از روی آرگومانهایی که شما به متد ارسال می کنید به صورت خودکار تشخیص می دهد که کدام متد را فراخوانی کرده اید یا کدام متد مد نظر شماست.

امضای یک متد نشان دهنده ترتیب و نوع پارامترهای آن است.

به مثال زیر توجه کنید :

```
void MyMethod(int x, double y, string z)
```

که امضای متد بالا

```
MyMethod(int, double, string)
```

به این نکته توجه کنید که نوع برگشتی و نام پارامترها شامل امضای متد نمی شوند.

در مثال زیر نمونه ای از سربارگذاری متد ها آمده است.

```
1   using System;
2
3   namespace MethodOverloadingDemo
4   {
5       public class Program
6       {
7           static void ShowMessage(double number)
8           {
9               Console.WriteLine("Double version of the method was called.");
10          }
11
12          static void ShowMessage(int number)
13          {
14              Console.WriteLine("Integer version of the method was called.");
15          }
16
17          static void Main()
18          {
19              ShowMessage(9.99);
20              ShowMessage(9);
21          }
22      }
23  }
```

نتیجه

```
Double version of the method was called.
Integer version of the method was called.
```

در برنامه بالا دو متد با نام مشابه تعریف شده اند. اگر سربارگذاری متد توسط سی شارپ پشتیبانی نمی شد برنامه زمان زیادی برای انتخاب یک متد از بین متدهایی که فراخوانی می شوند لازم داشت. رازی در نوع پارامترهای متد نهفته است. کامپایلر بین دو یا چند متد در صورتی فرق می گذارد که پارامترهای متفاوتی داشته باشند.

وقتی یک متد را فراخوانی می کنیم ، متد نوع آرگومانها را تشخیص می دهد.

در فراخوانی اول (خط 19) ما یک مقدار double را به متد ()ShowMessage ارسال کرده ایم در نتیجه متد ()ShowMessage (خطوط 7-10) که دارای پارامتری از نوع double اجرا می شود.

در بار دوم که متد فراخوانی می شود (خط 20) ما یک مقدار int را به متد ()ShowMessage ارسال می کنیم متد ()ShowMessage (خطوط 12-15) که دارای پارامتری از نوع int است اجرا می شود.

معنای اصلی سربارگذاری متد همین است که توضیح داده شد.

هدف اصلی از سربارگذاری متدها این است که بتوان چندین متد که وظیفه یکسانی انجام می دهند را تعریف کرد

تعداد زیادی از متدها در کلاسهای دات نت سربارگذاری می شوند مانند متد ()WriteLine از کلاس Console.

قبلا مشاهده کردید که این متد می تواند یک آرگومان از نوع رشته دریافت کند و آن را نمایش دهد، و در حالت دیگر می تواند دو یا چند آرگومان قبول کند.

بازگشت

بازگشت فرایندی است که در آن متد مدام خود را فراخوانی می کند تا زمانی که به یک مقدار مورد نظر برسد.

بازگشت یک مبحث پیچیده در برنامه نویسی است و تسط به آن کار را حتی نیست. به این نکته هم توجه کنید که بازگشت باید در یک نقطه متوقف شود وگرنه برای بی نهایت بار، متد، خود را فراخوانی می کند.

در این درس یک مثال ساده از بازگشت را برای شما توضیح می دهیم.

فاکتوریل یک عدد صحیح مثبت (n!) شامل حاصل ضرب همه اعداد مثبت صحیح کوچکتر یا مساوی آن می باشد.

به فاکتوریل عدد 5 توجه کنید.

```
5! = 5 * 4 * 3 * 2 * 1 = 120
```

بنابراین برای ساخت یک متد بازگشتی باید به فکر توقف آن هم باشیم.بر اساس توضیح بازگشت ، فاکتوریل فقط برای اعداد مثبت صحیح است. کوچکترین عدد صحیح مثبت 1 است. در نتیجه از این مقدار برای متوقف کردن بازگشت استفاده می کنیم.

```
1  using System;
2
3  public class Program
4  {
5      static long Factorial(int number)
6      {
7          if (number == 1)
8              return 1;
9
10         return number * Factorial(number - 1);
11     }
12
13     public static void Main()
14     {
15         Console.WriteLine(Factorial(5));
16     }
17 }
```

نتیجه

```
120
```

متد مقدار بزرگی را بر می گرداند چون محاسبه فاکتوریل می تواند خیلی بزرگ باشد.

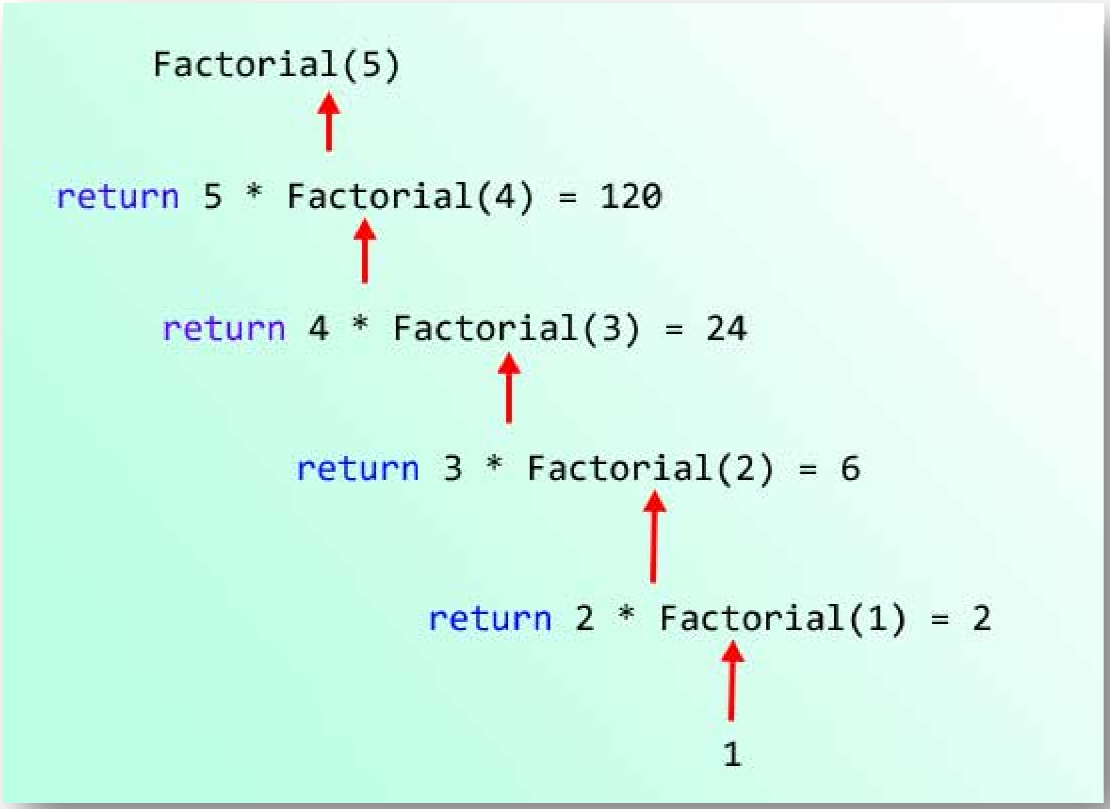
متد یک آرگومان که یک عدد است و می تواند در محاسبه مورد استفاده قرار گیرد را می پذیرد.

در داخل متد یک دستور if می نویسیم و در خط 7 می گوییم که اگر آرگومان ارسال شده برابر 1 باشد سپس مقدار 1 را برگردان در غیر اینصورت به خط بعد برو. این شرط باعث توقف تکرارها نیز می شود.

در خط 10 مقدار جاری متغیر **number** در عددی یک واحد کمتر از خودش **(number – 1)** ضرب می شود.در این خط متد **Factorial** خود را فراخوانی می کند و آرگومان آن در این خط همان **number – 1** است. مثلا اگر مقدار جاری **number** 10 باشد یعنی اگر ما بخواهیم فاکتوریل عدد 10 را به دست بیاوریم آرگومان متد **Factorial** در اولین ضرب **9** خواهد بود.

فرایند ضرب تا زمانی ادامه می یابد که آرگومان ارسال شده با عدد 1 برابر نشود.

شکل زیر فاکتوریل عدد **5** را نشان می دهد.



کد بالا را به وسیله یک حلقه **for** نیز می توان نوشت.

```
factorial = 1;

for ( int counter = number; counter >= 1; counter-- )
   factorial *= counter;
```

این کد از کد معادل بازگشتی آن آسان تر است. از بازگشت در زمینه های خاصی در علوم کامپیوتر استفاده می شود.

استفاده از بازگشت زمانی طبیعی تر به نظر می رسد که ما از غیر بازگشتی **(Iteration)** استفاده کنیم.

استفاده از بازگشت حافظه زیادی اشغال می کند پس اگر سرعت برای شما مهم است از آن استفاده نکنید.

شمارش (Enumeration)

شمارش راهی برای تعریف داده هایی است که می توانند مقادیری محدودی که شما از قبل تعریف کرده اید را بپذیرند.

به عنوان مثال شما می خواهید یک متغیر تعریف کنید که فقط مقادیر جهت (جغرافیایی)مانند **east**، **west**، **north** و **south** را در خود ذخیره کند. ابتدا یک **enumeration** تعریف می کنید و برای آن یک اسم انتخاب کرده و بعد از آن تمام مقادیر ممکن که می توانند در داخل بدنه آن قرار بگیرند تعریف می کنید.

به نحوه تعریف یک **enumeration** توجه کنید:

```
enum enumName
{
   value1,
   value2,
   value3,
   .
   .
   .
   valueN
}
```

ابتدا کلمه کلیدی **enum** و سپس نام آن را به کار می بریم. در سی شارپ برای نامگذاری **enumeration** از روش پاسکال استفاده کنید. در بدنه **enum** مقادیری وجود دارند که برای هر کدام یک نام در نظر گرفته شده است.

به یک مثال توجه کنید :

```
enum Direction
{
    North,
    East,
    South,
    West
}
```

در حالت پیشفرض مقادیری که یک **enumeration** می تواند ذخیره کند از نوع **int** هستند. به عنوان مثال مقدار پیشفرض **North** صفر و مقدار بقیه مقادیر یک واحد بیشتر از مقدار قبلی خودشان است. بنابراین مقدار **East** برابر ۱، مقدار **South** برابر ۲ و مقدار **West** برابر ۳ است.

می توانید این مقادیر پیشفرض را به دلخواه تغییر دهید، مانند:

```
enum Direction
{
    North = 3,
    East = 5,
    South = 7,
    West = 9
}
```

اگر به عنوان مثال هیچ مقداری به یک عنصر اختصاص ندهید آن عنصر به صورت خودکار مقدار می کیرد.

```
enum Direction
{
    North = 3,
    East = 5,
    South,
    West
}
```

در مثال بالا مشاهده می کنید که ما هیچ مقداری به **South** در نظر نگرفته ایم بنابر این به صورت خودکار یک واحد بیشتر از **East**یعنی ۶ و به **West** یک واحد بیشتر از **South** یعنی ۷ اختصاص داده می شود. همچنین می توان مقادیر یکسانی برای عناصر **enumeration** در نظر گرفت.

مثال :

```
enum Direction
{
    North = 3,
    East,
    South = North,
    West
}
```

می توانید مقادیر بالا را حدس بزنید؟ مقادیر **North**، **East**، **South**، **West** به ترتیب ۳، ۴، ۳، ۴ است. وقتی مقدار ۳ را به **North** می دهیم مقدار **East** برابر ۴ می شود. سپس وقتی مقدار **South**را برابر ۳ قرار دهیم به صورت اتوماتیک مقدار **West** برابر ۴ می شود.

اگر نمی خواهید که مقادیر آیتم های **enumeration** شما پیشفرض (از نوع **int**) باشد می توانید از نوع مثلا **byte** به عنوان نوع داده ای آیتمهای آن اسفاده کنید.

```
enum Direction : byte
{
    North,
    East,
    South,
    West
}
```

نوع داده ای **byte** فقط شامل مقادیر بین ۰ تا ۲۵۵ می شود بنابر این تعداد مقادیر که شما می توانید به **enumeration** اضافه کنید محدود می باشد.

به نحوه استفاده از **enumeration** در یک برنامه سی شارپ توجه کنید.

```
1   using System;
2
3   enum Direction
4   {
5       North = 1,
6       East,
7       South,
8       West
9   }
10
11  public class Program
12  {
13      public static void Main()
14      {
15          Direction myDirection;
16
17          myDirection = Direction.North;
18
19          Console.WriteLine("Direction: {0}", myDirection.ToString());
20      }
21  }
```

نتیجه

```
Direction: North
```

ابتدا **enumeration** را در خطوط ۹-۳ تعریف می کنیم. توجه کنید که **enumeration** را خارج از کلاس قرار داده ایم. این کار باعث می شود که **enumeration** در سراسر برنامه در دسترس باشد. میتوان **enumeration** را در داخل کلاس هم تعریف کرد ولی در این صورت فقط در داخل کلاس قابل دسترس است.

```
class Program
{
    enum Direction
    {
        //Code omitted
    }

    static void Main(string[] args)
    {
        //Code omitted
    }
}
```

برنامه را ادامه می دهیم. در داخل بدنه **enumeration** نام چهار جهت جغرافیایی وجود دارد که هر یک از آنها با ۱ تا ۴ مقدار دهی شده اند.

در خط ۱۵ یک متغیر تعریف شده است که مقدار یک جهت را در خود ذخیره می کند.

نحوه تعریف آن به صورت زیر است :

```
enumType variableName ;
```

در اینجا **enumType** نوع داده شمارشی (مثلا **Direction** یا مسیر) می باشد و**variableName** نیز نامی است که برای آن انتخاب کرده ایم که در مثال قبل **myDirection** است.

سپس یک مقدار به متغیر **myDirection** اختصاص می دهیم (خط ۱۷).

برای اختصاص یک مقدار به صورت زیر عمل می کنیم :

```
variable = enumType.value;
```

ابتدا نوع **Enumeration** سپس علامت نقطه و بعد مقدار آن (مثلا **North**) را می نویسیم.

می توان یک متغیر را فورا ،به روش زیر مقدار دهی کرد :

```
Direction myDirection = Direction.North;
```

حال در خط ۱۹ با استفاده از **Console.WriteLine()** مقدار **myDirection** را چاپ می کنیم.

توجه کنید که با استفاده از متد **ToString()** مقدار عددی **myDirection** را به رشته ، جهت چاپ تبدیل می کنیم.

تصور کنید که اگر **enumeration** نبود شما مجبور بودید که به جای کلمات اعداد را حفظ کنید چون مقادیر **enumeration** در واقع اعدادی هستند که با نام مستعار توسط شما یا هر کس دیگر تعریف می شوند.

متغیرهای شمارشی می توانند به انواع دیگری مانند **int** یا **string** تبدیل شوند. همچنین یک مقدار رشته ای می تواند به نوع شمارشی معادلش تبدیل شود.

تبدیل انواع شمارشی

می توان انواع شمارشی را به دیگر مقادیر تبدیل کرد و بالعکس.

مقادیر شمارشی در واقع مقادیر عددی هستند که برای درک بهتر آنها، به هر عدد یک نام اختصاص داده شده است.

به مثال زیر توجه کنید :

```
1   using System;
2
3   enum Direction
4   {
5       North,
6       East,
7       South,
8       West
9   }
10  public class Program
11  {
12      public static void Main()
13      {
14          Direction myDirection = Direction.East;
15          int myDirectionCode = (int)myDirection;
16
17          Console.WriteLine("Value of East is {0}", myDirectionCode);
18
19          myDirection = (Direction)3;
20          Console.WriteLine("\nDirection: {0}", myDirection.ToString());
21      }
22  }
```

نتیجه

```
Value of East is 1

Direction: West
```

در خط ۱۴ متغیر **myDirection** را به مقدار **East** نوع شمارشی **Direction** اختصاص داده ایم.

در حالت پیشفرض مقدار **East** در داخل آیتمهای این داده شمارشی ۱ می باشد.

در خط ۱۵ نشان نحوه تبدیل یک آیتم از نوع شمارشی به عدد صحیح معادل آن به روش تبدیل صریح نشان داده شده است.

نحوه این تبدیل به صورت زیر است:

```
variable = (DestinationDataType)enumerationVariable;
```

از آنجاییکه متغیر **myDirectionCode** (خط ۱۵) از نوع **int** است در نتیجه یک مقدار **int** باید در آن قرار بگیرد.

می توان به سادگی نوع داده مقصد را در داخل یک جفت پرانتز قرار داد و آن را کنار نوع شمارشی بگذارید (خط ۱۵).

نتیجه یک مقدار تبدیل شده را برگشت می دهد.

در خط ۱۹ معکوس این کار را انجام می دهیم.در این خط یک مقدار صحیح را به یک مقدار شمارشی تبدیل می کنیم.

مقدار ۳ را برابر آیتم **West** قرار می دهیم.

برای تبدیل آن از روشی شبیه به تبدیل یک نوع شمارشی به صحیح استفاده می کنیم(تبدیل صریح).

به این نکته توجه کنید که اگر عددی را که می خواهید تبدیل کنید در محدوده انواع شمارشی نباشد، تبدیل انجام می شود ولی آن آیتم شمارشی و عدد برابر هم نیستند.

به عنوان مثال :

```
myDirection = (Direction)10;

Console.WriteLine("Direction: {0}", myDirection.ToString());
Direction: 10
```

از آنجاییکه عدد ۱۰ مقدار هیچ کدام از آیتمهای نوع شمارشی مثال بالا نیست (مقدار آیتمهای نوع شمارشی مثال بالا به ترتیب ۰ و ۱ و ۲ و ۳ می باشد)خروجی **Console** خود عدد را نشان می دهد ولی اگر به جای عدد ۱۰ هر کدام از مقادیر عددی ذکر شده را قرار دهید آیتم معادل با آن نمایش داده خواهد شد.

- **تبدیل یک نوع رشته ای به یک نوع شمارشی**

می توان یک نوع رشته ای را به نوع شمارشی تبدیل کرد.

مثلا می خواهید رشته **"West"** را به نوع شمارشی **Direction.West** مثال بالا تبدیل کنید.

برای این کار باید از کلاس **Enum** و فضای نام **System** به صورت زیر استفاده کنید :

```
Direction myDirection = (Direction)Enum.Parse(typeof(Direction), "West");

Console.WriteLine("Direction: {0}", myDirection.ToString());
Direction: West
```

متد **Enum.Parse()** دارای دو پارامتر است.

اولین پارامتر نوع شمارشی است.

با استفاده از عملگر **typeof** نوع شمارشی را برگشت می دهیم.

دومین پارامتر ، رشته ای است که قرار است به نوع شمارشی تبدیل شود .

چون مقدار برگشتی از نوع شی (**object**)است بنابراین یک تبدیل مناسب نوع شمارشی لازم است.

با این جزییات الان می دانیم که چگونه یک رشته را به نوع شمارشی تبدیل کنیم.

```
enumType name = (enumType)Enum.Parse(typeof(enumType), string);
```

اگر رشته ای که به متد ارسال می کنید جز آیتمهای داده شمارشی نباشد با خطا مواجه می شوید.

ساختارها

ساختارها یا **struct** انواع داده ای هستند که توسط کاربر تعریف می شوند(**user-define**) و می توانند دارای فیلد و متد باشند. با ساختارها می توان نوع داده ای خیلی سفارشی ایجاد کرد.
فرض کنید می خواهیم داده ای ایجاد کنیم که نه تنها نام شخص را ذخیره کند بلکه سن و حقوق ماهیانه او را نیز در خود جای دهد.

برای تعریف یک ساختار به صورت زیر عمل می کنیم :

```
struct StructName
{
   member1;
   member2;
   member3;
   ...
   member4;
}
```

برای تعریف ساختار از کلمه کلیدی **struct** استفاده می شود. برای نامگذاری ساختارها از روش نامگذاری **Pascal** استفاده می شود.
اعضا در مثال بالا (**member1-5**) می توانند متغیر باشند یا متد.

در زیر مثالی از یک ساختار آمده است :

```
1   using System;
2
3   public struct Employee
4   {
5       public string name;
6       public int age;
7       public decimal salary;
8   }
9
10  public class Program
11  {
12      public static void Main()
13      {
14          Employee employee1;
15          Employee employee2;
16
17          employee1.name = "Jack";
18          employee1.age = 21;
19          employee1.salary = 1000;
20
21          employee2.name = "Mark";
22          employee2.age = 23;
23          employee2.salary = 800;
24
25          Console.WriteLine("Employee 1 Details");
26          Console.WriteLine("Name: {0}", employee1.name);
27          Console.WriteLine("Age: {0}", employee1.age);
28          Console.WriteLine("Salary: {0:C}", employee1.salary);
29
30          Console.WriteLine(); //Seperator
31
32          Console.WriteLine("Employee 2 Details");
33          Console.WriteLine("Name: {0}", employee2.name);
34          Console.WriteLine("Age: {0}", employee2.age);
```

```
35          Console.WriteLine("Salary: {0:C}", employee2.salary);
36      }
37  }
```

نتیجه

```
Employee 1 Details
Name: Jack
Age: 21
Salary: $1000.00

Employee 2 Datails
Name: Mike
Age: 23
Salary: $800.00
```

برای درک بهتر، کد بالا را شرح می دهیم :

در خطوط ۸-۳ یک ساختار تعریف شده است. به کلمه **Public** در هنگام تعریف توجه کنید. این کلمه کلیدی نشان می دهد که **Employee** در هر جای برنامه قابل دسترسی و استفاده باشد و حتی خارج از برنامه.

Public یکی از سطوح دسترسی است که توضیحات بیشتر در مورد آن در درسهای آینده آمده است.

قبل از نام ساختار از کلمه کلیدی **struct** استفاده می کنیم.

نام ساختار نیز از روش نامگذاری **Pascal** پیروی می کند.

در داخل بدنه ساختار سه فیلد تعریف کرده ایم.

این سه فیلد مشخصات **Employee** (کارمند) مان را نشان می دهند.

مثلا یک کارمند دارای نام، سن و حقوق ماهانه می باشد. همچنین هر سه فیلد به صورت **Public** تعریف شده اند بنابراین در خارج از ساختار نیز می توان آنها را فراخوانی کرد.

در خطوط ۱۴ و ۱۵ دو نمونه از کلاس **Employee** تعریف شده است.

تعریف یک نمونه از ساختارها بسیار شبیه به تعریف یک متغیر معمولی است.

ابتدا نوع ساختار و سپس نام آن را مشخص می کنید.

در خطوط ۱۷ تا ۲۳ به فیلدهای مربوط به هر **employee** مقادیری اختصاص می دهید.

برای دسترسی به فیلدها در خارج از ساختار باید آنها را به صورت **Public** تعریف کنید.

ابتدا نام متغیر را تایپ کرده و سپس علامت دات (.) و در آخر نام فیلد را می نویسیم.

وقتی که از عملگر دات استفاده می کنیم این عملگر اجازه دسترسی به اعضای مخصوص آن ساختار یا کلاس را به شما می دهد.

در خطوط ۲۵ تا ۳۵ نشان داده شده که شما چطور می توانید به مقادیر ذخیره شده در هر فیلد ساختار دسترسی یابید.

ساختارها انواع مقداری هستند.

این بدین معنی است که اگر مثلا در مثال بالا **employee2** را برابر **employee1** قرار دهید، **employee2** همه مقادیر صفات **employee1** را به جای اینکه به آنها مراجعه کند ،کپی برداری می کند .

کلاس یک ساختار ساده است ولی از انواع مرجع به حساب می آید.

در مورد کلاس در درسهای آینده توضیح خواهیم داد.

می توان به ساختار ، متد هم اضافه کرد.

مثال زیر اصلاح شده مثال قبل است.

```
using System;

public struct Employee
{
    public string name;
    public int age;
    public decimal salary;

    public void SayThanks()
    {
        Console.WriteLine("{0} thanked you!", name);
    }
}

public class Program
{
    public static void Main()
    {
        Employee employee1;
        Employee employee2;

        employee1.name = "Jack";
        employee1.age = 21;
        employee1.salary = 1000;

        employee2.name = "Mark";
        employee2.age = 23;
        employee2.salary = 800;

        Console.WriteLine("Employee 1 Details");
        Console.WriteLine("Name: {0}", employee1.name);
        Console.WriteLine("Age: {0}", employee1.age);
        Console.WriteLine("Salary: {0:C}", employee1.salary);

        employee1.SayThanks();

        Console.WriteLine(); //Seperator

        Console.WriteLine("Employee 2 Details");
        Console.WriteLine("Name: {0}", employee2.name);
        Console.WriteLine("Age: {0}", employee2.age);
        Console.WriteLine("Salary: {0:C}", employee2.salary);

        employee2.SayThanks();
    }
}
```

نتیجه

```
Employee 1 Details
Name: Jack
Age: 21
Salary: $1000.00
Jack thanked you!

Employee 2 Details
Name: Mike
Age: 23
Salary: $800.00
Mike thanked you!
```

در خطوط ۹ تا ۱۲ یک متد در داخل ساختار تعریف شده است.

این متد یک پیام را در صفحه نمایش نشان می دهد و مقدار فیلد name را گرفته و یک پیام منحصر به فرد برای هر نمونه نشان می دهد.

برای فراخوانی متد، به جای اینکه بعد از علامت دات نام فیلد را بنویسیم، نام متد را نوشته و بعد از آن همانطور که در مثال بالا مشاهده می کنید (خطوط ۳۵ و ۴۴) پرانتزها را قرار می دهیم و در صورتی که متد به آرگومان هم نیاز داشت در داخل پرانتز آنها را می نویسیم.

برنامه نویسی شیء گرا (Object Oriented Programming)

برنامه نویسی شی گرا (OOP) شامل تعریف کلاسها و ساخت اشیاء مانند ساخت اشیاء در دنیای واقعی است.

برای مثال یک ماشین را در نظر بگیرید.این ماشین دارای خواصی مانند رنگ ، سرعت ، مدل ، سازنده و برخی خواص دیگر است. همچنین دارای رفتارها و حرکاتی مانند شتاب و پیچش به چپ و راست و ترمز است.

اشیاء در سی شارپ تقلیدی از یک شی مانند ماشین در دنیای واقعی هستند.

برنامه نویسی شی گرا با استفاده از کدهای دسته بندی شده کلاسها و اشیاء را بیشتر قابل کنترل می کند.

در ابتدا ما نیاز به تعریف یک کلاس برای ایجاد اشیاء مان داریم.

شیء در برنامه نویسی شی گراء از روی کلاسی که شما تعریف کرده اید ایجاد می شود.

برای مثال نقشه ساختمان شما یک کلاس است که ساختمان از روی آن ساخته شده است.

کلاس شامل خواص یک ساختمان مانند مساحت،بلندی و مواد مورد استفاده در ساخت خانه می باشد.

در دنیای واقعی ساختمان ها نیز بر اساس یک نقشه (کلاس) پایه گذاری(تعریف)شده اند.

برنامه نویسی شیء گرا یک روش جدید در برنامه نویسی است که بوسیله برنامه نویسان مورد استفاده قرار می گیرد و به آنها کمک می کند که برنامه هایی با قابلیت استفاده مجدد، خوانا و راحت طراحی کنند.

سی شارپ نیز یک برنامه شی گراست و هر چیز در سی شارپ یک شی است.

در درس زیر به شما نحوه تعریف کلاس و استفاده از اشیاء آموزش داده خواهد شد.

همچنین شما با دو مفهوم وراثت و چند ریختی که از مباحث مهم در برنامه نویسی شیء گرا هستند در آینده آشنا می شوید.

تعریف یک کلاس

کلاس به شما اجازه می دهد یک نوع داده ای که توسط کاربر تعریف می شود و شامل فیلدها و خواص (properties) و متدها است را ایجاد کنید.

کلاس در حکم یک نقشه برای یک شی می باشد.

شی یک چیز واقعی است که از ساختار، خواص و یا رفتارهای کلاس پیروی می کند.

وقتی یک شی می سازید یعنی اینکه یک نمونه از کلاس ساخته اید (در درس ممکن است از کلمات شی و نمونه به جای هم استفاده شود).

ابتدا ممکن است فکر کنید که کلاس ها و ساختارها شبیه هم هستند.

تفاوت مهم بین این دو این است که کلاسها از نوع مرجع و ساختارها از نوع داده ای هستند.

در درسهای آینده این موضوع شرح داده خواهد شد.

اگر یادتان باشد در بخشهای اولیه این آموزش کلاسی به نام Program تعریف کردیم که شامل متد Main() بود و ذکر شد که این متد نقطه آغاز هر برنامه است.

تعریف یک کلاس مشابه تعریف یک ساختار است.

اما به جای استفاده از کلمه کلیدی struct باید از کلمه کلیدی class استفاده شود.

```
class ClassName
{
    field1;
    field2;
    ...
    fieldN;

    method1;
    method2;
    ...
    methodN;
}
```

این کلمه کلیدی را قبل از نامی که برای کلاسمان انتخاب می کنیم می نویسیم.

در نامگذاری کلاسها هم از روش نامگذاری Pascal استفاده می کنیم.

در بدنه کلاس فیلدها و متدهای آن قرار داده می شوند.

فیلدها اعضای داده ای خصوصی هستند که کلاس از آنها برای رفتارها و ذخیره مقادیر خاصیت هایش (property) استفاده می کند.

متدها رفتارها یا کارهایی هستند که یک کلاس می تواند انجام دهد.

در زیر نحوه تعریف و استفاده از یک کلاس ساده به نام **person** نشان داده شده است.

```
using System;

public class Person
{
    public string name;
    public int age;
    public double height;

    public void TellInformation()
    {
        Console.WriteLine("Name: {0}", name);
        Console.WriteLine("Age: {0} years old", age);
        Console.WriteLine("Height: {0}cm", height);
    }
}

public class Program
{
    public static void Main()
    {
        Person firstPerson = new Person();
        Person secondPerson = new Person();

        firstPerson.name = "Jack";
        firstPerson.age = 21;
        firstPerson.height = 160;
        firstPerson.TellInformation();

        Console.WriteLine(); //Separator

        secondPerson.name = "Mike";
        secondPerson.age = 23;
        secondPerson.height = 158;
        secondPerson.TellInformation();
    }
}
```

نتیجه :

```
Name: Jack
Age: 21 years old
Height: 160cm

Name: Mike
Age: 23 years old
Height: 158cm
```

برنامه بالا شامل دو کلاس **Person** و **Program** می باشد.

می دانیم که کلاس **Program** شامل متد **()Main** است که برنامه برای اجرا به آن احتیاج دارد ولی اجازه دهید که بر روی کلاس **Person** تمرکز کنیم.

در خطوط **3-15** کلاس **Person** تعریف شده است.

در خط ۳ یک نام به کلاس اختصاص داده ایم تا به وسیله آن قابل دسترسی باشد.

همچنین سطح دسترسی آن را **public** تعریف کرده ایم تا در دیگر کلاسها قابل شناسایی باشد. درباره سطوح دسترسی در یک درس جداگانه بحث خواهیم کرد.

در داخل بدنه کلاس فیلدهای آن تعریف شده اند (خطوط ۷-۵)

این سه فیلد تعریف شده خصوصیات واقعی یک فرد در دنیای واقعی را در خود ذخیره می کنند. یک فرد در دنیای واقعی دارای نام، سن، و قد می باشد.

در خطوط ۱۴-۹ یک متد هم در داخل کلاس به نام **()TellInformation** تعریف شده است که رفتار کلاسمان است و مثلا اگر از فرد سوالی بپرسیم در مورد خودش چیزهایی می گوید.

در داخل متد کدهایی برای نشان دادن مقادیر موجود در فیلدها نوشته شده است.

نکته ای درباره فیلدها وجود دارد و این است که چون فیلدها در داخل کلاس تعریف و به عنوان اعضای کلاس در نظر گرفته شده اند محدوده آنها یک کلاس است.

این بدین معناست که فیلدها فقط می توانند در داخل کلاس یعنی جایی که به آن تعلق دارند و یا به وسیله نمونه ایجاد شده از کلاس مورد استفاده قرار بگیرند.

در داخل متد **()Main** ، خطوط ۲۲-۲۱ دو نمونه یا دو شی از کلاس **Person** ایجاد می کنیم. برای ایجاد یک نمونه از یک کلاس باید از کلمه کلیدی **new** و به دنبال آن نام کلاس و یک جفت پرانتز قرار دهیم. وقتی نمونه کلاس ایجاد شد، سازنده را صدا می زنیم.

یک سازنده متد خاصی است که برای مقداردهی اولیه به فیلدهای یک شی به کار می رود.

وقتی هیچ آرگومانی در داخل پرانتزها قرار ندهید ،کلاس یک سازنده پیشفرض بدون پارامتر را فراخوانی می کند.

درباره سازنده ها در درس های آینده توضیح خواهیم داد.

در خطوط ۲۶-۲۴ مقادیری به فیلدهای اولین شی ایجاد شده از کلاس **Person** (**first Person**) اختصاص داده شده است.

برای دسترسی به فیلدها یا متدهای یک شی از علامت نقطه (دات) استفاده می شود.

به عنوان مثال کد **firstPerson.name** نشان دهنده فیلد **name** از شی **firstPerson** می باشد.

برای چاپ مقادیر فیلدها باید متد **()TellInformation** شی **firstPerson** را فراخوانی می کنیم.

در خطوط ۳۴-۳۱ نیز مقادیری به شی دومی که قبلا از کلاس ایجاد شده تخصیص می دهیم و سپس متد **()TellInformation** را فراخوانی می کنیم.

به این نکته توجه کنید که firstPerson و secondPerson نسخه های متفاوتی از هر فیلد دارند بنابراین تعیین یک نام برای secondPerson هیچ تاثیری بر نام firstPerson ندارد.

در مورد اعضای کلاس در درسهای آینده توضیح خواهیم داد.

سازنده ها

سازنده ها متدهای خاصی هستند که وجود آنها برای ساخت اشیا لازم است.

آنها به شما اجازه می دهند که مقادیری را به هر یک از اعضای داده ای یک آرایه اختصاص دهید و کدهایی که را که می خواهید هنگام ایجاد یک شی اجرا شوند را به برنامه اضافه کنید.

اگر از هیچ سازنده ای در کلاس تان استفاده نکنید، کامپایلر از سازنده پیشفرض که یک سازنده بدون پارامتر است استفاده می کند.

می توانید در برنامه تان از تعداد زیادی سازنده استفاده کنید که دارای پارامترهای متفاوتی باشند.

در مثال زیر یک کلاس که شامل سازنده است را مشاهده می کنید.

```
using System;

namespace ConstructorsDemo
{
    public class Person
    {
        public string name;
        public int age;
        public double height;

        //Explicitly declare a default constructor
        public Person()
        {
        }

        //Constructor that has 3 parameters
        public Person(string n, int a, double h)
        {
            name = n;
            age = a;
            height = h;
        }

        public void ShowInformation()
        {
            Console.WriteLine("Name: {0}", name);
            Console.WriteLine("Age: {0} years old", age);
            Console.WriteLine("Height: {0}cm", height);
        }
    }

    public class Program
    {
        public static void Main()
        {
            Person firstPerson = new Person();
            Person secondPerson = new Person("Mike", 23, 158);

            firstPerson.name = "Jack";
            firstPerson.age = 21;
            firstPerson.height = 160;
            firstPerson.ShowInformation();

            Console.WriteLine(); //Seperator

            secondPerson.ShowInformation();
        }
    }
}
```

نتیجه

```
Name: Jack
Age: 21 years old
Height: 160cm

Name: Mike
Age: 23 years old
Height: 158cm
```

همانطور که مشاهده می کنید در مثال بالا دو سازنده را به کلاس **Person** اضافه کرده ایم.

یکی از آنها سازنده پیشفرض (خطوط ۱۲-۱۰) و دیگری سازنده ای است که سه آرگومان قبول می کند (خطوط ۲۰- ۱۵).

به این نکته توجه کنید که سازنده درست شبیه به یک متد است با این تفاوت که نه مقدار برگشتی دارد و نه از نوع **void** است.

نام سازنده باید دقیقا شبیه نام کلاس باشد.

سازنده پیشفرض در داخل بدنه اش هیچ چیزی ندارد.

سازنده پیشفرض وقتی فراخوانی می شود که ما از هیچ سازنده ای در کلاس مان استفاده نکنیم.

در آینده متوجه می شوید که چطور می توان مقادیر پیشفرضی به اعضای داده ای اختصاص داد، وقتی که از یک سازنده پیشفرض استفاده می کنید .

به دومین سازنده توجه کنید.

اولا که نام آن شبیه نام سازنده اول است.

سازنده ها نیز مانند متدها می توانند سربارگذاری شوند.

حال اجازه دهید که چطور می توانیم یک سازنده خاص را هنگام تعریف یک نمونه از کلاس فراخوانی کنیم.

```
Person firstPerson = new Person();
Person secondPerson = new Person("Mike", 23, 158);
```

در اولین نمونه ایجاد شده از کلاس **Person** از سازنده پیشفرض استفاده کرده ایم چون پارامتری برای دریافت آرگومان ندارد.

در دومین نمونه ایجاد شده، از سازنده ای استفاده می کنیم که دارای سه پارامتر است.

کد زیر تاثیر استفاده از دو سازنده مختلف را نشان می دهد :

```
firstPerson.name = "Jack";
firstPerson.age = 21;
firstPerson.height = 160;
firstPerson.ShowInformation();

Console.WriteLine(); //Seperator

secondPerson.ShowInformation();
```

همانطور که مشاهده می کنید لازم است که به فیلدهای شی ای که از سازنده پیشفرض استفاده می کند مقادیری اختصاص داده شود تا این شی نیز با فراخوانی متد **ShowInformation()** آنها را نمایش دهد.

حال به شی دوم که از سازنده دارای پارامتر استفاده می کند توجه کنید، مشاهده می کنید که با فراخوانی متد **ShowInformation()** همه چیز همانطور که انتظار می رود اجرا می شود.

این بدین دلیل است که شما هنگام تعریف نمونه و از قبل مقادیری به هر یک از فیلدها اختصاص داده اید بنابراین آنها نیاز به مقدار دهی مجدد ندارند مگر اینکه شما بخواهید این مقادیر را اصلاح کنید.

- **اختصاص مقادیر پیشفرض به سازنده پیشفرض**

در مثالهای قبلی یک سازنده پیشفرض با بدنه خالی نشان داده شد.

شما می توانید به بدنه این سازنده پیشفرض کدهایی اضافه کنید.

همچنین می توانید مقادیر پیشفرضی به فیلدهای آن اختصاص دهید.

```
public Person()
{
   name = "No Name";
   age = 0;
   height = 0;
}
```

همانطور که در مثال بالا می بینید سازنده پیشفرض ما چیزی برای اجرا دارد.

اگر نمونه ای ایجاد کنیم که از این سازنده پیشفرض استفاده کند ، نمونه ایجاد شده مقادیر پیشفرض سازنده پیشفرض را نشان می دهد.

```
Person person1 = new Person();

person1.ShowInformation();
```

```
Name: No Name
Age: 0 years old
Height: 0cm
```

استفاده از کلمه کلیدی this

راهی دیگر برای ایجاد مقادیر پیشفرض استفاده از کلمه کلیدی **this** است.

مثال زیر اصلاح شده مثال قبل است و نحوه استفاده از ۴ سازنده با تعداد پارامترهای مختلف را نشان می دهد.

```
using System;

public class Person
{
    public string name;
    public int age;
    public double height;

    public Person()
        : this("No Name", 0, 0)
    {
    }

    public Person(string n)
        : this(n, 0, 0)
    {
    }

    public Person(string n, int a)
        : this(n, a, 0)
    {
    }

    public Person(string n, int a, double h)
    {
        name = n;
        age = a;
        height = h;
    }

    public void ShowInformation()
    {
        Console.WriteLine("Name: {0}", name);
        Console.WriteLine("Age: {0} years old", age);
        Console.WriteLine("Height: {0}cm\n", height);
    }
}

public class Program
{
    public static void Main()
    {
        Person firstPerson = new Person();
        Person secondPerson = new Person("Jack");
        Person thirdPerson = new Person("Mike", 23);
        Person fourthPerson = new Person("Chris", 18, 152);

        firstPerson.ShowInformation();
        secondPerson.ShowInformation();
        thirdPerson.ShowInformation();
        fourthPerson.ShowInformation();
    }
}
```

نتیجه :

```
Name: No Name
Age: 0 years old
Height: 0cm

Name: Jack
Age: 0 years old
Height: 0cm

Name: Mike
Age: 23 years old
Height: 0cm

Name: Chris
Age: 18 years old
Height: 152cm
```

ما چهار سازنده بری اصلاح کلاسمان تعریف کرده ایم.

شما می توانید تعداد زیادی سازنده برای مواقع لزوم در کلاس داشته باشید.

اولین سازنده یک سازنده پیشفرض است.

دومین سازنده یک پارامتر از نوع رشته دریافت می کند.

سومین سازنده دو پارامتر و چهارمین سازنده سه پارامتر می گیرد.

به چارمین سازنده در خطوط ۲۹-۲۴ توجه کنید.

سه سازنده دیگر به این سازنده وابسته هستند.

در خطوط ۱۲-۹ یک سازنده پیشفرض بدون پارامتر تعریف شده است.

توجه کنید که از کلمه کلیدی **this** باید بعد از علامت کالن (:) استفاده کنید.

این کلمه کلیدی به شما اجازه می دهد که یک سازنده دیگر موجود در داخل کلاس را فراخوانی کنید.

مقادیر پیشفرضی به فیلدها از طریق سازنده پیشفرض اختصاص می دهیم.

چون ما سه آرگومان برای پرانتزهای بعد از کلمه کلیدی **this** سازنده پیشفرض در نظر گرفته ایم در نتیجه سازنده ای که دارای سه پارامتر است فراخوانی شده و سه آرگومان به پارامترهای آن ارسال می شود.

کدهای داخل بدنه چهارمین سازنده اجرا می شوند و مقادیر پارامترهای آن به هر یک از اعضای داده ای مربوط اختصاص داده می شود.

نمی توانیم هیچ کدی را در داخل بدنه اولین سازنده بنویسیم چون توسط سازنده چهارم اجرا می شوند.

اگر کدی در داخل بدنه سازنده پیشفرض بنویسیم قبل از بقیه کدها اجرا می شود.

دومین سازنده (خطوط ۱۷-۱۴) به یک آرگومان نیاز دارد که همان فیلد **name** کلاس **Person** است.

وقتی این پارامتر با یک مقدار رشته ای پر شد، سپس به پارامترهای سازنده چهارم ارسال شده و در کنار دو مقدار پیشفرض دیگر (۰ برای **age** و ۰ برای **height**) قرار می گیرد. پس در داخل بدنه دومین سازنده هم نباید کدی بنویسیم چون این کد نیز توسط سازنده چهارم اجرا می شود.

در خط ۲۲-**19** سومین سازنده تعریف شده است که بسیار شبیه دومین سازنده است با این تفاوت که دو پارامتر دارد.

مقدار دو پارامتر سومین سازنده به اضافه ی یک مقدار پیشفرض صفر برای سومین آرگومان ، به چهارمین سازنده با استفاده از کلمه کلیدی **this** ارسال می شود.

```
Person firstPerson = new Person();
Person secondPerson = new Person("Jack");
Person thirdPerson = new Person("Mike", 23);
Person fourthPerson = new Person("Chris", 18, 152);
```

همانطور که مشاهده می کنید با ایجاد چندین سازنده برای یک کلاس، چندین راه برای ایجاد یک شی بر اساس داده هایی که نیاز داریم به وجود می آید.

در مثال بالا ۴ نمونه از کلاس **Person** ایجاد کرده ایم و چهار تغییر در سازنده آن به وجود آورده ایم.

سپس مقادیر مربوط به فیلدهای هر نمونه را نمایش می دهیم.

یکی از موراد استفاده از کلمه کلیدی **this** به صورت زیر است.

فرض کنید نام پارامترهای متد کلاس شما یا سازنده، شبیه نام یکی از فیلدها باشد.

```
public Person(string name, int age, double height)
{
   name = name;
   age = age;
   height = height;
}
```

این نوع کدنویسی ابهام بر انگیز است و کامپایلر نمی تواند متغیر را تشخیص داده و مقداری به آن اختصاص دهد.

اینجاست که از کلمه کلیدی **this** استفاده می کنیم.

```
public Person(string name, int age, double height)
{
this.name = name;
this.age = age;
this.height = height;
}
```

قبل از هر فیلدی کلمه کلیدی **this** را می نویسیم و نشان می دهیم که این همان چیزی است که می خواهیم به آن مقداری اختصاص دهیم. کلمه کلیدی **this** ارجاع یک شی به خودش را نشان می دهد.

سطح دسترسی

سطح دسترسی مشخص می کند که متدها یک کلاس یا اعضای داده ای در چه جای برنامه قابل دسترسی هستند.

در این درس می خواهیم به سطح دسترسی **Private** و **Public** نگاهی بیندازیم.

سطح دسترسی **Public** زمانی مورد استفاده قرار می گیرد که شما بخواهید به یک متد یا فیلد در خارج از کلاس و حتی پروژه دسترسی یابید.

به عنوان مثال به کد زیر توجه کنید :

```
1  using System;
2
3  public class Test
4  {
5      public int number;
6  }
7
8  public class Program
9  {
10     public static void Main()
11     {
12         Test x = new Test();
13
14         x.number = 10;
15     }
16 }
```

در این مثال کلاس **test** را به صورت **public** تعریف کرده ایم.
این کار به کلاس **program** اجازه می دهد که از کلاس **Test** نمونه ایجاد کند.
سپس یک عضو داده ای به صورت **public** در داخل کلاس **Test** تعریف می کنیم (خط ۵).
با تعریف این عضو به صورت **public** می توانیم آن را در خارج از کلاس **Test** و در داخل متد **()Main** کلاس **Program** مقدار دهی کنیم.
اگر از کلمه کلیدی **public** استفاده نکنیم نمی توانیم در داخل کلاس **program** نمونه ای از کلاس **Test** ایجاد کنیم و به اعضای آن دسترسی یابیم و این به نوعی به معنی استفاده از سطح دسترسی **private** می باشد.

```
1  using System;
2
3  private class Test
4  {
5      public int number;
6  }
7
8  public class Program
9  {
10     public static void Main()
11     {
12         Test x = new Test();
13
14         x.number = 10;
15     }
16 }
```

همانطور که در مثال بالا مشاهده می کنید این بار از کلمه **private** در کلاس **test** استفاده کرده ایم (خط ۳).

وقتی که برنامه را کامپایل می کنیم با خطا مواجه می شویم چون کلاس **Test** در داخل کلاس **Program** و یا هر کلاس دیگر قابل دسترسی نیست.

به این نکته توجه کنید که با اینکه عضو داده ای کلاس **Test** به صورت **Public** تعریف شده است باز هم نمی توان به آن در داخل کلاس **Program** دسترسی یافت چون کلاسی که در داخل آن قرار دارد از نوع **Private** است در نتیجه تمام اعضای مربوطه در خارج از آن غیر قابل دسترسی هستند.

نکته دیگر اینکه اگر شما برای یک کلاس سطح دسترسی تعریف نکنید آن کلاس دارای سطح دسترسی داخلی (**internal**) می شود به این معنی که فقط کلاس های داخل پروژه ای که با آن کار می کنید و می توانند به آن کلاس دسترسی یابند.

استفاده از سطح دسترسی **internal** معادل استفاده از سطح دسترسی **public** است با این تفاوت که در خارج از پروژه قابل دسترسی نیست.

کدهای زیر دارای تاثیر یکسانی هستند.

```
class Test
{

}
```

```
internal class Test
{

}
```

اگر یک کلاس را به صورت **public** و اعضای آن را به صورت **Private** تعریف کنیم ، آنگاه می توان یک نمونه از کلاس را در داخل کلاس های دیگر ایجاد کرد ولی اعضای آن قابل دسترسی نیستند.

اعضای داده ای **private** فقط به وسیله متد داخل کلاس **Test** قابل دسترسی هستند.

```
1   using System;
2
3   public class Test
4   {
5       private int number;
6   }
7
8   public class Program
9   {
10      public static void Main()
11      {
12          Test x = new Test();
13
14          x.number = 10; //Error, number is private
15      }
16  }
```

سطوح دسترسی دیگری هم در سی شارپ وجود دارد که بعد از مبحث وراثت در درسهای آینده در مورد آنها توضیح خواهیم داد.

کپسوله سازی

کپسوله کردن(تلفیق داده ها با یکدیگر) یا مخفی کردن اطلاعات فرایندی است که طی آن اطلاعات حساس یک موضوع از دید کاربر مخفی می شود و فقط اطلاعاتی که لازم باشد برای او نشان داده می شود.

وقتی که یک کلاس تعریف می کنیم معمولا تعدادی اعضای داده ای (فیلد) برای ذخیره مقادیر مربوط به شی نیز تعریف می کنیم. برخی از این اعضای داده ای توسط خود کلاس برای عملکرد متدها و برخی دیگر از آنها به عنوان یک متغیر موقت به کار می روند.

به این اعضای داده ای، اعضای مفید نیز می گویند چون فقط در عملکرد متدها تاثیر دارند و مانند یک داده قابل رویت کلاس نیستند.

لازم نیست که کاربر به تمام اعضای داده ای یا متدهای کلاس دسترسی داشته باشد.

اینکه فیلدها را طوری تعریف کنیم که در خارج از کلاس قابل دسترسی باشند بسیار خطرناک است چون ممکن است کاربر رفتار و نتیجه یک متد را تغییر دهد.

به برنامه ساده زیر توجه کنید :

```
1   using System;
2
3   public class Test
4   {
5       public int five = 5;
6
7       public int AddFive(int number)
8       {
9           number += five;
10          return number;
11      }
12  }
13
14  public class Program
15  {
16      public static void Main()
17      {
18          Test x = new Test();
19
20          x.five = 10;
21          Console.WriteLine(x.AddFive(100));
22      }
23  }
```

نتیجه

```
110
```

متد داخل کلاس **Test** به نام **AddFive** دارای هدف ساده ای است و آن اضافه کردن مقدار ۵ به هر عدد می باشد. در داخل متد **Main()** یک نمونه از کلاس **Test** ایجاد کرده ایم و مقدار فیلد آن را از ۵ به ۱۰ تغییر می دهیم. حال قابلیت متد **AddFive()** به خوبی تغییر می کند و شما نتیجه متفاوتی مشاهده می کنید. اینجاست که اهمیت کپسوله سازی مشخص می شود. اینکه ما در درسهای قبلی فیلدها را به صورت **public** تعریف کردیم و به کاربر اجازه دادیم که در خارج از کلاس به آنها دسترسی داشته باشد کار اشتباهی بود. فیلدها باید همیشه به صورت **private** تعریف شوند.

خواص

Property (خصوصیت) استانداردی در سی شارپ برای دسترسی به اعضای داده ای با سطح دسترسی private در داخل یک کلاس می باشد.

هر property دارای دو بخش می باشد، یک بخش جهت مقدار دهی (بلوک set) و یک بخش برای دسترسی به مقدار (بلوک get) یک داده private می باشد.

Property ها باید به صورت public تعریف شوند تا در کلاسهای دیگر نیز قابل دسترسی می باشند.

در مثال زیر نحوه تعریف و استفاده از **property** آمده است :

```
using System;

public class Person
{
    private string name;
    private int age;
    private double height;

    public string Name
    {
        get
        {
            return name;
        }
        set
        {
            name = value;
        }
    }

    public int Age
    {
        get
        {
            return age;
        }
        set
        {
            age = value;
        }
    }

    public double Height
    {
        get
        {
            return height;
        }
        set
        {
            height = value;
        }
    }

    public Person(string name, int age, double height)
    {
        this.name = name;
        this.age = age;
        this.height = height;
    }
```

```
51
52  }
53
54  public class Program
55  {
56      public static void Main()
57      {
58          Person person1 = new Person("Jack", 21, 160);
59          Person person2 = new Person("Mike", 23, 158);
60
61          Console.WriteLine("Name: {0}", person1.Name);
62          Console.WriteLine("Age: {0} years old", person1.Age);
63          Console.WriteLine("Height: {0}cm", person1.Height);
64
65          Console.WriteLine(); //Seperator
66
67          Console.WriteLine("Name: {0}", person2.Name);
68          Console.WriteLine("Age: {0} years old", person2.Age);
69          Console.WriteLine("Height: {0}cm", person2.Height);
70
71          person1.Name = "Frank";
72          person1.Age = 19;
73          person1.Height = 162;
74
75          person2.Name = "Ronald";
76          person2.Age = 25;
77          person2.Height = 174;
78
79          Console.WriteLine(); //Seperator
80
81          Console.WriteLine("Name: {0}", person1.Name);
82          Console.WriteLine("Age: {0} years old", person1.Age);
83          Console.WriteLine("Height: {0}cm", person1.Height);
84
85          Console.WriteLine();
86
87          Console.WriteLine("Name: {0}", person2.Name);
88          Console.WriteLine("Age: {0} years old", person2.Age);
89          Console.WriteLine("Height: {0}cm", person2.Height);
90      }
91  }
```

```
Name: Jack
Age: 21 years old
Height: 160cm

Name: Mike
Age: 23 years old
Height: 158cm

Name: Frank
Age: 19 years old
Height: 162cm

Name: Ronald
Age: 25 years old
Height: 174cm
```

در برنامه بالا نحوه استفاده از **property** آمده است.همانطور که مشاهده می کنید در این برنامه ما سه خصوصیت که هر کدام مربوط به اعضای داده ای هستند تعریف کرده ایم (سه فیلد با سطح دسترسی **private**).

```
private string name;
private int age;
private double height;
```

دسترسی به مقادیر این فیلدها فقط از طریق **property** های ارائه شده امکان پذیر است.

```
9         public string Name
10        {
11            get
12            {
13                return name;
14            }
15            set
16            {
17                name = value;
18            }
19        }
20
21        public int Age
22        {
23            get
24            {
25                return age;
26            }
27            set
28            {
29                age = value;
30            }
31        }
32
33        public double Height
34        {
35            get
36            {
37                return height;
38            }
39            set
40            {
41                height = value;
42            }
43        }
```

وقتی یک خاصیت ایجاد می کنیم ، باید سطح دسترسی آن را **public** تعریف کرده و نوع داده ای را که بر می گرداند یا قبول می کند را مشخص کنیم.

به این نکته توجه کنید که نام **property** ها همانند نام فیلدهای مربوطه می باشد با این تفاوت که حرف اول آنها بزرگ نوشته می شود.

البته یادآور می شویم که شباهت نام **property** ها و فیلدها اجبار نیست و یک قرارداد در سی شارپ می باشد.

در داخل بدنه دو بخش می بینید ، یکی بخش **set** و دیگری بخش **get**.

بخش **get** ، که با کلمه کلیدی **get** نشان داده شده است به شما اجازه می دهد که یک مقدار را از فیلدها (اعضای داده ای) استخراج کنید.

بخش **set** ، که با کلمه کلیدی **set** نشان داده شده است برای مقدار دهی به فیلدها (اعضای داده ای) به کار می رود.

به کلمه کلیدی **value** در داخل بلوک **set** توجه کنید. **Value** همان مقداری است که به **property** اختصاص می دهیم.

برای دسترسی به یک خاصیت می توانید از علامت دات (.) استفاده کنید.

```
Console.WriteLine("Name: {0}", person1.Name);
Console.WriteLine("Age: {0} years old", person1.Age);
Console.WriteLine("Height: {0}cm", person1.Height);
```

فراخوانی یک خاصیت باعث اجرای کد داخل بدنه بلوک **get** آن می شود.

سپس این بلوک مانند یک متد مقداری به فراخوان برگشت می دهد.

مقدار دهی به یک **property** بسیار آسان است.

```
person1.Name = "Frank";
person1.Age = 19;
person1.Height = 162;
```

دستورات بالا بخش **set** مربوط به هر **property** را فراخوانی کرده و مقادیری به هر یک از فیلدها اختصاص می دهد.
استفاده از **property** ها کد نویسی را انعطاف پذیر می کند مخصوصا اگر بخواهید یک اعتبارسنجی برای اختصاص یک مقدار به فیلدها یا استخراج یک مقدار از آنها ایجاد کنید.
مثلا شما می توانید یک محدودیت ایجاد کنید که فقط اعداد مثبت به فیلد **age** (سن) اختصاص داده شود.
می توانید با تغییر بخش **set** خاصیت **Age** این کار را انجام دهید :

```
1   public int Age
2   {
3       get
4       {
5           return age;
6       }
7       set
8       {
9           if (value > 0)
10          age = value;
11          else
12          age = 0;
13      }
14  }
```

حال اگر کاربر بخواهد یک مقدار منفی به فیلد **age** اختصاص دهد مقدار **age** صفر خواهد شد.

همچنین می توان یک **property** فقط خواندنی (**read-only**) ایجاد کرد. این **property** فاقد بخش **set** است.
به عنوان مثال می توان یک خاصیت **Name** فقط خواندنی مانند زیر ایجاد کرد :

```
public string Name
{
    get
    {
        return name;
    }
}
```

در این مورد اگر بخواهید یک مقدار جدید به فیلد **name** اختصاص دهید با خطا مواجه می شوید.

نکته دیگری که باید به آن توجه کنید این است که شما می توانید برای بخش **set** یا **get** سطح دسترسی ایجاد کنید.

به تکه کد زیر توجه کنید :

```
public string Name
{
    get
    {
        return name;
    }
    private set
    {
        name = value;
    }
}
```

خاصیت **Name** فقط در خارج از کلاس قابل خواندن است اما متدها فقط داخل کلاس **Person** می توانند مقادیر جدید بگیرند.

یک **property** می تواند دارای دو فیلد باشد. به کد زیر توجه کنید :

```
private string firstName;
private string lastName;

public FullName
{
   get { return firstName + " " + lastName; }
}
```

همانطور که در مثال بالا مشاهده می کنید یک **property** فقط خواندنی تعریف کرده ایم که مقدار برگشتی آن ترکیبی از دو فیلد **firstName** و **lastName** است که به وسیله فاصله از هم جدا شده اند.

سی شارپ همچنین یک راه حل کوتاه برای ایجاد **property** ارائه می دهد. در این روش می توانید یک **property** بدون فیلد ایجاد کنید.

```
public int MyProperty { get; set; }
```

این ویژگی فراخوانی خودکار property نام دارد و در سی شارپ ۳.۰ معرفی شده است.
به این نکته توجه کنید که در این روش هیچ کدی برای بخش set و get نمی نویسیم.
دستور بالا معادل تعریف یک فیلد از نوع int با سطح دسترسی private است و property مربوط در یک خط مختصر نوشته شده و اجرا می شود.
کامپایلر کد بالا را به صورت خودکار به عنوان یک property شناسایی می کند و فیلد مربوط به آن در طول زمان اجرای برنامه ساخته می شود.
توجه کنید وقتی یک Property خودکار ایجاد می کنید باید هر دو بخش get و set آن را نیز تعریف کنید.
همچنین نباید هیچ کدی در داخل این دو بخش بنویسید.

فضای نام

فضای نام راهی برای دسته بندی کدهای برنامه می باشد. هر چیز در دات نت حداقل در یک فضای نام قرار دارد.

وقتی برای یک کلاس اسمی انتخاب می کنید ممکن است برنامه نویسان دیگر به صورت اتفاقی اسمی شبیه به آن برای کلاسشان انتخاب کنند.

وقتی شما از آن کلاسها در برنامه تان استفاده کنید از آنجاییکه از کلاسهای همنام استفاده می کنید در برنامه ممکن است خطا به وجود آید.

فضاهای نامی از وقوع این خطاها جلوگیری کرده یا آنها را کاهش می دهند.

تا کنون و در درسهای قبلی ما فقط با یک فضای نام آشنا شده ایم و آن فضای نام System است که شامل تعداد زیادی کلاس و متد ، مانند کلاس Console و متد ()Writeline می باشد.

اما اگر یک پروژه جدید ایجاد کنید به صورت پیشفرض یک فضای نام برای شما ایجاد خواهد شد که نام آن شبیه به نام پروژه تان می باشد.

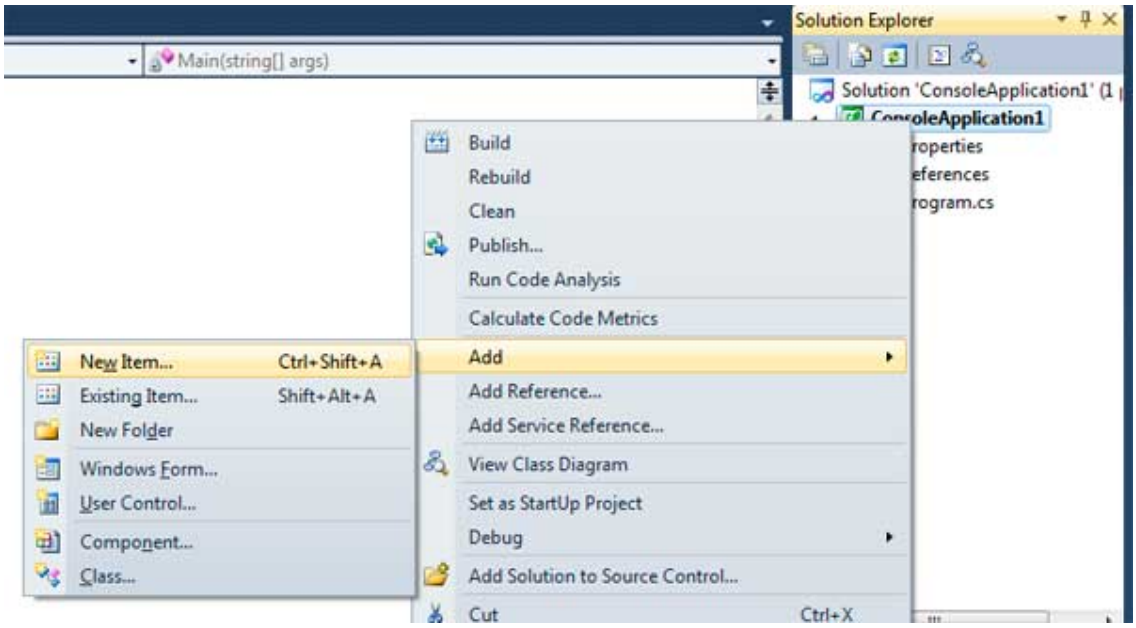
در این درس به شما نشان می دهیم که چگونه کلاسهایتان در در کدهای جداگانه بنویسید و سپس از آنها در فایلهای جدا استفاده کنید.

برنامه Visual Studio 2010 یا Visual C# 2010 را اجرا و یک پروژه جدید ایجاد کنید.

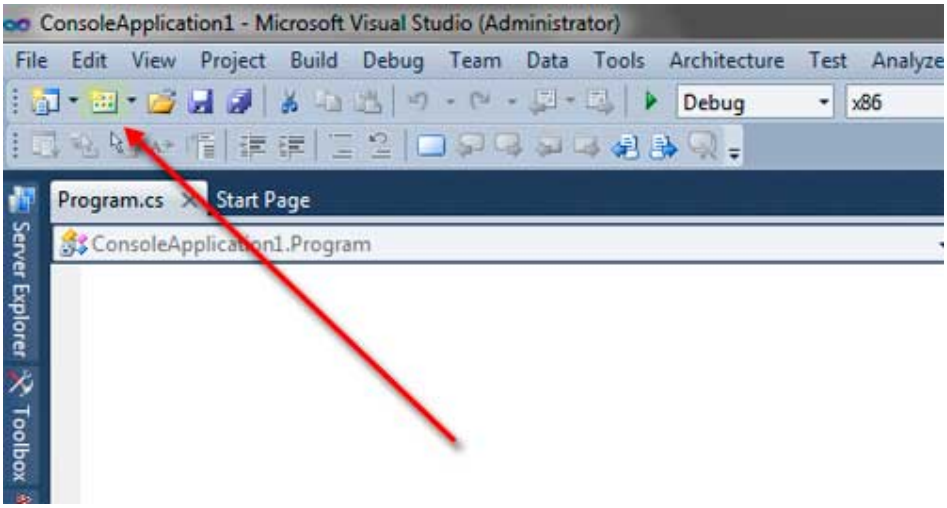
بعد از اینکه پروژه ایجاد شد ، یک فایل جدید ایجاد کنید.

چندین راه برای ایجاد یک فایل وجود دارد.

یکی از راه ها این است که بر روی پروژه تان در Solution Explorer راست کلیک کرده و سپس گزینه Add و بعد New Item را انتخاب کنید.

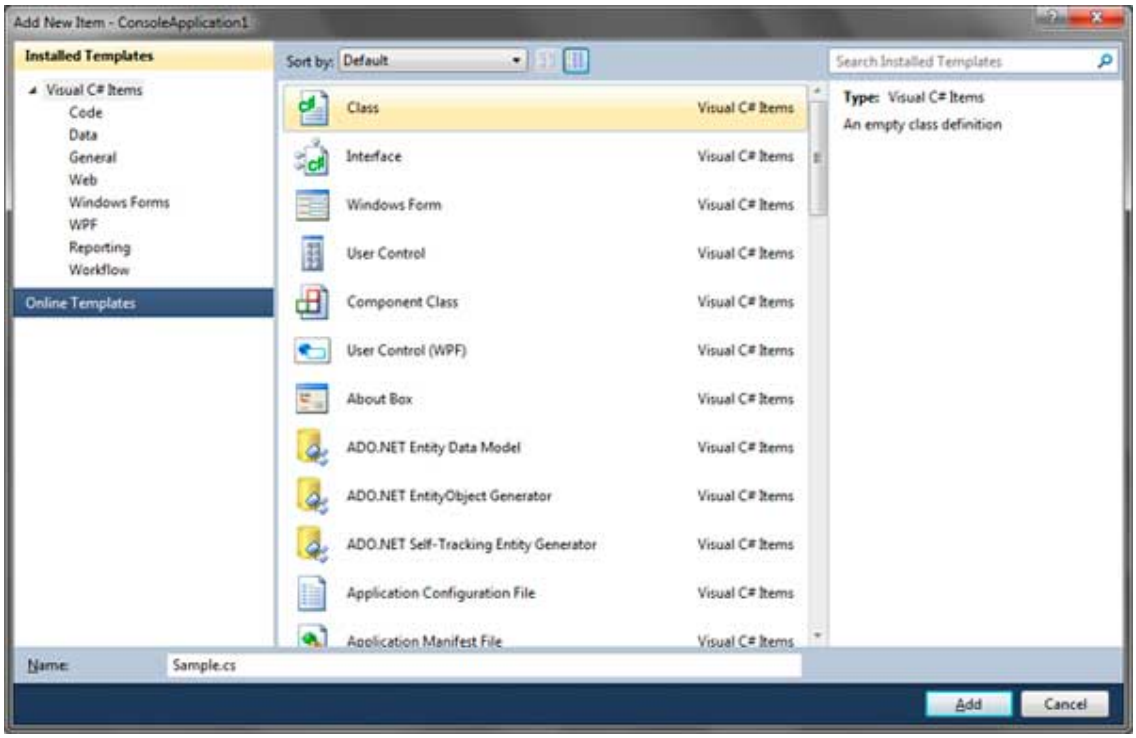


راه دیگر کلیک کردن بر روی دکمه **Add New Item** در **Toolbar** است :



همچنین می توان از دکمه تر کیبی **Ctrl + Shift + A** استفاده کرد.

هر کدام از این را ه ها را که انتخاب کنید در نهایت یک صفحه برای شما نشان داده می شود و از شما سوال می شود که چه فایلی را می خواهید ایجاد کنید :



گزینه **Class** را انتخاب کرده و نام آنرا **Sample.cs** بگذارید

کدهای سی شارپ دارای پسوند **cs** هستند.

برای درک منظور این درس همه کدهایی که در هنگام ایجاد کلاس به وجو می آیند را حذف کنید و کدهای زیر را وارد کنید :

```
namespace MyNamespace
{
    class Sample
    {
        public void ShowMessage()
        {
            System.Console.WriteLine("Hello World!");
        }
    }
}
```

همانطور که در کد بالا مشاهده می کنید فضای نام مان را تعریف کرده و نام آن را **MyNamespace** می گذاریم.

در داخل کلاس مان (**Sample**) یک متد برای نمایش پیغام وجود دارد.

به این نکته توجه کنید که با استفاده از فضای نام **System** ، به متد **()WriteLine** کلاس **Console** دسترسی یافته ایم.

می توانید با استفاده از کلمه کلیدی **using** فضای نام را از قبل تعریف کنید و آن را هنگام فراخوانی متد **()WriteLine** ننویسید (کد زیر).

حال به فایل **Program.cs** برنامه ای که قبلا ایجاد کردید بروید.

محتویات آنرا پاک کرده و کدهای زیر را می نویسید :

```
1   using MyNamespace;
2
3   class Program
4   {
5       static void Main()
6       {
7           Sample test = new Sample();
8
9           test.ShowMessage();
10      }
11  }
```

```
Hello World!
```

با استفاده از کلمه کلیدی **using** همه محتویات فضای نام **MyNamespace** را که قبلا ایجاد کردیم وارد برنامه جدید می کنیم(خط ۱).

اگر خط اول کد بالا را حذف کنیم، برای استفاده از هر چیز باید قبل از **Sample** از کلمه **MyNamespace** استفاده کنیم.

```
MyNamespace.Sample test = new MyNamespace.Sample();
```

می توان چندین کلاس یا رابط (**interface**) را به یک فضای نام اضافه کرد.

```
namespace MyNamespace
{
    class Sample1
    {

    }

    class Sample2
    {

    }
}
```

شما محدود به دسته بندی کدهای کلاستان در داخل یک فضای نام نیستید. می توانید یک فضای نام تو در تو ایجاد کنید و کدهایتان را در درون آن بنویسید :

```
namespace MyNamespace1
{
    namespace MyNamespace2
    {
        class Sample
        {
            public void ShowMessage()
            {
                System.Console.WriteLine("Hello World!");
            }
        }
    }
}
```

برای دسترسی به کلاس **Sample** ، مجبورید اول نام تمام فضاهای نامی را که کلاس **Sample** در آنها قرار دارد بنویسید.

```
MyNamespace1.MyNamespace2.Sample
```

یا می توان از کلمه کلیدی **using** استفاده کرد :

```
using MyNamespace1.MyNamespace2;
```

دات نت فریم ورک دارای فضاهای نام تو در تو می باشد.

به عنوان مثال **System.Data.SqlClient** سه فضای تو در تو می باشد.

می توان برای راحتی در کد نویسی فضاهای نامی تو در تو، یک فضای نامی مستعاری ایجاد کنید :

```
using AliasNamespace = MyNamespace1.MyNamespace2;
```

ساختارها در برابر کلاسها

تفاوت بین کلاس و ساختار چیست؟ ساختار ها انواع مقداری هستند مانند **int** ، **Double** و **String** .
وقتی یک مقدار از ساختار را در یک متغیر کپی می کنید، در اصل خود مقدار را کپی کرده اید نه آدرس یا مرجع آن را.
کلاسها انواع مرجع هستند مانند همه کلاسهای دات نت.
اجازه دهید تفاوت این دو را با یک مثال توضیح دهیم.

```
using System;

struct MyStructure
{
    public string Message { get; set; }
}

class MyClass
{
    public string Message { get; set; }
}

class Program
{
    static void Main()
    {
        MyStructure structure1 = new MyStructure();
        MyStructure structure2 = new MyStructure();

        structure1.Message = "ABC";
        structure2 = structure1;

        Console.WriteLine("Showing that structure1 " +
            "was copied to structure2.");
        Console.WriteLine("structure2.Message = {0}", structure2.Message);

        Console.WriteLine("\nModifying the value of structure2.Message...");
        structure2.Message = "123";

        Console.WriteLine("\nShowing that structure1 was not affected " +
            "by the modification of structure2");
        Console.WriteLine("structure1.Message = {0}", structure1.Message);


        MyClass class1 = new MyClass();
        MyClass class2 = new MyClass();

        class1.Message = "ABC";
        class2 = class1;

        Console.WriteLine("\n\nShowing that class1 " +
            "was copied to class2.");
        Console.WriteLine("class2.Message = {0}", class2.Message);

        Console.WriteLine("\nModifying the value of class2.Message...");
        class2.Message = "123";

        Console.WriteLine("\nShowing that class1 was also affected " +
            "by the modification of class2");
        Console.WriteLine("class1.Message = {0}", class1.Message);
    }
}
```

نتیجه

```
Showing that structure1 was copied to structure2.
structure2.Message = ABC

Modifying the value of structure2.Message...

Showing that structure1 was not affected by the modification of structure2
structure1.Message = ABC


Showing that class1 was copied to class2.
class2.Message = ABC

Modifying the value of class2.Message...

Showing that class1 was also affected by the modification of class2
class1.Message = 123
```

در بالا یک ساختار و یک کلاس ایجاد کرده ایم و تفاوت بین استفاده از این دو را نشان داده ایم.
یک خاصیت به نام **Message** برای هر دو قرار داده ایم.
سپس دو نمونه از هر کدام ایجاد کرده ایم (خطوط ۱۸-۱۷ و ۳۶-۳۵).
مقداری به خاصیت **Message** از نمونه اول ایجاد شده از ساختار (**structure1**) اختصاص می دهیم.
سپس مقدار **structure1** را برابر **structure2** قرار می دهیم، با این کار همه چیزهای داخل **structure1** در **structure2** کپی می شود.
برای ثابت کردن اینکه همه محتویات **structure1** کپی شده است، مقدار خاصیت **Message** ، **structure2** را نشان می دهیم و مشاهده می کنیم که همان مقدار خاصیت **Message**، **structure1** می باشد.
برای اثبات اینکه ساختارها انواع مقداری هستند یک پیغام دیگر را به خاصیت **Message**، **structure2** اختصاص می دهیم.
خاصیت **Message** ، **structure1** تحت تاثیر قرار نمی گیرد چون **structure2** یک کپی از **structure1** می باشد.
حال اثبات می کنیم که چرا کلاس ها انواع مرجع هستند.
کلاسها وقتی با یک متغیر برابر قرار داده می شوند آدرس خود را ارسال می کنند نه مقدارشان را.
بنابراین وقتی یک خاصیت از شیی که دارای آدرس شیء اصلی است را ویرایش می کنید خاصیت شیء اصلی نیز تغییر می کند.
وقتی یک شی را به عنوان آرگومان به متد ارسال می کنید، فقط آدرس شی ارسال می شود.
هر تغییری که در شی داخل متد به وجود بیاید بر شی اصلی که آدرس آن به متد ارسال شده است نیز تاثیر می گذارد.

کتابخانه کلاس

می توانید کتابخانه کلاسی ایجاد کنید که مجموعه ای از کلاسها و کدهایی است که می توانند کامپایل شوند و در نرم افزارهای دیگر برای استفاده مجدد به کار روند.

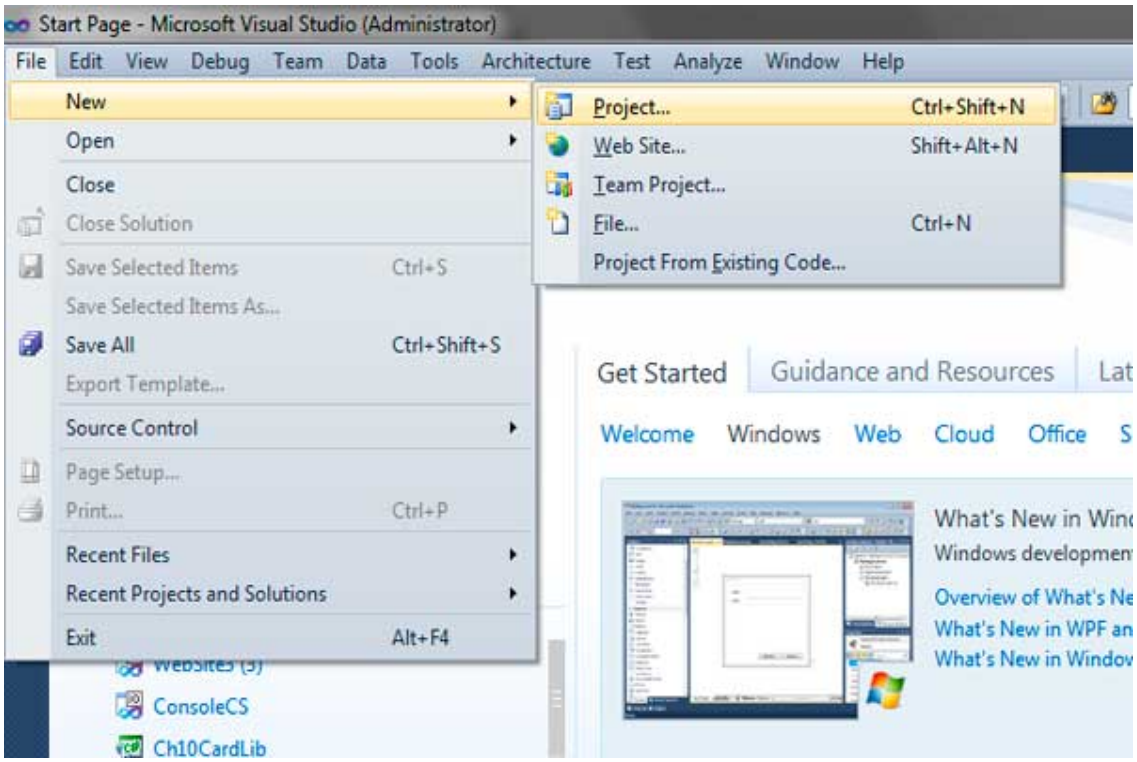
کتابخانه های کلاس به فایلهای DLL کامپایل می شوند.

DLL ها با هیچ برنامه ی خواندن فایل متنی قابل خواندن نیستند.

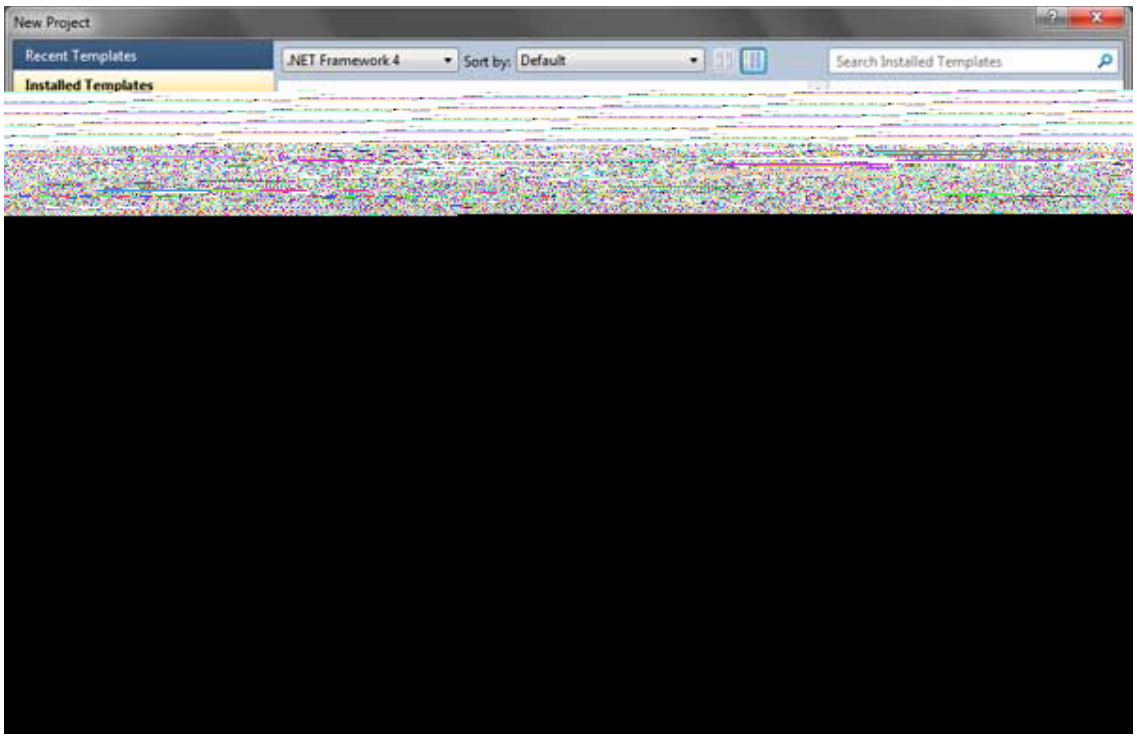
کتابخانه کلاس پروژه ای است که می توان از آن به عنوان یک راه حل شخصی برای حل مشکلات استفاده کرد.

حا ل یک کتابخانه کلاس به روش زیر ایجاد کرده و نام آن را Sample می گذاریم.

به مسیر Go to File > New > Project می رویم :



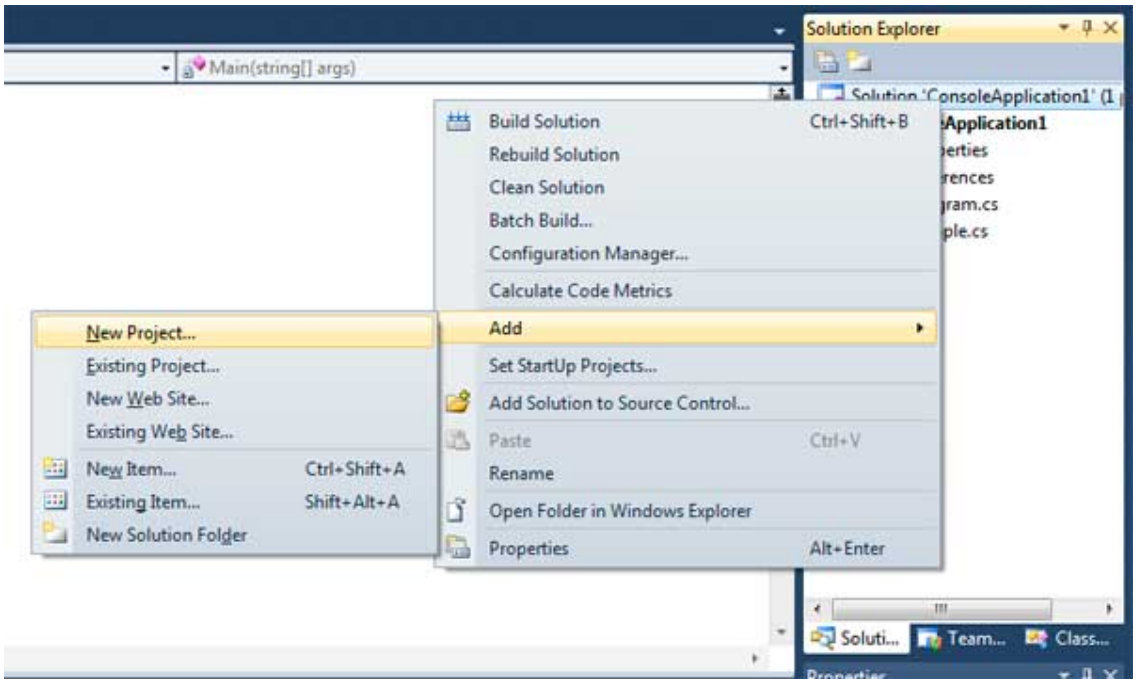
در لیست ظاهر شده گزینه **Class Library** را انتخاب می کنیم و همانطور که نشان داده شده است نام آن را **Sample** می گذاریم.



بعد از ایجاد کتابخانه کلاس، می توانید کدهایی که لازم دارید به آن اضافه کنید.

همچنین می توان یک کتابخانه کلاس را هنگامی که در حال کار بر روی یک پروژه هستید ایجاد کنید.

برای این کار به قسمت solution explorer پروژه تان رفته و روی solution کلیک راست کرده و مانند شکل زیر گزینه New Project را انتخاب می کنید.



بعد کدهایی که لازم دارید در داخل آن بنویسید و سپس برای ساخت نهایی آن از نوار Menu گزینه Build و در نهایت Build Solution را انتخاب می کنید.

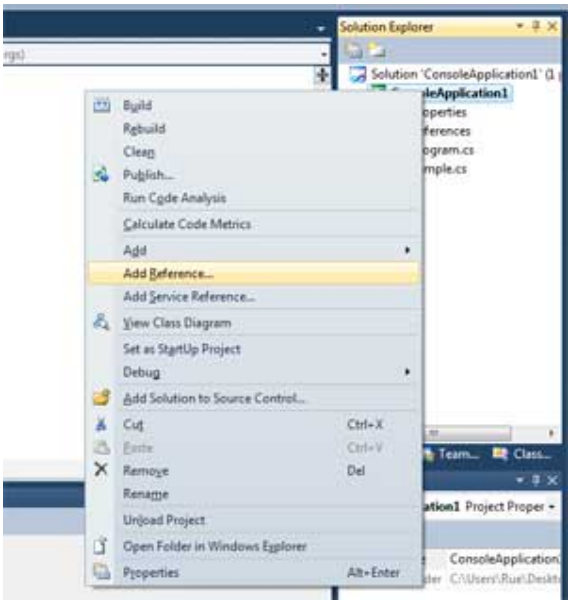
مشاهده خواهید کرد که تمام کدهایتان به یک فایل DLL تبدیل می شود.

حال می توانید از این فایل در سایر برنامه ها استفاده کنید.

یک برنامه کنسول دیگر ایجاد کنید.

بعد از ایجاد برنامه جدید، DLL ی را که از قبل ایجاد کرده اید به آن اضافه کنید.

برای اضافه کردن DLL بر روی پروژه راست کلیک کرده و مانند شکل زیر گزینه Add Reference را انتخاب می کنیم.



با کلیک بر روی این گزینه کادر زیر نمایش داده خواهد شد که سربرگ(**tab**) اول آن به شما اجازه اضافه کردن همه فایل های اسمبلی در دات نت را می دهد.

به این نکته توجه کنید که همه این فایلها در حالت پیشفرض وراد پروژه شما نمی شوند.

اگر پوشه **References** داخل **solution explorer** را باز کنید همه فایلهای اسمبلی که در دسترس شما قرار داده شده اند را مشاهده خواهید کرد.

اگر بخواهید از فایلهای اسمبلی بیشتری در دات نت استفاده کنید میتوانید آنها را از اولین سربرگ (سربرگ **.NET**) به پروژه اضافه کنید.

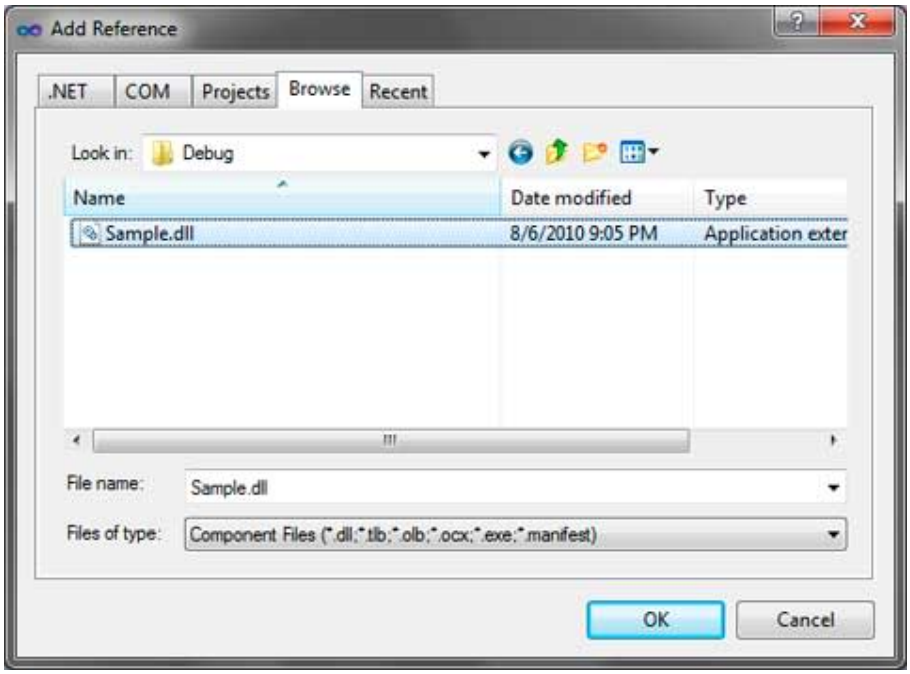
سربرگ **projects**، **solution** جاری را برای پروژه های کتابخانه کلاس اسکن کرده و بعد از یافتن، آنها را لیست می کند.

سومین سربرگ برای پیدا کردن فایلهای **DLL** ی که در خارج از **solution** جاری هستند به کار می رود.

به عنوان مثال اگر شما یک کتابخانه کلاس ساده در یک **solution** جداگانه ایجاد کنید، می توانید ان را وارد **solution** دیگر کرده و از آن استفاده کنید.

مکان فایل **DLL**. در داخل پوشه **Debug** یا **Release** پروژه کتابخانه کلاس می باشد.

این دو پوشه نیز به نوبه خود در داخل پوشه **bin** واقع در پوشه **Solution** کتابخانه کلاس قرار دارند.


Add Reference
.NET
COM
Projects
Browse
Recent
Look in:
Debug
Name
Date modified
Type
Sample.dll
8/6/2010 9:05 PM
Application exter
File name:
Sample.dll
Files of type:
Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)
OK
Cancel

وراثت

وراثت به یک کلاس اجازه می دهد که خصوصیات یا متدهایی را از کلاس دیگر به ارث برد.

وراثت مانند رابطه پدر و پسری می ماند به طوریکه فرزند خصوصیاتی از قبیل قیافه و رفتار را از پدر خود به ارث برده باشد.

کلاس پایه یا کلاس والد کلاسی است که بقیه کلاسها از آن ارث می برند.

کلاس مشتق یا کلاس فرزند کلاسی است که از کلاس پایه ارث بری می کند.

همه متد و خصوصیات کلاس پایه می توانند در کلاس مشتق مورد استفاده قرار بگیرند به استثنای اعضا و متدهای با سطح دسترسی **private** .

همه کلاسها در **.NET Framework** از کلاس **Object** ارث می برند.

مفهوم اصلی وراثت در مثال زیر نشان داده شده است :

```
1   using System;
2
3   class Parent
4   {
5       private string message;
6
7       public string Message
8       {
9           get { return message; }
10          set { message = value; }
11      }
12
13      public void ShowMessage()
14      {
15          Console.WriteLine(message);
16      }
17
18      public Parent(string message)
19      {
20          this.message = message;
21      }
22  }
23
24  class Child : Parent
25  {
26      public Child(string message)
27          : base(message)
28      {
29
30      }
31  }
```

در این مثال دو کلاس با نامهای **Parent** و **Child** تعریف شده است.

در این مثال یک عضو را یکبار با سطح دسترسی **private** (خط ۵)و یکبار با سطح دسترسی **public** (خط ۱۱-۷) تعریف کرده ایم.

سپس یک متد را برای نمایش پیام تعریف کرده ایم.

یک سازنده در کلاس **Parent** تعریف شده است که یک آرگومان از نوع رشته قبول می کند و یک پیغام نمایش می دهد.

حال به کلاس **Child** توجه کنید (خط ۲۴). این کلاس تمام متدها و خاصیت های کلاس **Parent** را به ارث برده است.

نحوه ارث بری یک کلاس به صورت زیر است :

```
class DerivedClass : BaseClass
```

براحتی می توان با قرار دادن یک کالن (:) بعد از نام کلاس و سپس نوشتن نام کلاسی که از آن ارث بری می شود (کلاس پایه) این کار را انجام داد.

در داخل کلاس **Child** هم یک سازنده ساده وجود دارد که یک آرگومان رشته ای قبول می کند.

وقتی از وراثت در کلاسها استفاده می کنیم، هم سازنده کلاس مشتق و هم سازنده پیشفرض کلاس پایه هر دو اجرا می شوند.

سازنده پیشفرض یک سازنده بدون پارامتر است.

اگر برای یک کلاس سازنده ای تعریف نکنیم کامپایلر به صورت خودکار یک سازنده برای آن ایجاد می کند.

اگر هنگام صدا زدن سازنده کلاس مشتق بخواهیم سازنده کلاس پایه را صدا بزنیم باید از کلمه کلیدی **base** استفاده کنیم.

کلمه کلیدی **base** یک سازنده از کلاس پایه را صدا می زند.

قبل از این کلمه کلیدی باید علامت کالن (**:**) را تایپ کنیم.

در مثال بالا به وسیله تامین مقدار پارامتر **message** سازنده کلاس مشتق و ارسال آن به داخل پرانتز کلمه کلیدی **base** ، سازنده معادل آن در کلاس پایه فراخوانی شده و مقدار **message** را به آن ارسال می کند.

سازنده کلاس **parent** هم این مقدار (مقدار **message**) را در یک عضو داده ای (فیلد) **Private** قرار می دهد.

می توانید کدهایی را به داخل بدنه سازنده **Child** اضافه کنید تا بعد از سازنده **Parent** اجرا شوند.

اگر از کلمه کلیدی **base** استفاده نشود به جای کلاس پایه سازنده پیشفرض فراخوانی می شود.

اجازه بدهید که اشیایی از کلاسهای **Parent** و **Child** بسازیم تا نشان دهیم که چگونه کلاس **Child** متدها و خواص کلاس **parent** را به ارث می برد.

```
1   class Program
2   {
3       static void Main()
4       {
5           Parent myParent = new Parent("Message from parent.");
6           Child myChild = new Child("Message from child.");
7
8           myParent.ShowMessage();
9
10          myChild.ShowMessage();
11
12          myParent.Message = "Modified message of the parent.";
13          myParent.ShowMessage();
14
15          myChild.Message = "Modified message of the child.";
16          myChild.ShowMessage();
17
18          //myChild.message; ERROR: can't access private members of base class
19      }
20  }
```

نتیجه

```
Message from parent.
Message from child.
Modified message of the parent.
Modified message of the child.
```

هر دو شی را با استفاده از سازنده های مربوط به خودشان مقدار دهی می کنیم.(خطوط ۶-۵)

سپس با استفاده از ارث بری و از طریق شی **Child** به اعضا و متدهای کلاس **parent** دسترسی می یابیم.

حتی اگر کلاس **Child** از کلاس **Parent** ارث ببرد باز هم اعضای با سطح دسترسی **Private** در کلاس **Child** قابل دسترسی نیستند(خط ۱۸).

سطح دسترسی **Protect** که در درس آینده توضیح داده خواهد شد به شما اجازه دسترسی به اعضا و متدهای کلاس پایه را می دهد.

به نکته دیگر توجه کنید. اگر کلاس دیگری بخواهد از کلاس **Child** ارث بری کند، باز هم تمام متدها و خواص کلاس **Child** که از کلاس **Parent** به ارث برده است را به ارث می برد.

```
class GrandChild : Child
{
    //Empty Body
}
```

این کلاس هیچ چیزی در داخل بدنه ندارد. وقتی کلاس **GrandChild** را ایجاد می کنید و یک خاصیت از کلاس **Parent** را فراخوانی می کنید با خطا مواجه می شوید.

چون هیچ سازنده ای که یک آرگومان رشته ای قبول کند در داخل بدنه **GrandChild** تعریف نشده است بنابراین شما می توانید فقط از سازنده پیشفرض یا بدون پارامتر استفاده کنید.

```
GrandChild myGrandChild = new GrandChild();
```

```
myGrandChild.Message = "Hello my grandchild!";
myGrandChild.ShowMessage();
```

وقتی یک کلاس ایجاد می کنیم و سازنده **GrandChild** را فراخوانی می کنیم ابتدا سازنده کلاس **Parent** فراخوانی می شود و سپس سازنده **Child** و در نهایت سازنده **GrandChild** اجرا می شود .

برنامه زیر ترتیب اجرای سازنده ها را نشان می دهد.

دوباره کلاسها را برای خوانایی بیشتر در داخل کدهای جدا قرار می دهیم.

```
using System;

class Parent
{
    public Parent()
    {
        Console.WriteLine("Parent constructor was called!");
    }
}

class Child : Parent
{
    public Child()
    {
        Console.WriteLine("Child constructor was called!");
    }
}

class GrandChild : Child
{
    public GrandChild()
    {
        Console.WriteLine("GrandChild constructor was called!");
    }
}

class Program
{
    static void Main()
    {
        GrandChild myGrandChild = new GrandChild();
    }
}
```

نتیجه

```
Parent constructor was called!
Child constructor was called!
GrandChild constructor was called!
```

سطح دسترسی Protect

سطح دسترسی **protect** اجازه می دهد که اعضای کلاس، فقط در کلاسهای مشتق شده از کلاس پایه قابل دسترسی باشند.
بدیهی است که خود کلاس پایه هم می تواند به این اعضا دسترسی داشته باشد.
کلاسهایی که از کلاس پایه ارث بری نکرده اند نمی توانند به اعضای با سطح دسترسی **protect** یابند.
در مورد سطوح دسترسی **public** و **private** قبلا توضیح دادیم.

در جدول زیر نحوه دسترسی به سه سطح ذکر شده نشان داده شده است :

قابل دسترسی در	public	private	protected
داخل کلاس	true	true	true
خارج از کلاس	true	false	false
کلاس مشتق	true	false	true

مشاهده می کنید که **public** بیشترین سطح دسترسی را داراست. صرف نظر از مکان، اعضای **public** در هر جا فراخوانی می شوند و قابل دسترسی هستند.
اعضای **private** فقط در داخل کلاسی که به آن تعلق دارند قابل دسترسی هستند.
اعضای **protect** فقط در کلاسی که در آن تعریف شده اند و همچنین کلاسهای مشتق شده از آن کلاس قابل دسترسی هستند.

کد زیر رفتار اعضای دارای این سه سطح دسترسی را نشان می دهد :

```
1   using System;
2
3   class Parent
4   {
5       protected int protectedMember = 10;
6       private int privateMember = 10;
7       public int publicMember = 10;
8   }
9
10  class Child : Parent
11  {
12      public Child()
13      {
14          protectedMember = 100;
15          privateMember = 100;
16          publicMember = 100;
17      }
18  }
19
20  class Program
21  {
22      public static void Main()
23      {
24          Parent myParent = new Parent();
25
26          myParent.protectedMember = 100;
27          myParent.privateMember = 100;
28          myParent.publicMember = 100;
29      }
30  }
```

کدهایی که با خط قرمز نشان داده شده اند نشان دهنده وجود خطا هستند چون آنها اجازه دسترسی به فیلدهای Protect کلاس Parent را ندارند.

همانطور که در خط ۱۵ مشاهده می کنید کلاس Child سعی می کند که به عضو Private کلاس Parent دسترسی یابد.

از آنجاییکه اعضای Private در خارج از کلاس قابل دسترسی نیستند، حتی کلاس مشتق در خط ۱۵ نیز ایجاد خطا می کند.

اگر شما به خط ۱۴ توجه کنید کلاس Child می تواند به عضو Protect کلاس Parent دسترسی یابد چون کلاس Child از کلاس Parent مشتق شده است.

حال به خط ۲۶ جاییکه می خواهیم در کلاس Program به فیلد Protect کلاس Parent دسترسی یابیم نگاهی بیندازید. می بینید که برنامه پیغام خطا می دهد چون کلاس Program از کلاس Parent مشتق نشده است.

همچنین کلاس Program به اعضای Private کلاس Parent نیز نمی تواند دسترسی یابد.

اعضای Static

اگر بخواهیم عضو داده ای (فیلد) یا خاصیتی ایجاد کنیم که در همه نمونه های کلاس قابل دسترسی باشد از کلمه کلیدی **static** استفاده می کنیم.

کلمه کلیدی **static** برای اعضای داده ای و خاصیت هایی به کار می رود که می خواهند در همه نمونه های کلاس تقسیم شوند.

وقتی که یک متد یا خاصیت به صورت **static** تعریف شود، می توانید آنها را بدون ساختن نمونه ای از شی، فراخوانی کنید.

به مثالی در مورد متدها و خاصیتهای **static** توجه کنید :

```
using System;

class SampleClass
{
    public static string StaticMessage { get; set; }
    public string NormalMessage { get; set; }

    public static void ShowStaticMessage()
    {
        Console.WriteLine(StaticMessage);
    }

    public void ShowNormalMessage()
    {
        Console.WriteLine(NormalMessage);
    }

    public void ShowStaticFromInstance()
    {
        Console.WriteLine(StaticMessage);
    }
}

class Program
{
    public static void Main()
    {
        SampleClass sample1 = new SampleClass();
        SampleClass sample2 = new SampleClass();

        SampleClass.StaticMessage = "This is the static message!";
        SampleClass.ShowStaticMessage();

        sample1.NormalMessage = "\nMessage from sample1!";
        sample1.ShowNormalMessage();
        sample1.ShowStaticFromInstance();

        sample2.NormalMessage = "\nMessage from sample2!";
        sample2.ShowNormalMessage();
        sample2.ShowStaticFromInstance();
    }
}
```

نتیجه

```
This is the static message!
```

```
Message from sample1!
This is the static message!

Message from sample2!
This is the static message!
```

در مثال بالا یک خاصیت استاتیک به نام StaticMessage (خط ۵) و یک متد استاتیک به نام ShowStaticMessage() (خطوط ۱۱-۸) تعریف کرده ایم.
مقدار خاصیت StaticMessage در همه نمونه های کلاس SampleClass قابل دسترسی است.
متد استاتیک را نمی توان به وسیله نمونه ایجاد شده از کلاس SampleClass فراخوانی کرد.
برای فراخوانی یک متد یا خاصیت استاتیک ، به سادگی می توان نام کلاس و بعد از آن علامت دات (.) و در آخر نام متد یا خاصیت را نوشت.
این موضوع را می توان در خطوط (۳۲-۳۱) مشاهده کرد. مشاهده می کنید که لازم نیست هیچ نمونه ای از کلاس ایجاد شود.
همانطور که مشاهده می کنید یک پیغام را به StaticMessage اختصاص داده ایم و با فراخوانی یک متد استاتیک (ShowStaticMessage)مقدار آن را نمایش می دهیم.
در مرحله بعد خاصیت NormalMessage را به وسیله دو نمونه ایجاد شده فراخوانی می کنیم و یک متد را هم برای نشان دادن مقدار آن فراخوانی می کنیم.
همانطور که مشاهده می کنید نمونه های ایجاد شده دارای مقدار NormalMessage مخصوص خودشان هستند چون خاصیت NormalMessage غیر استاتیک است.
سپس ShowStaticFromInstance() فراخوانی می کنیم که متدی برای نشان دادن مقدار StaticMessage می باشد.

متدهای غیر استاتیک می توانند از فیلدها و خاصیت های استاتیک استفاده کنند ولی عکس این قضیه امکان پذیر نیست. به عنوان مثال اگر شما یک متد static داشته باشید نمی توانید از هر خاصیت ، متد، یا فیلدی که static نیست استفاده کنید.

```
private int number = 10;

public static void ShowNumber()
{
   Console.WriteLine(number); //Error: cannot use non-static member
                                                                    }
```

اصرار بر این کار باعث بروز خطا می شود. یک مثال متد WriteLine کلاس Console است. اگر از یک متد غیر استاتیک استفاده می کنید باید به روش زیر عمل کنید :

```
Console display = new Console();

display.WriteLine("My message!");
```

شما قبل از نمایش پیغام ابتدا باید یک نمونه از کلاس Console ایجاد کنید .

به این نکته نیز توجه کنید که متد Main() باید به صورت static تعریف شود. کامپایلر به Static بودن متد Main() نیاز دارد تا بتواند برنامه را بدون ساختن نمونه اجرا کند.

متدهای مجازی

متدهای مجازی متدهایی از کلاس پایه هستند که می توان به وسیله یک متد از کلاس مشتق آن را **override** کرده و به صورت دلخواه پیاده سازی نمود.

به عنوان مثال شما متد **A** را در کلاس **A** دارید و کلاس **B** از کلاس **A** ارث بری می کند، در این صورت متد **A** در کلاس **B** در دسترس خواهد بود.

اما متد **A** دقیق همان متدی است که از کلاس **A** به ارث برده شده است.

حال اگر بخواهید که این متد رفتار متفاوتی از خود نشان دهد چکار می کنید؟

متد مجازی این مشکل را برطرف می کند.

به تکه کد زیر توجه کنید.

```
1   using System;
2
3   class Parent
4   {
5       public virtual void ShowMessage()
6       {
7           Console.WriteLine("Message from Parent.");
8       }
9   }
10
11  class Child : Parent
12  {
13      public override void ShowMessage()
14      {
15          Console.WriteLine("Message from Child.");
16      }
17  }
18
19  class Program
20  {
21      public static void Main()
22      {
23          Parent myParent = new Parent();
24          Child myChild = new Child();
25
26          myParent.ShowMessage();
27          myChild.ShowMessage();
28      }
29  }
```

نتیجه

```
Message from Parent.
Message from Child.
```

متد مجازی با قرار دادن کلمه کلیدی **virtual** هنگام تعریف متد، تعریف می شود.(خط ۵)

این کلمه کلیدی نشان می دهد که متد می تواند **override** شود یا به عبارت دیگر می تواند به صورت دیگر پیاده سازی شود .

کلاسی که از کلاس **Parent** ارث می برد شامل متدی است که متد مجازی کلاس پایه را **override** یا به صورت دیگری پیاده سازی می کند.

با استفاده از کلمه کلیدی **override** می توان متد مجازی کلاس پایه را به صورت دیگر پیاده سازی کرد.

با استفاده از کلمه کلیدی **base** (خط ۱۷) می توانید متد مجازی را در داخل متد **override** شده فراخوانی کنید.

```
1   using System;
2
3   class Parent
4   {
5       private string name = "Parent";
6
7       public virtual void ShowMessage()
8       {
9           Console.WriteLine("Message from Parent.");
10      }
11  }
12
13  class Child : Parent
14  {
15      public override void ShowMessage()
16      {
17          base.ShowMessage();
18          Console.WriteLine("Message from Child.");
19      }
20  }
21
22  class Program
23  {
24      public static void Main()
25      {
26          Parent myParent = new Parent();
27          Child myChild = new Child();
28
29          myParent.ShowMessage();
30          myChild.ShowMessage();
31      }
32  }
```

نتیجه

```
Message from Parent.
Message from Parent.
Message from Child.
```

اگر بخواهید از متد پایه استفاده کنید و هیچ کدی داخل متد **override** نباشد (مثلا فرض کنید خط ۱۸ در مثال بالا وجود نداشته باشد)، بسیار شبیه به این است که از هیچ متد **override** ی استفاده نکرده اید.

نمی توان متد غیر **virtual** و یا یک متد **static** را **override** کرد.

متد **override** نیز باید دارای سطح دسترسی شبیه به متد مجازی باشد.

می توان یک کلاس دیگر که از کلاس **Child** ارث بری می کند ایجاد کرده و دوباره متد **ShowMessage** را **override** کرده و آنرا به صورت دیگر پیاده سازی کنیم.

اگر بخواهید متدی را که ایجاد کرده اید به وسیله سایر کلاسها **override** نشود کافیست که از کلمه کلیدی **sealed** به صورت زیر استفاده کنید :

```
public sealed override void ShowMessage()
```

حال اگر کلاس دیگری از کلاس **Child** ارث ببرد نمی تواند متد **ShowMessage()** را **override** کند.

به یک مثال دیگر توجه کنید. فرض کنید می خواهیم متد **ToString()** کلاس **System.Object** را **override** کنیم.

همانطور که در درس آینده خواهید دید همه کلاسها در سی شارپ از کلاس **Object** ارث می برند.

```
1   using System;
2
3   class Person
4   {
5       public string FirstName { get; set; }
6       public string LastName { get; set; }
7
8       public override string ToString()
9       {
10          return FirstName + " " + LastName;
11      }
12  }
13
14  class Program
15  {
16      public static void Main()
17      {
18          Person person1 = new Person();
19
20          person1.FirstName = "John";
21          person1.LastName = "Smith";
22
23          Console.WriteLine(person1.ToString());
24      }
25  }
```

نتیجه

```
John Smith
```

از آنجاییکه متد **ToString()** کلاس **System.Object** را **override** کرده ایم ، به جای چاپ نوع شی پیشفرض خروجی ما سفارشی شده و نام و نام خانوادگی نمایش داده می شود.