سی شارپ به زبان ساده

مؤلف : یونس ابراهیمی

سر شناسه : ابراهیمی، یونس - ۱۳۶۰

عنوان و نام پدید آورنده : سی‌شارپ به زبان ساده / مؤلف: یونس ابراهیمی

مشخصات نشر : نبض دانش، تهران - ۱۳۹۵

مشخصات ظاهری : ۷۵۴ صفحه مصور

شابک : ۹۷۸-۶۰۰-۷۷۰۳-۹۴-۶

وضعیت فهرست نویسی : فیپا

موضوع : سی‌شارپ (زبان برنامه نویسی کامپیوتر)

رده بندی دیویی : ۰۰۵/۱۳۳

رده بندی کنگره : ۱۳۹۵ ۲ الف ۹۵ س /QA۷۶/۷۳

شماره کتاب شناسی ملی : ۴۳۳۰۵۰۷

عنوان : سی‌شارپ به زبان ساده

مؤلف : یونس ابراهیمی

ناشر : نبض دانش

سال چاپ : ۱۳۹۵

نوبت چاپ : دوم

تیراژ : ۲۰۰

قیمت : ۵۹۰۰۰ تومان

تقدیم به:

همسر و پسر عزیزم

مبانی زبان سی شارپ

ویندوز فرم

دات نت فریم ورک

LINQ

ADO.NET

معماری سه لایه

مقدمه

همگام با پیشرفت فناوری‌های دیگر، زبان‌های برنامه نویسی نیز ارتقا پیدا کردند. وقتی زبان C طراحی و پیاده سازی شد، تحول بزرگی در دنیای برنامه نویسی به وجود آمد. زبان‌های متعددی از خانواده زبان C طراحی و پیاده سازی شدند که محبوب‌ترین آنها زبان سی‌شارپ است. خوشبختانه C#.NET این روزها به عنوان یکی از دروس رشته‌های کامپیوتر در دانشگاه‌های کشور تدریس می‌شود و این نشان از توانایی‌ها و اهمیت این زبان است. منابع متعددی برای معرفی و به کارگیری زبان C#.NET عرضه شده است که جای تقدیر و تشکر دارد. اما کتاب حاضر دارای ویژگی‌های بارزی از جمله، بیان ساده مطالب، ارائه مثال‌های متنوع، بررسی دقیق و موشکافانه موضوعات و سلسله مراتب آموزشی است. مثال‌هایی که در کتاب ارائه شده‌اند همگی دارای هدف خاصی هستند به طوریکه هر کدام، یک یا چند نکته زبان سی‌شارپ را به خواننده آموزش می‌دهند. در این کتاب ما به شما نحوه برنامه نویسی به زبان سی‌شارپ را به صورت تصویری آموزش می‌دهیم. سعی کنید حتماً بعد از خواندن مباحث، آنها را به صورت عملی تمرین کنید و اینکه قابلیت و مفهوم کدها را بفهمید نه آنها را حفظ کنید.

بی شک این اثر، خالی از اشکال نیست و از شما خوانندگان عزیز می‌خواهم که با نظرات و پیشنهادات خود بنده را در تکمیل و رفع نواقص آن از طریق پست الکترونیکی younes.ebrahimi.1391@gmail.com یاری بفرمایید.

در پایان جا دارد از مهندسان عزیز، آقای سیاوش ابراهیمی و محمد ابراهیمی که در تکمیل دو بخش LINQ و ویندوز فرم اینجانب را کمک کردند صمیمانه تشکر کنم. امیدوارم این هدیه ناقابل را از بنده پذیرا باشند.

برای دریافت فایل‌ها و آپدیت‌های جدید این کتاب به سایت **www.w3-farsi.com** مراجعه فرمایید.

راه‌های ارتباط با نویسنده

وب سایت	: www.w3-farsi.com
لینک تلگرام	: https://telegram.me/ebrahimi_younes
ID تلگرام	: @ebrahimi_younes
پست الکترونیکی	: younes.ebrahimi.1391@gmail.com

فصل اول

مبانی زبان سی شارپ

سی‌شارپ چیست؟

سی‌شارپ (C#) یک زبان برنامه نویسی شیء گرا است که توسط شرکت مایکروسافت ساخته شده و ترکیبی از قابلیت‌های خوب C++ و Java است. اگر با این دو زبان آشنایی دارید این شانس را دارید زبان سی‌شارپ را راحت یاد بگیرید. این زبان به قدری راحت است که هم کسانی که قبلاً برنامه نویسی نکرده‌اند و هم دانش آموزان می‌توانند راحت آن را یاد بگیرند. از سی‌شارپ می‌توان برای ساخت برنامه‌های تحت ویندوز، تحت وب، وب سرویس‌ها، برنامه‌های موبایل و بازی‌ها استفاده کرد. می‌توان به جای واژه ویژوال سی‌شارپ از کلمه سی‌شارپ استفاده کرد، اما ویژوال سی‌شارپ به معنای استفاده همزمان از سی‌شارپ و محیط گرافیکی ویژوال استودیو می‌باشد. این زبان برنامه نویسی تنها زبانی است که مخصوصاً برای دات نت فریم ورک طراحی شده است.

سی‌شارپ از کتابخانه کلاس دات نت که شامل مجموعه بزرگی از اجزاء از قبل ساخته شده است، استفاده می‌کند. این اجزا به ساخت هر چه سریع‌تر برنامه‌ها کمک می‌کنند. سی‌شارپ یک برنامه بسیار قدرتمند و شیء گرا است و با آن می‌توان برنامه‌هایی 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 سی‌شارپ ابتدا باید دات نت فریم ورک نصب شود. سی‌شارپ یکی از زبان‌هایی است که از تکنولوژی‌های دیگر دات نت مانند ASP.NET، Silverlight و XNA پشتیبانی می‌کند. همچنین یک محیط توسعه یکپارچه دارد که آن نیز به نوبه خود دارای ابزارهای مفیدی است که به شما در کدنویسی در سی‌شارپ کمک می‌کند.

با ظهور C# 7.0 قابلیت های جدیدی به این زبان اضافه شد که به شما امکان می دهند که برنامه هایی بهینه تر و پربار تر با کدنویسی کمتر بنویسید. حال که اسم نسخه ۷ سی شارپ به میان آمد بهتر است که با نسخه های مختلف این زبان از ابتدا تاکنون که در جدول زیر آمده است آشنا شوید:

نسخه سی‌شارپ	نسخه .NET Framework	نسخه Visual Studio	تاریخ ارائه
C# 1.0	.NET Framework 1.0	Visual Studio .NET 2002	January 2002
C# 1.1	.NET Framework 1.1	Visual Studio .NET 2003	April 2003
C# 2.0	.NET Framework 2.0	Visual Studio 2005	November 2005
C# 3.0	.NET Framework 3.0\3.5	Visual Studio 2008	November 2007
C# 4.0	.NET Framework 4.0	Visual Studio 2010	April 2010
C# 5.0	.NET Framework 4.5	Visual Studio 2012/2013	August 2012
C# 6.0	.NET Framework 4.6	Visual Studio 2015	July 2015
C# 7.0	.NET Framework 4.6.2	Visual Studio 2017	March 2017

دلیل پیدایش این زبان بر طبق دانشنامه Wikipedia بدین شرح است که:

در سال ۱۹۹۹، شرکت Sun Microsystems اجازه استفاده از زبان برنامه‌نویسی JAVA را در اختیار Microsoft قرار داد تا در سیستم عامل خود از آن استفاده کند. جاوا در اصل به هیچ پلت فرم یا سیستم عاملی وابسته نبود، ولی مایکروسافت برخی از مفاد قرار داد را زیر پا گذاشت و قابلیت مستقل از سیستم عامل بودن جاوا را از آن برداشت. شرکت Sun Microsystems پرونده‌ای علیه مایکروسافت درست کرد و مایکروسافت مجبور شد تا زبان شیءگرای جدیدی با کامپایلر جدید که به C++ شبیه بود را درست کند. آندرس هلزبرگ (Anders Hejlsberg) سرپرستی و مدیریت این پروژه را بر عهده گرفت و گروهی را برای طراحی زبانی جدید تشکیل داد و نام آن را Cool گذاشت. مایکروسافت در نظر داشت اسم این زبان را تا آخر Cool قرار دهد، ولی به دلیل مناسب نبودن برای اهداف تجاری این کار را نکرد. در ارائه و معرفی رسمی چارچوب دات‌نت در سال ۲۰۰۰ این زبان به سی‌شارپ تغییر نام یافت.

برای آشنایی بیشتر با این زبان به لینک زیر مراجعه کنید:

```
http://msdn.microsoft.com/en-us/library/z1zx9t92.aspx
```

سی‌شارپ به طور دائم توسط مایکروسافت به روز شده و ویژگیهای جدیدی به آن اضافه می‌شود و یکی از بهترین زبان‌های برنامه نویسی دات نت است.

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. تاکنون 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 کلاس‌ها، ساختارها، داده‌های شمارشی و.... می‌باشد. مهم‌ترین قسمت .NET Framework زبان مشترک زمان اجرا (CLR) است که محیطی را فراهم می‌آورد که برنامه‌ها در آن اجرا شوند. این چارچوب ما را قادر می‌سازد که برنامه‌هایی که تحت آن نوشته شده‌اند اعم از C#.Net، Visual Basic.Net و C++ را بهتر درک کنیم. کدهایی که تحت CLR و دات‌نت اجرا می‌شوند، کدهای مدیریت شده نامیده می‌شوند، چون CLR جنبه‌های مختلف نرم‌افزار را در زمان اجرا مدیریت می‌کند. در زمان کامپایل کدها به زبان مشترک میانی (CIL) که نزدیک و تقریباً شبیه به زبان اسمبلی است ترجمه می‌شوند. ما باید کدهایمان را به این زبان ترجمه کنیم، چون فقط این زبان برای دات‌نت قابل فهم است. برای مثال کدهای C# و Visual Basic.Net هر دو به زبان مشترک میانی (CIL) ترجمه می‌شوند. به همین دلیل است که برنامه‌های مختلف در دات‌نت که با زبان‌های متفاوتی نوشته شده‌اند می‌توانند با هم ارتباط برقرار کنند. اگر یک زبان سازگار با دات‌نت می‌خواهید 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 می‌شوند. کدهای ترجمه شده به زبان میانی به کامپایلر فقط در زمان (JIT) منتقل می‌شوند. این کامپایلر

در لحظه فقط کدهایی را که برنامه در آن زمان نیاز دارد به زبان ماشین ترجمه می‌کند. در زیر نحوه تبدیل کدهای سی‌شارپ به یک برنامه اجرایی به طور خلاصه آمده است:

- برنامه نویس برنامه خود را با یک زبان دات‌نت مانند سی‌شارپ می‌نویسد.
- کدهای سی‌شارپ به کدهای معادل آن در زبان میانی تبدیل می‌شوند.
- کدهای زبان میانی در یک فایل اسمبلی ذخیره می‌شوند.
- وقتی کدها اجرا می‌شوند کامپایلر JIT کدهای زبان میانی را در لحظه به کدهایی که برای کامپیوتر قابل خواندن باشند تبدیل می‌کند.

دات نت ویژگی دیگری به نام سیستم نوع مشترک (CTS) نیز دارد که بخشی از CLR است و نقشه‌ای برای معادل سازی انواع داده‌ها در دات‌نت می‌باشد. با CTS نوع int در سی‌شارپ و نوع Integer در ویژوال بیسیک یکسان هستند، چون هر دو از نوع System.Int32 مشتق می‌شوند. پاک کردن خانه‌های بلا استفاده حافظه در یک فایل (Garbage collection) یکی دیگر از ویژگی‌های دات‌نت فریم ورک است. هنگامی که از منابعی، زیاد استفاده نشود دات‌نت فریم ورک حافظه استفاده شده توسط برنامه را آزاد می‌کند.

ویژوال استودیو

ویژوال استودیو محیط توسعه یکپارچه ای است، که دارای ابزارهایی برای کمک به شما برای توسعه برنامه های سی شارپ و دات نت می باشد.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 اینتلی سنس (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.

در حال حاضر آخرین نسخه ویژوال استودیو Visual Studio 2017 است. این نسخه به دو نسخه Visual Studio Professional (ارزان قیمت) و Visual Studio Enterprise (گرانقیمت) تقسیم می شود و دارای ویژگی های متفاوتی هستند. خبر خوب برای توسعه دهندگان نرم افزار این است که مایکروسافت تصمیم دارد که ویژوال استودیو را به صورت متن باز ارائه دهد. یکی از نسخه های ویژوال استودیو،

Visual Studio Community می باشد که آزاد است و می توان آن 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 (Community) کامل نیست و خلاصه شده نسخه اصلی است. به هر حال استفاده از Visual Studio Community که جایگزین Visual Studio Express شده و به نوعی همان نسخه Visual Studio Professional است، برای انجام تمرینات این سایت کافی است.

Visual Studio Enterprise 2017 دارای محیطی کاملتر و ابزارهای بیشتری جهت عیب یابی و رسم نمودارهای مختلف است که در Visual Studio Community 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 و نصب ویژوال استودیو

در این درس می‌خواهیم نحوه دانلود و نصب نرم افزار Visual Studio Community 2017 را آموزش دهیم. در جدول زیر لیست نرم افزارها و سخت افزارهای لازم جهت نصب ویژوال استودیو ۲۰۱۷ آمده است:

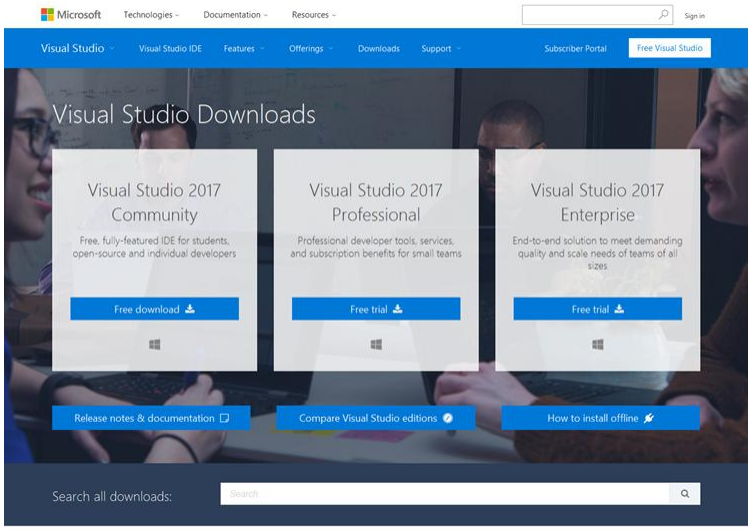
سیستم عامل	سخت افزار
Windows 10	1.6 GHz or faster processor
Windows 8.1	1 GB of RAM (1.5 GB if running on a virtual machine)
Windows 8	4 GB of available hard disk space
Windows 7 Service Pack 1	5400 RPM hard disk drive
Windows Server 2012 R2	DirectX 9-capable video card that runs at 1024 x 768 or higher display resolution
Windows Server 2012	
Windows Server 2008 R2 SP1	

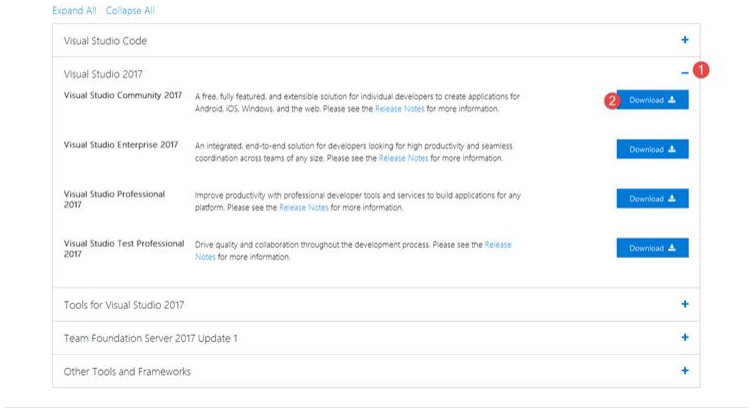
دانلود Visual Studio Community 2017

Visual Studio Community 2017 به صورت آزاد در دسترس است و می‌توانید آن را از لینک زیر دانلود کنید:

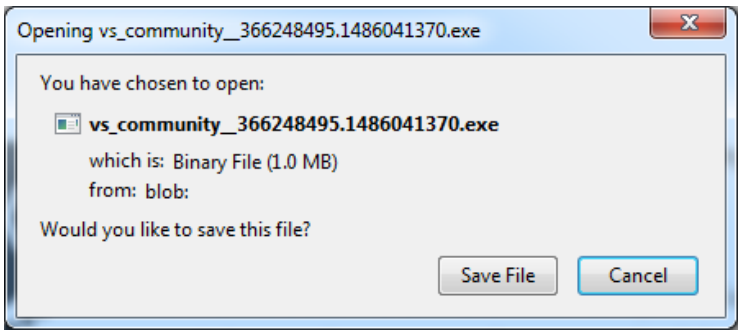
```
https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx
```

با کلیک بر روی لینک بالا صفحه ای به صورت زیر ظاهر می شود که در داخل این صفحه می توان با کلیک بر روی Visual Studio Community 2017آن را دانلود کرد:

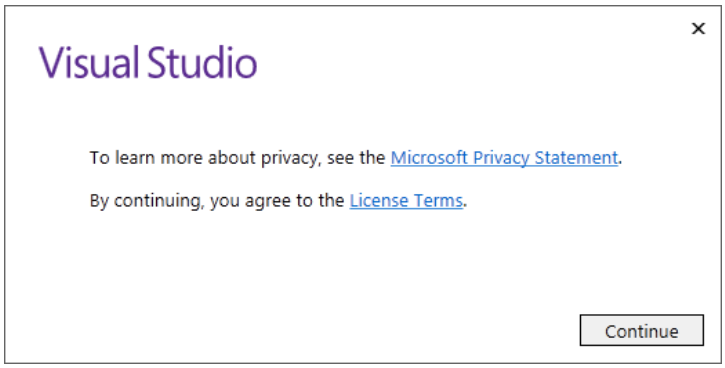


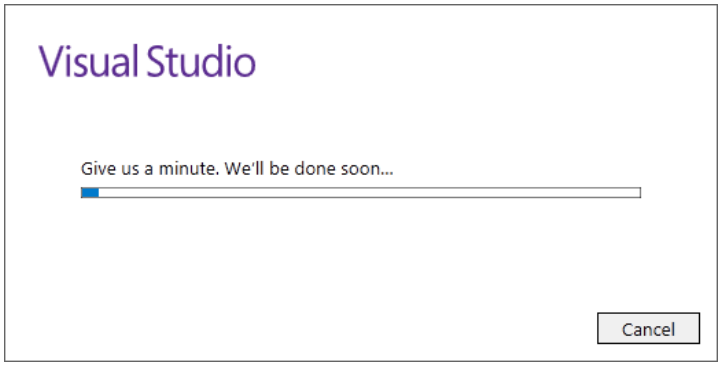


بعد از کلیک بر روی گزینه `Download` یک صفحه به صورت زیر باز می شود و از شما می خواهد که فایلی با نام `vs_community.exe` را ذخیره کنید:

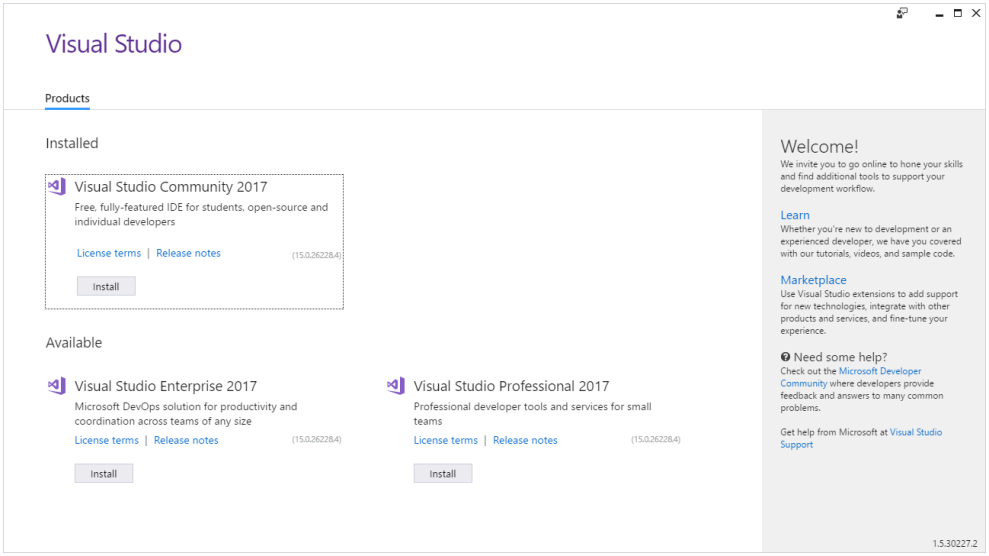


با ذخیره و اجرای این فایل مراحل نصب `Visual Studio Community 2017` آغاز می شود (`Visual Studio Community 2017` حدود ۵ گیگابایت حجم دارد و برای دانلود آن به یک اینترنت پر سرعت دارید):

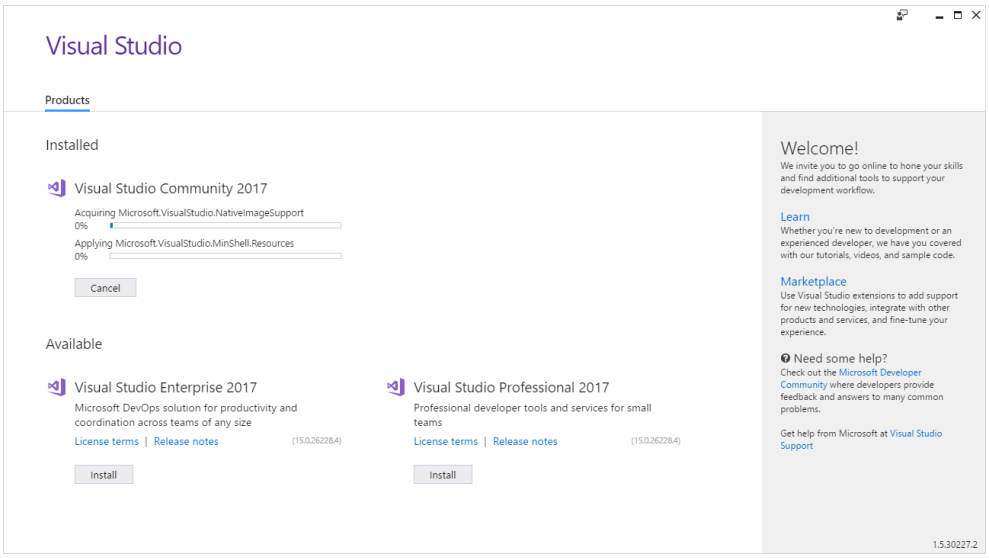




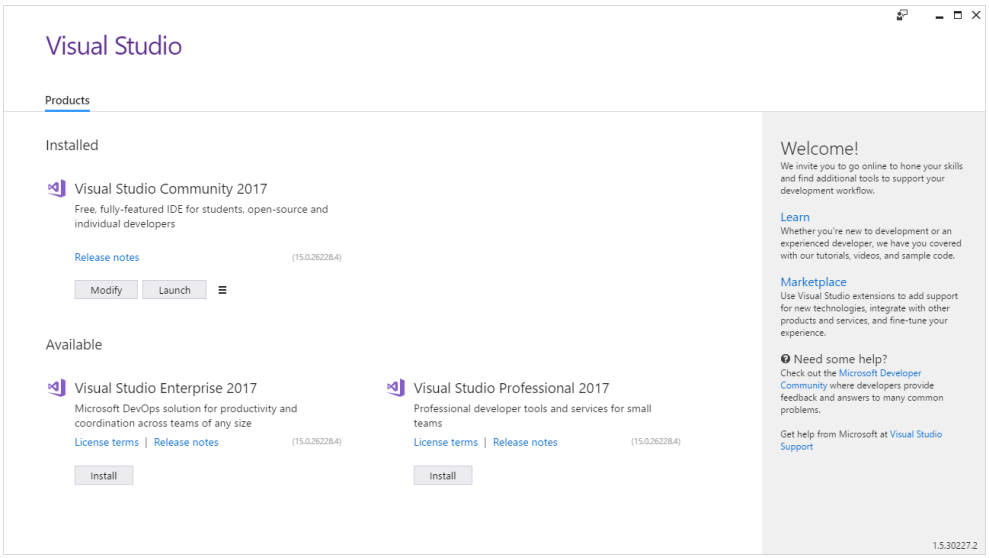
بعد از گذراندن دو صفحه بالا صفحه ای به صورت زیر باز می شود که در آن نسخه های مختلف ویژوال استودیو به شما نمایش داده می شود. بر روی گزینه `Install` روبروی `Visual Studio Community` کلیک کنید:



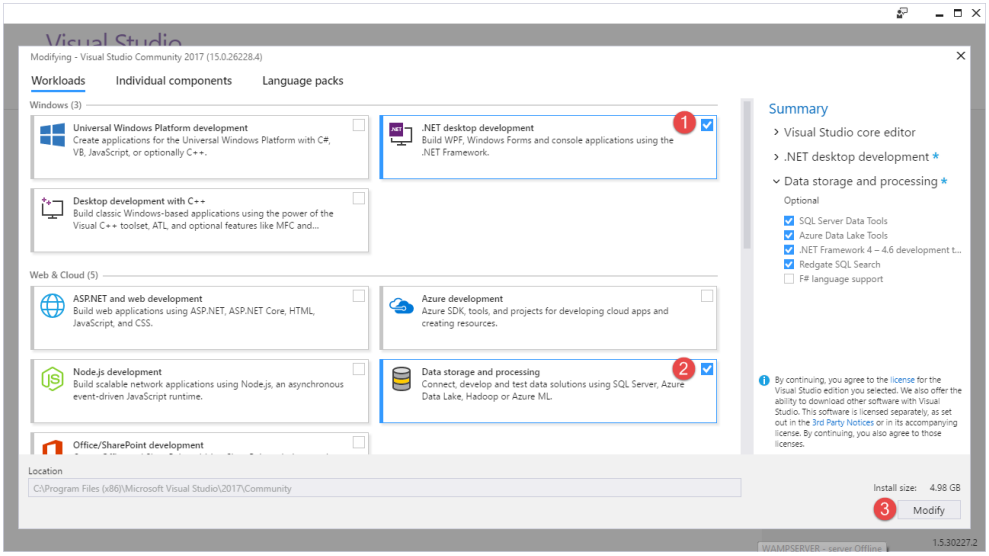
بعد از کلیک بر روی دکمه `Install` مرحله نصب شروع می شود:



بعد از اتمام مرحله بالا صفحه ای به صورت زیر باز می شود:



در صفحه بالا بر روی گزینه `Modify` کلیک کنید و گزینه های زیر را تیک بزنید و سپس بر روی دکمه `Modify` کلیک کنید:



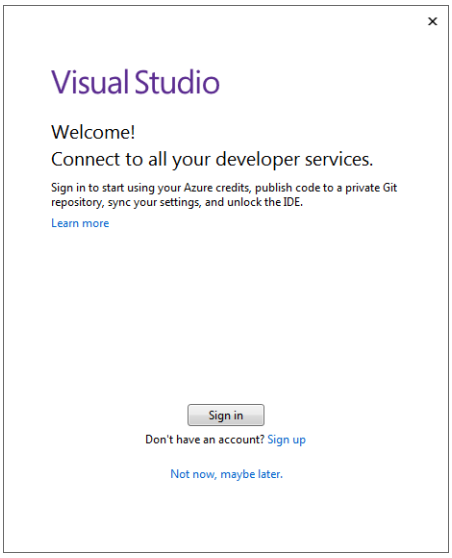
بعد از این مرحله ویژوال استودیو به صورت کامل نصب شده و شما می توانید از آن استفاده کنید.

شروع کار با `Visual Studio Community`

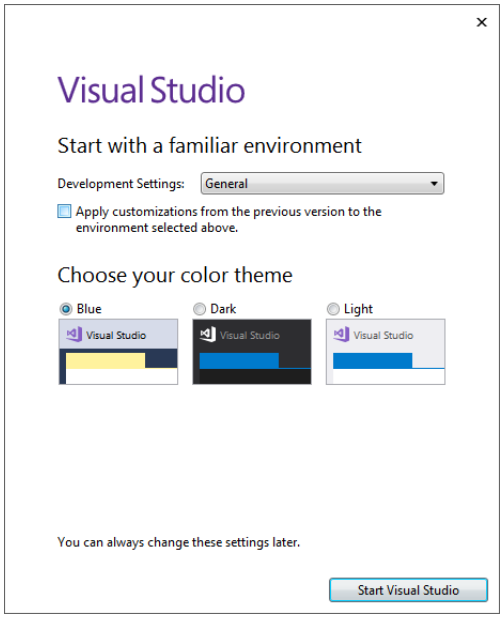
برنامه ویژوال استودیو را اجرا کرده و منتظر بمانید تا صفحه آن بارگذاری شود:

اگر دارای یک اکانت مایکروسافت باشید می توانید تغییراتی که در ویژوال استودیو می دهید را در فضای ابری ذخیره کرده و اگر آن را در کامپیوتر دیگر نصب کنید، می توانید با وارد شده به اکانت خود، تغییرات را به صورت خودکار بر روی ویژوال استودیویی که تازه نصب شده اعمال کنید.

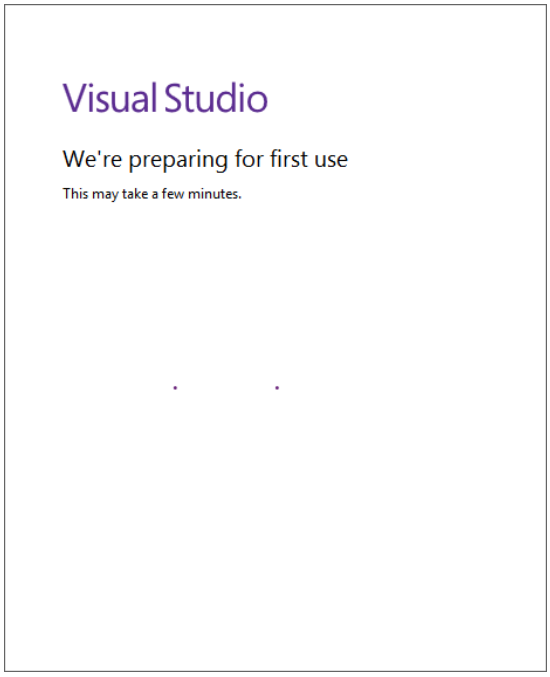
البته می توانید این مرحله را با زدن دکمه `Not now, maybe later` رد کنید:



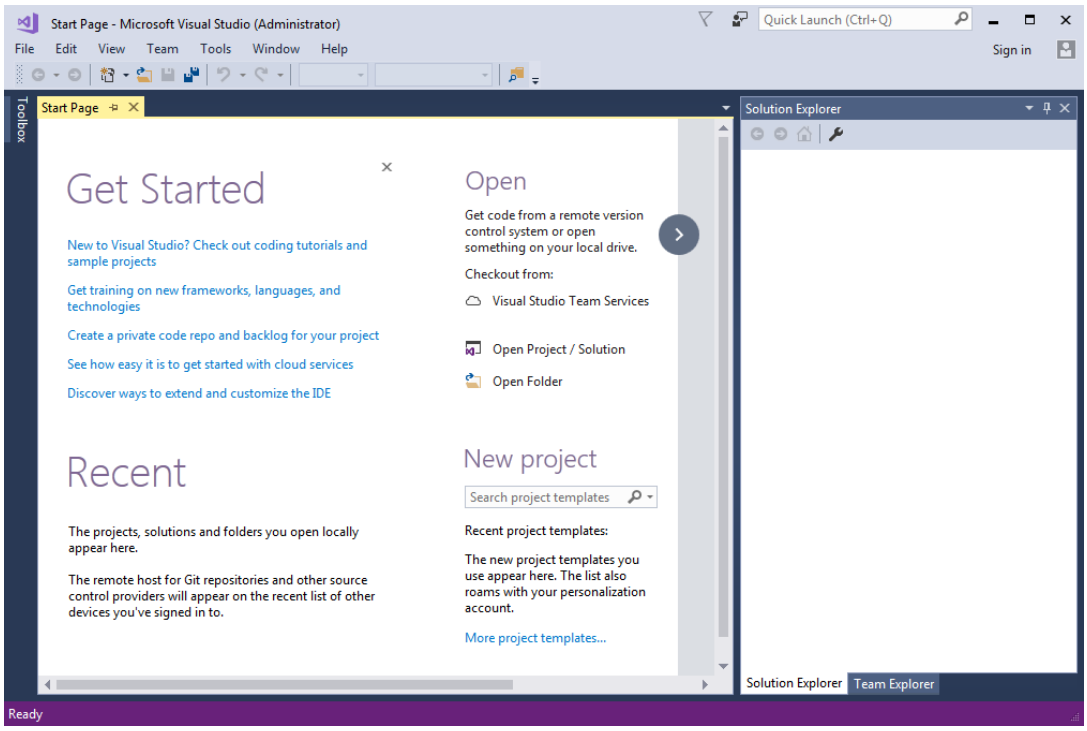
شما می توانید از بین سه ظاهر از پیش تعریف شده در ویژوال استودیو یکی را انتخاب کنید. من به صورت پیشفرض ظاهر `Blue` را انتخاب می کنم ولی شما می توانید بسته به سلیقه خود، ظاهر دیگر را انتخاب کنید:



بعد از زدن دکمه `Start Visual Studio` صفحه ای به صورت زیر ظاهر می شود:

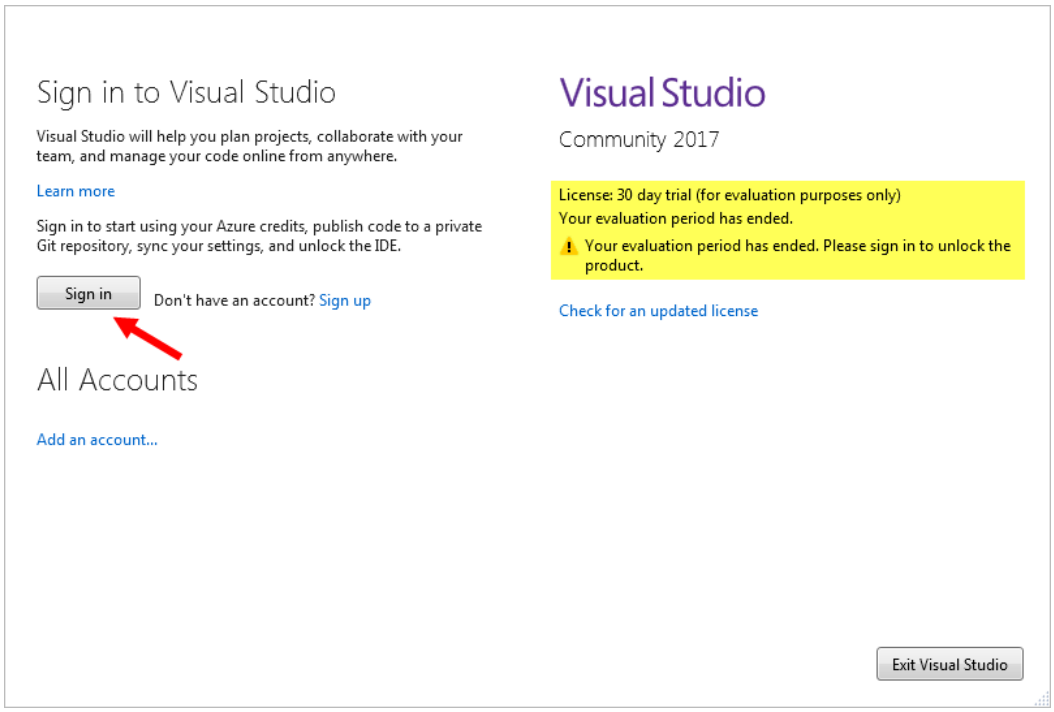


بعد از بارگذاری کامل `Visual Studio Community` صفحه اصلی برنامه به صورت زیر نمایش داده می شود که نشان از نصب کامل آن دارد :



قانونی کردن ویژوال استودیو

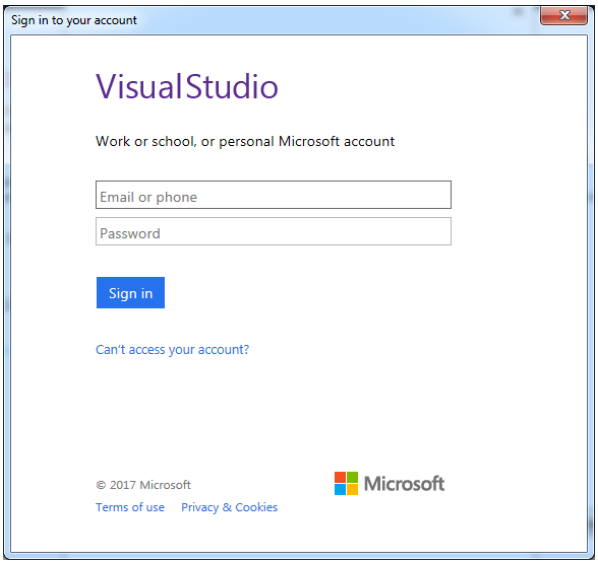
Visual Studio Community 2017 رایگان است. ولی گاهی اوقات ممکن است با پیغامی به صورت زیر مبنی بر منقضی شدن آن مواجه شوید :



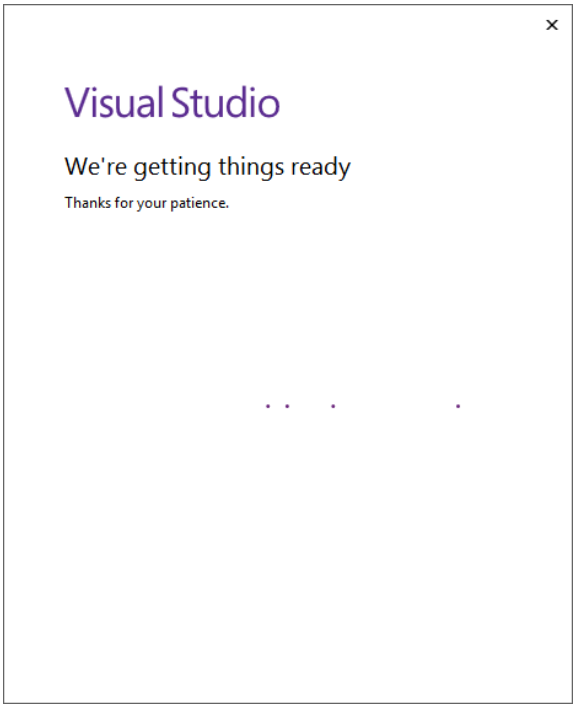
همانطور که در شکل بالا مشاهده می کنید، بر روی دکمه Signin کلیک می کنید تا وارد اکانت مایکروسافت خود شوید. اگر اکانت ندارید، می توانید از لینک زیر یک اکانت ایجاد کنید :

```
goo.gl/hMPYnE
```

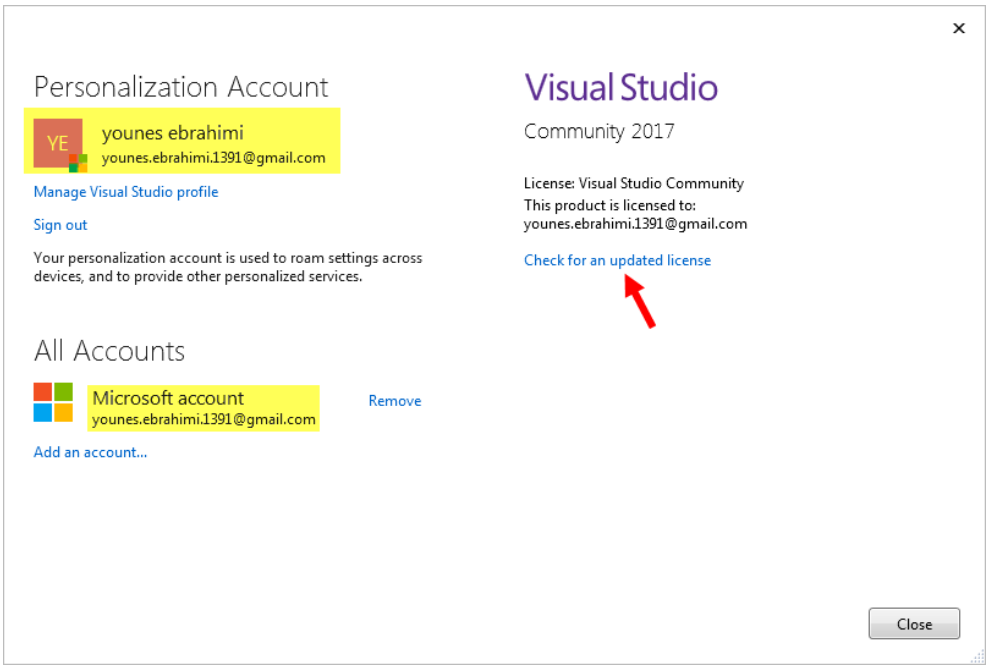
بعد از ایجاد اکانت همانطور که در شکل بالا مشاهده می کنید، بر روی گزینه Singin کلیک می کنیم. با کلیک بر روی این گزینه صفحه ای به صورت زیر ظاهر می شود که از شما مشخصات اکانتتان را می خواهد، آنها را وارد کرده و بر روی گزینه Singin کلیک کنید :



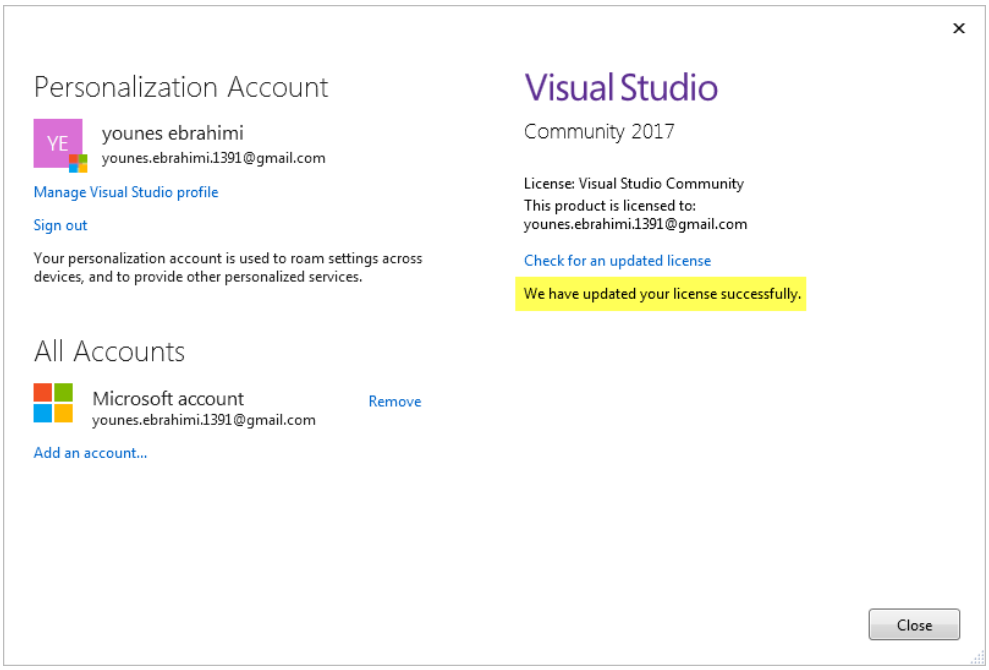
با کلیک بر روی گزینه `Signin` پنجره ای به صورت زیر نمایش داده می شود، منتظر می مانید تا پنجره بسه شود:



با بسته شدن پنجره بالا، پنجره ای به صورت زیر ظاهر می شود که مشخصات اکانت شما در آن نمایش داده می شود، که نشان از ورود موفقیت آمیز شما دارد. در این صفحه بر روی گزینه `Check an updated license` کلیک کنید:



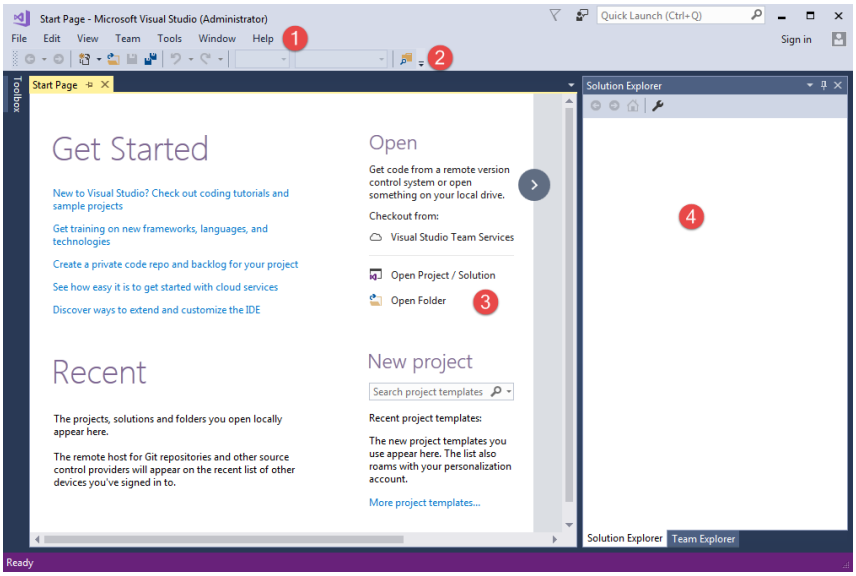
با کلیک بر روی این گزینه بعد از چند ثانیه پیغام `we have updated your license successfully` نمایش داده می شود و به این صورت ویژوال استودیو قانونی می شود:



به ویژوال استودیو خوش آمدید

در این بخش می‌خواهیم درباره قسمت‌های مختلف محیط ویژوال استودیو به شما مطالبی آموزش دهیم. لازم است که با انواع ابزارها و ویژگی‌های این محیط آشنا شوید. برنامه ویژوال استودیو را اجرا کنید :

بعد از اینکه صفحه بالا بسته شد وارد صفحه آغازین ویژوال استودیو می‌شویم :



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

منو بار (Menu Bar)

منو بار (۱)، شامل منوهای مختلفی برای ساخت، توسعه، نگهداری، خطایابی و اجرای برنامه‌ها است. با کلیک بر روی هر منو دیگر منوهای وابسته به آن ظاهر می‌شوند. به این نکته توجه کنید که منوبار دارای آیتم‌های مختلفی است که فقط در شرایط خاصی ظاهر می‌شوند. به عنوان مثال آیتم‌های 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 پروژه‌ها و خروج از آنها می‌باشد.
Edit	شامل دستوراتی جهت ویرایش از قبیل کپی کردن، جایگزینی و پیدا کردن یک مورد خاص می‌باشد.
View	به شما اجازه می‌دهد تا پنجره‌های بیشتری باز کرده و یا به آیتم‌های toolbar آیتمی اضافه کنید.
Project	شامل دستوراتی در مورد پروژه‌ای است که شما بر روی آن کار می‌کنید.
Debug	به شما اجازه کامپایل، اشکال زدایی و اجرای برنامه را می‌دهد.
Data	شامل دستوراتی برای اتصال به دیتابیس‌ها می‌باشد.
Format	شامل دستوراتی جهت مرتب کردن اجزای گرافیکی در محیط گرافیکی برنامه می‌باشد.
Tools	شامل ابزارهای مختلف، تنظیمات و ... برای ویژوال سی‌شارپ و ویژوال استودیو می‌باشد.
Window	به شما اجازه تنظیمات ظاهری پنجره‌ها را می‌دهد.
Help	شامل اطلاعاتی در مورد برنامه ویژوال استودیو می‌باشد.

نوار ابزار (Toolbars)

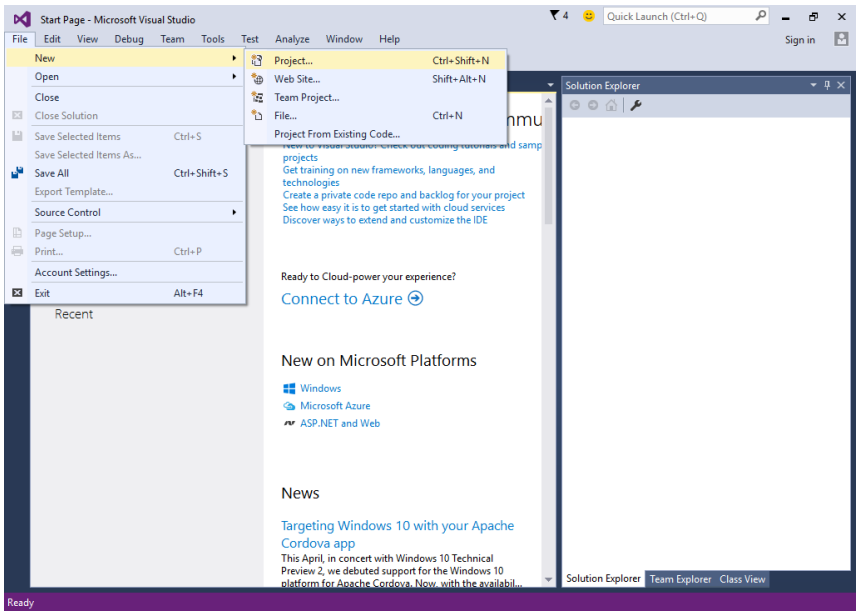
Toolbar (۲)، به طور معمول شامل همان دستوراتی است که در داخل منوها قرار دارند. Toolbar همانند یک میانبر عمل می‌کند. هر دکمه در Toolbar دارای آیکونی است که کاربرد آنرا نشان می‌دهد. اگر در مورد عملکرد هر کدام از این دکمه‌ها شک داشتید، می‌توانید با نشانگر ماوس بر روی آن مکث کوتاهی بکنید تا کاربرد آن به صورت یک پیام (tool tip) نشان داده شود. برخی از دستورات مخفی هستند و تحت شرایط خاص ظاهر می‌شوند. همچنین می‌توانید با کلیک راست بر روی منطقه خالی از Toolbar و یا از مسیر View > Toolbars دستورات بیشتری به آن اضافه کنید. برخی از دکمه‌ها دارای فلش‌های کوچکی هستند که با کلیک بر روی آنها دیگر دستورات وابسته به آنها ظاهر می‌شوند. سمت چپ هر Toolbar به شما اجازه جا به جایی آن را می‌دهد.

صفحه آغازین(Start Page)

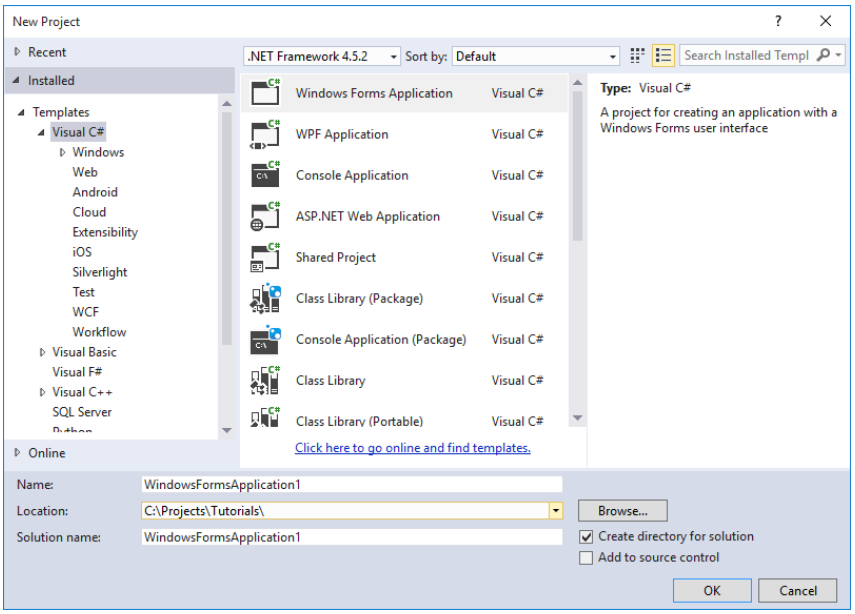
Start Page (۳)، برای ایجاد یک پروژه و باز کردن، مورد استفاده قرار می‌گیرد. همچنین اگر از قبل پروژه‌ای ایجاد کرده‌اید می‌توانید آن را در Recent Projects مشاهده و اجرا کنید. بخشهای مهم ویژوال استودیو توضیح داده شد در مورد بخش‌های بعدی در درس‌های آینده توضیحات بیشتری خواهیم داد.

گردشی در ویژوال استودیو

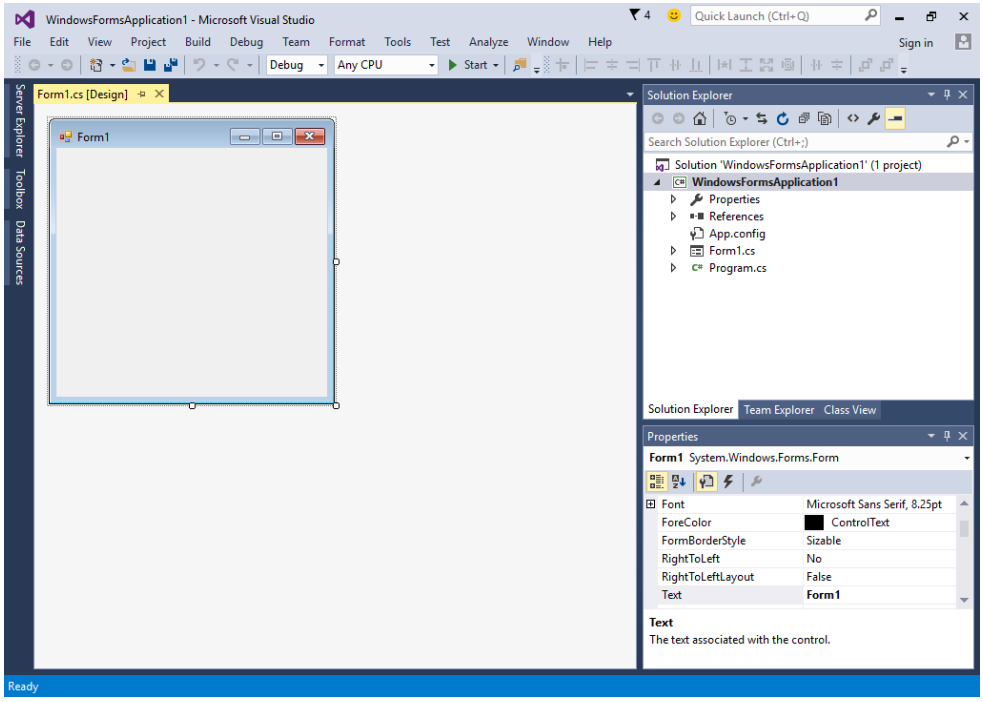
Visual Studio Community از تعداد زیادی پنجره و منو تشکیل شده است که هر کدام برای انجام کار خاصی به کار می‌روند. اجازه دهید با نفوذ بیشتر در محیط ویژوال استودیو با این قسمت‌ها آشنا شویم. از مسیر File > New Project یک پنجره فرم ایجاد کنید.



پنجره‌ای به شکل زیر نمایش داده خواهد شد.



همانطور که در شکل بالا نشان داده شده است، گزینه `Windows Forms Application` و یک اسم برای پروژه انتخاب می‌کنیم و بر روی دکمه OK کلیک می‌کنیم تا صفحه زیر نمایان شود:



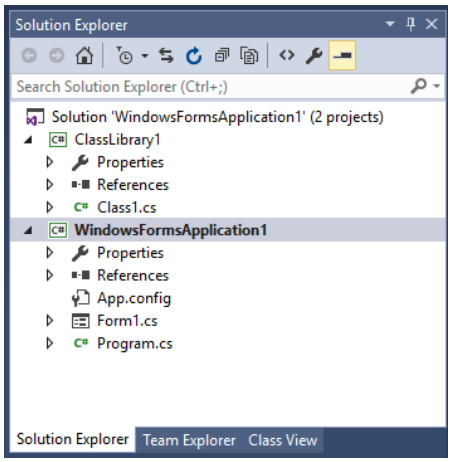
صفحه طراحی(Design)

این صفحه در حکم یک ناحیه برای طراحی فرم‌های ویندوزی شما است. فرم‌های ویندوزی رابط‌های گرافیکی بین کاربر و کامپیوتر هستند و محیط ویندوز نمونه بارزی از یک رابط گرافیکی یا GUI است. شما در این صفحه می‌توانید کنترل‌هایی مانند دکمه‌ها، برچسب‌ها و ... به فرمتان اضافه

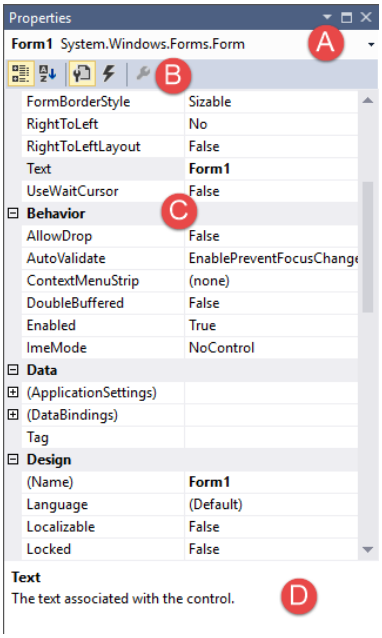
کنید. جزییات بیشتر در مورد فرم‌های ویندوزی و کنترل‌ها و برنامه‌نویسی شیء گرا در فصل فرم‌های ویندوزی آمده است. اما توصیه می‌شود ابتدا مبانی برنامه‌نویسی را مطالعه کنید.

مرورگر پروژه(Solution Explorer)

پروژه و فایل‌های مربوط به آن را نشان می‌دهد. یک Solution برنامه‌ای که توسط شما ساخته شده است را نشان می‌دهد. ممکن است این برنامه یک پروژه ساده یا یک پروژه چند بخشی باشد. اگر Solution Explorer در صفحه شما نمایش داده نمی‌شود می‌توانید از مسیر View > Other Windows > Solution Explorer و یا با کلیدهای میانبر Ctrl+Alt+L آنرا نمایان کنید. اگر چندین پروژه در حال اجرا هستند پروژه‌ای که با خط برجسته (Bold) نشان داده شده پروژه فعال می‌باشد و هنگام اجرای برنامه اجرا می‌شود. اگر بخواهید پروژه‌ای را که فعال نیست اجرا کنید، بر روی نام پروژه در Solution Explorer کلیک راست کنید و سپس گزینه Set as StartUp Project را انتخاب نمایید. Solution Explorer زیر یک Solution با ۲ پروژه را نشان می‌دهد. هر پروژه شامل فایل‌ها و فولدرهای مربوط به خود است.



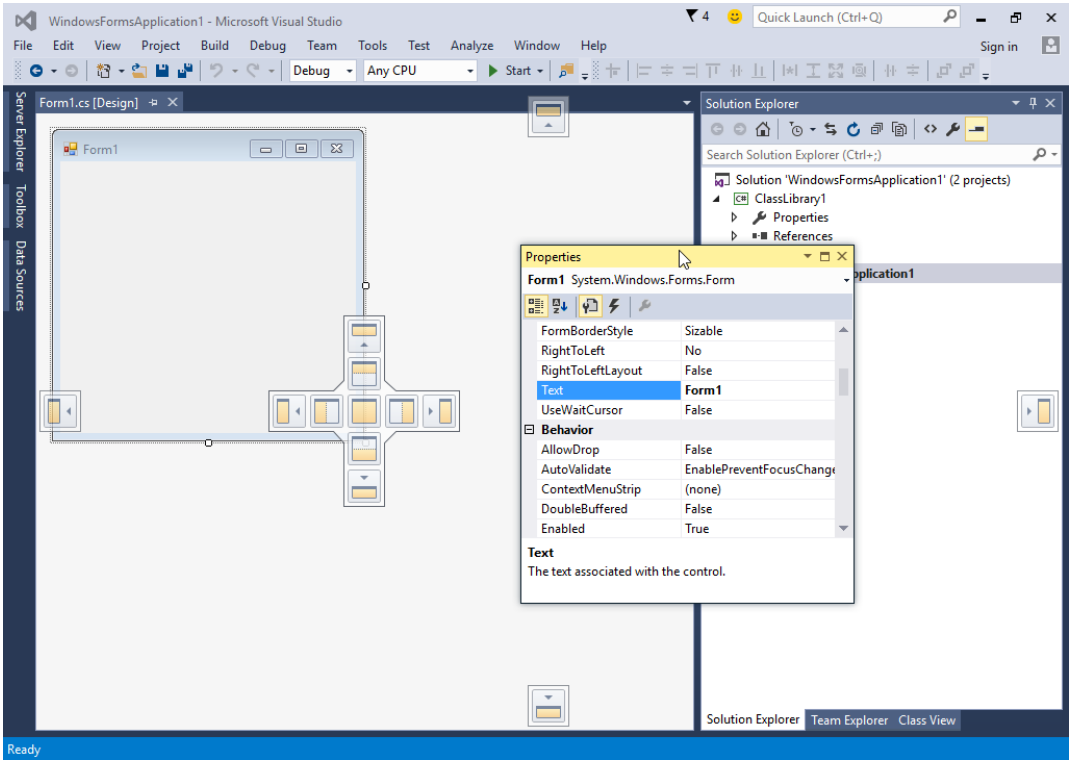
پنجره خواص(Properties)



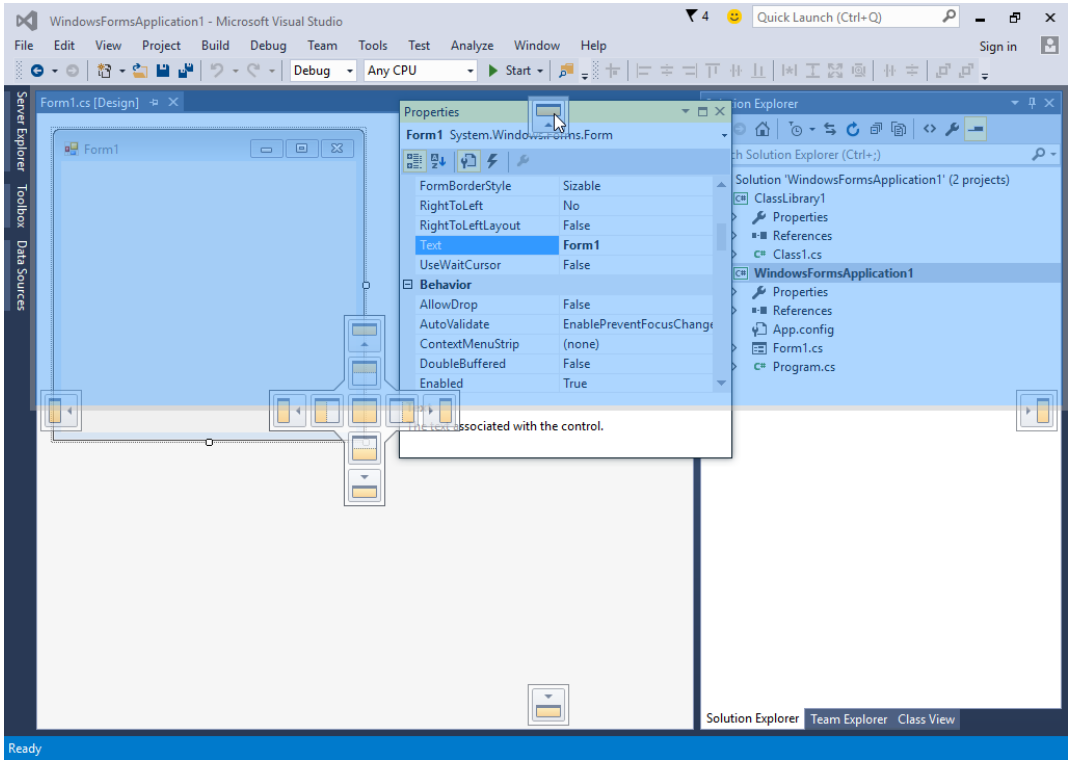
پنجره خواص (Properties)، خواص و رویدادهای مختلف هر آیتم انتخاب شده اعم از فرم، فایل، پروژه و کنترل را نشان می‌دهد. اگر این پنجره مخفی است می‌توانید از مسیر View > Other Windows > Properties Window یا کلید میانبر F4 آنرا ظاهر کنید. در مورد خواص در درس‌های آینده مفصل توضیح خواهیم داد. خاصیت‌ها، ویژگی‌ها و صفات اشیاء را نشان می‌دهند. 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 می‌باشد. یک رویداد (event) اتفاقی است که در شرایط خاصی پیش می‌آید. مانند وقتی که بر روی دکمه (button) کلیک و یا متنی را در داخل جعبه متن (text box)اصلاح می‌کنیم. کمبو باکس (combo box) شکل بالا که با حرف A نشان داده شده است به شما اجازه می‌دهد که شیء مورد نظرتان (دکمه، فرم و...) را که می‌خواهید 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 دکمه‌های مفیدی قرار دارند (B). برخی از این دکمه‌ها در شرایط خاصی فعال می‌شوند. دکمه اول خاصیت اشیاء را بر اساس دسته‌های مختلف و دومین دکمه خواص را بر اساس حروف الفبا مرتب می‌کند که پیشنهاد می‌کنیم از این دکمه برای دسترسی سریع به خاصیت مورد نظرتان استفاده کنید. سومین دکمه هم وقتی ظاهر می‌شود که یک کنترل یا یک فرم را در محیط طراحی انتخاب کنیم. این دکمه به شما اجازه دسترسی به خواص فرم و یا کنترل انتخاب شده را می‌دهد. چهارمین دکمه (که به شکل یک جرقه نمایش داده شده) رویدادهای فرم و یا کنترل انتخاب شده را نشان می‌دهد. در پایین شکل بالا توضیحات کوتاهی در مورد خاصیت‌ها و رویدادها نشان داده می‌شود. 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 می‌شود.

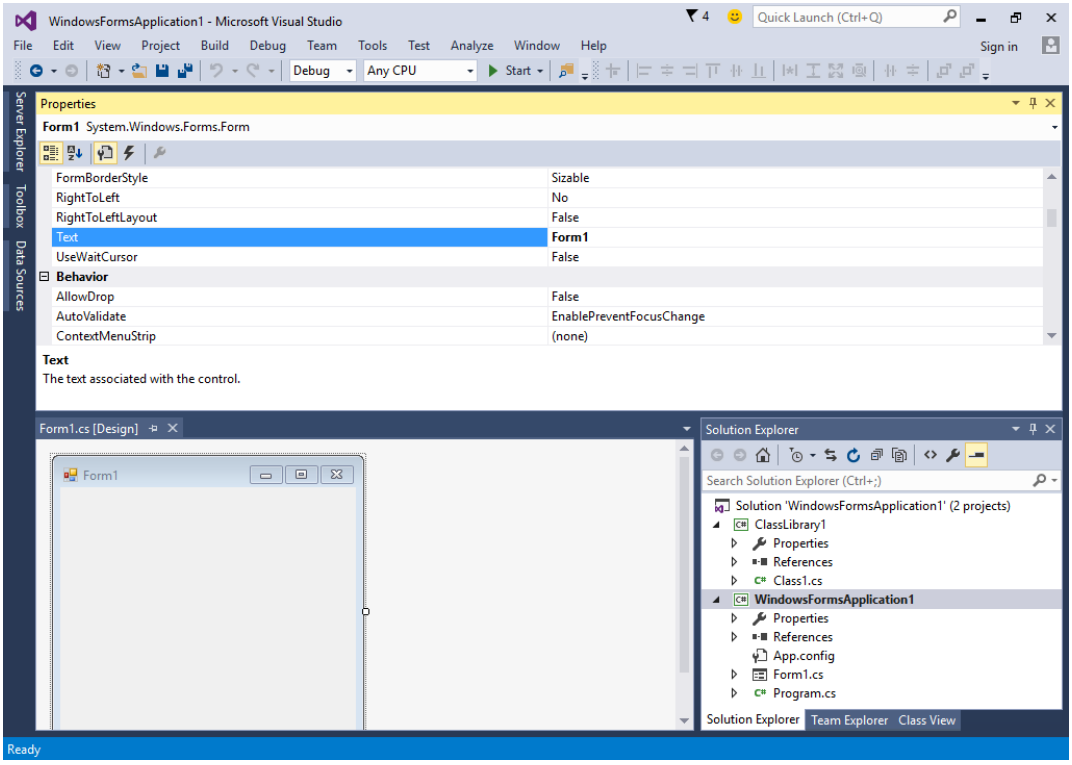
تغییر ظاهر ویژوال استودیو

اگر موقعیت پنجره‌ها و یا ظاهر برنامه ویژوال سی‌شارپ را دوست نداشته باشید، می‌توانید به دلخواه آن را تغییر دهید. برای این کار بر روی نوار عنوان (`title bar`) کلیک کرده و آنرا می‌کشید تا پنجره به شکل زیر به حالت شناور در آید:

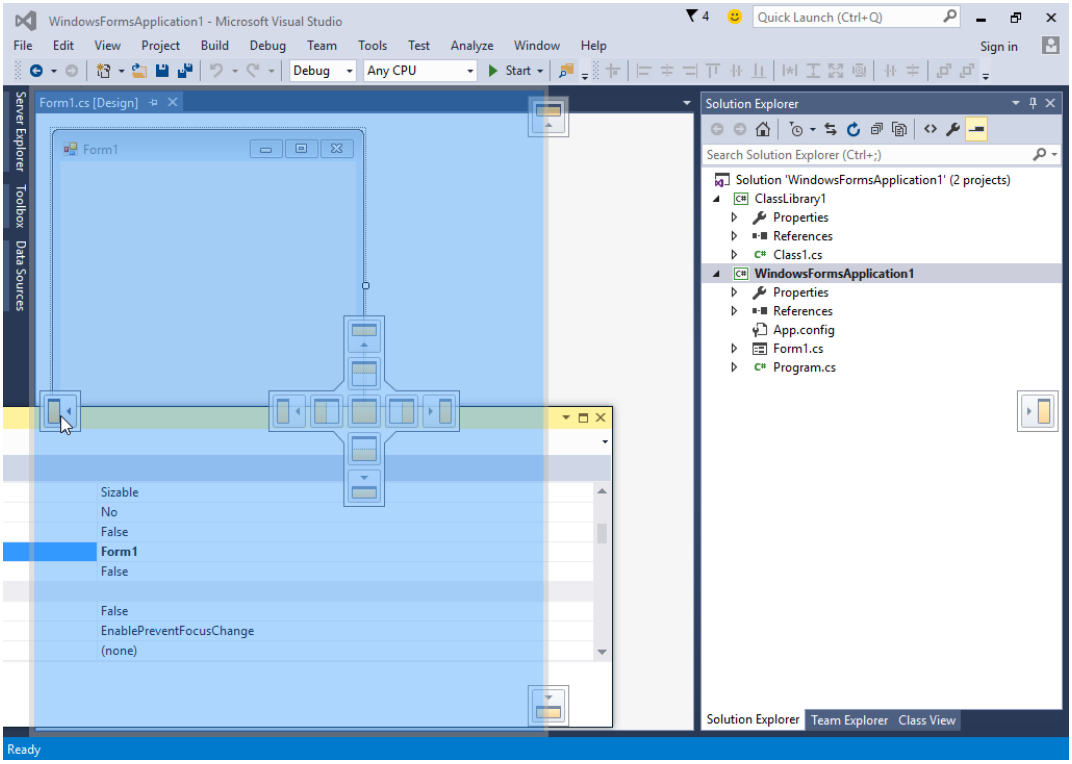


در حالی که هنوز بر روی پنجره کلیک کرده‌اید و آن را می‌کشید یک راهنما (فلشی با چهار جهت) ظاهر می‌شود و شما را در قرار دادن پنجره در محل دلخواه کمک می‌کند. به عنوان مثال شما می‌توانید پنجره را در بالاترین قسمت محیط برنامه قرار دهید. منطقه‌ای که پنجره قرار است در آنجا قرار بگیرد به رنگ آبی در می‌آید:

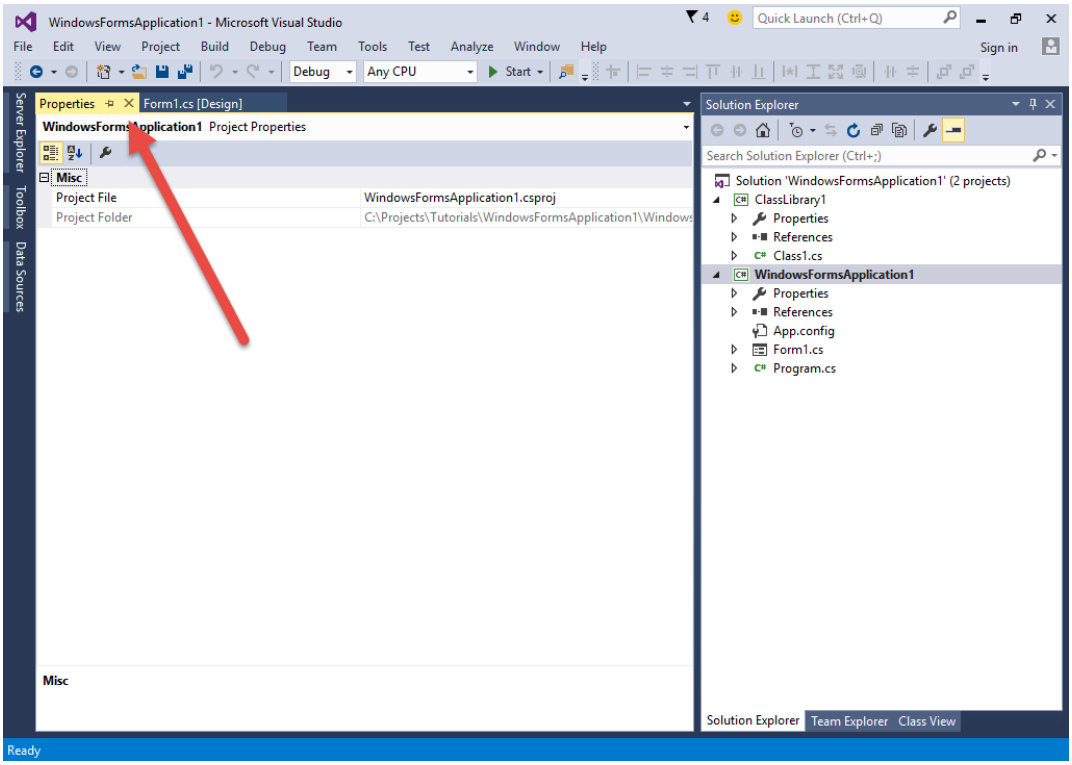

WindowsFormsApplication1 - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Start
Form1.cs [Design]
Form1
Properties
Form1 System.Windows.Forms.Form
FormBorderStyle Sizable
RightToLeft No
RightToLeftLayout False
Text Form1
UseWaitCursor False
Behavior
AllowDrop False
AutoValidate EnablePreventFocusChange
ContextMenuStrip (none)
DoubleBuffered False
Enabled True
Solution Explorer
Solution 'WindowsFormsApplication1' (2 projects)
ClassLibrary1
Properties
References
Class1.cs
WindowsFormsApplication1
Properties
References
App.config
Form1.cs
Program.cs
Solution Explorer Team Explorer Class View
Ready


WindowsFormsApplication1 - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Start
Properties
Form1 System.Windows.Forms.Form
FormBorderStyle Sizable
RightToLeft No
RightToLeftLayout False
Text Form1
UseWaitCursor False
Behavior
AllowDrop False
AutoValidate EnablePreventFocusChange
ContextMenuStrip (none)
Text
The text associated with the control.
Form1.cs [Design]
Form1
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'WindowsFormsApplication1' (2 projects)
ClassLibrary1
Properties
References
Class1.cs
WindowsFormsApplication1
Properties
References
App.config
Form1.cs
Program.cs
Solution Explorer Team Explorer Class View
Ready

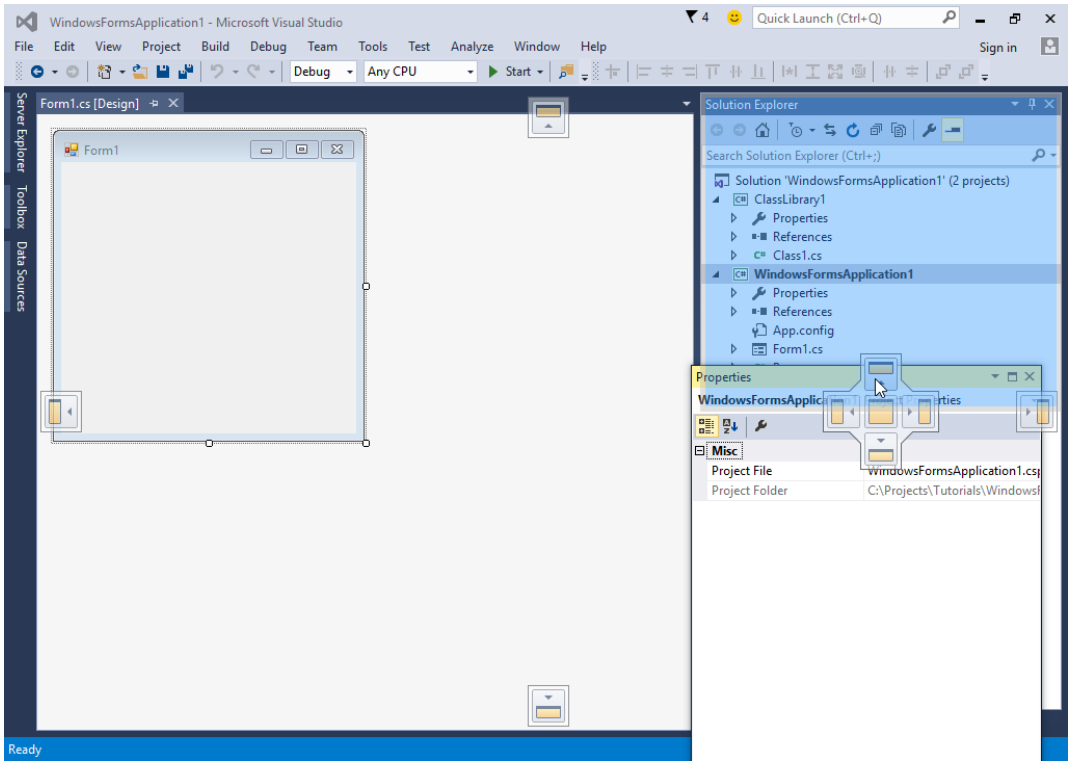
پنجره در قسمت بالای محیط قرار داده شده است. راهنمای صلیب شکل حاوی جعبه‌های مختلفی است که به شما اجازه می‌دهد پنجره انتخاب شده را در محل دلخواه محیط ویژوال استودیو قرار دهید. به عنوان مثال پنجره Properties را انتخاب و آنرا به چپ‌ترین قسمت صلیب در پنجره نمایش داده شده نزدیک و رها کنید، مشاهده می‌کنید که پنجره مذکور در سمت چپ پنجره Design View قرار می‌گیرد:



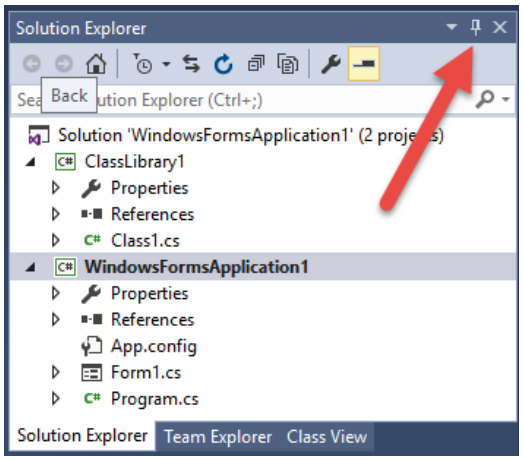
کشیدن پنجره به مرکز صلیب راهنما باعث ترکیب آن با پنجره مقصد می‌شود که در مثال بالا شما می‌توانید به عنوان یک تب به پنجره Properties دست پیدا کنید.



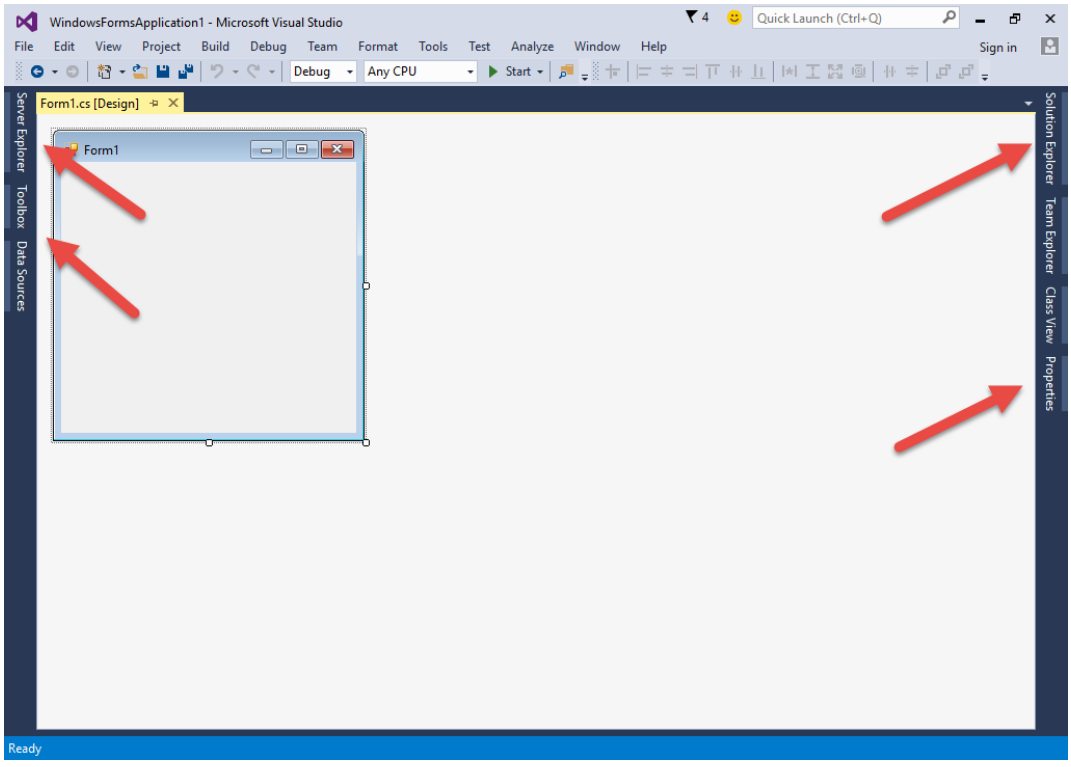
اگربه عنوان مثال پنجره Properties را روی پنجره Solution Explorer بکشید، یک صلیب راهنمای دیگر نشان داده می‌شود. با کشیدن پنجره به قسمت پایینی صلیب پنجره Properties، زیر پنجره Solution Explorer قرار خواهد گرفت.



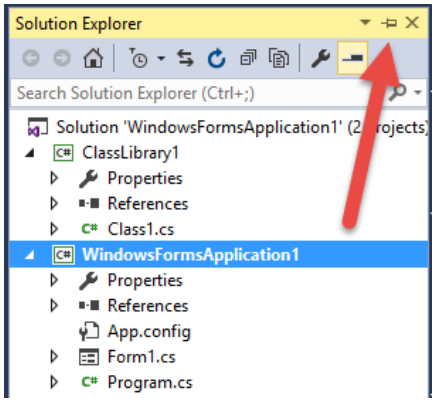
قسمتی از محیط برنامه که می‌خواهید پنجره در آنجا قرار بگیرد به رنگ آبی در می‌آید. ویژوال استودیو همچنین دارای خصوصیتی به نام autohide است که به صورت اتوماتیک پنجره‌ها را مخفی می‌کند. هر پنجره دارای یک آیکون سنجاق مانند نزدیک دکمه close می‌باشد.



بر روی این آیکون کلیک کنید تا قابلیت auto-hide فعال شود. برای دسترسی به هر یک از پنجره‌ها می‌توان با ماوس بر روی آنها توقف یا بر روی تب‌های کنار محیط ویژوال سی‌شارپ کلیک کرد.



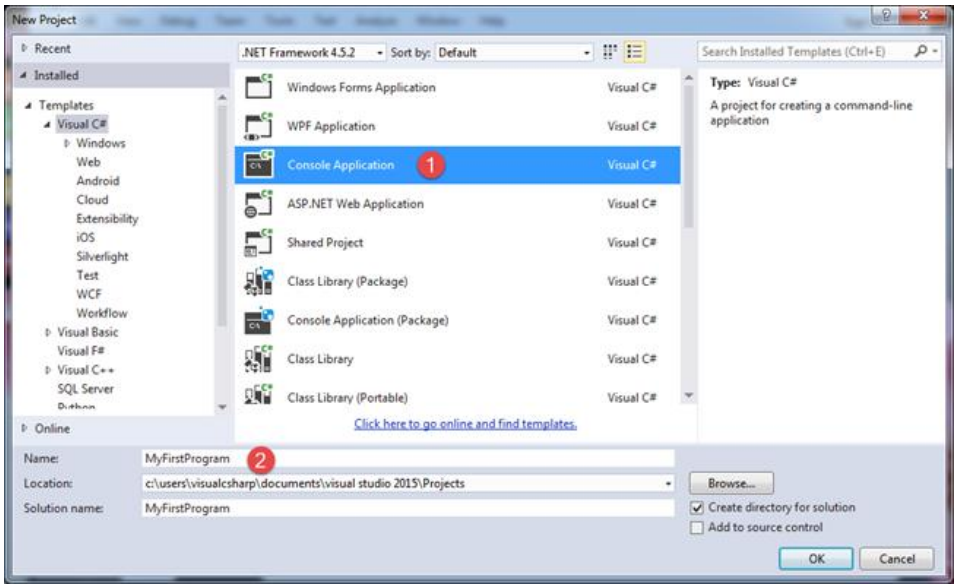
برای غیر فعال کردن این ویژگی در هر کدام از پنجره‌ها کافیست پنجره را انتخاب کرده و دوباره بر روی آیکون مورد نظر کلیک کنید.



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 می‌باشد.

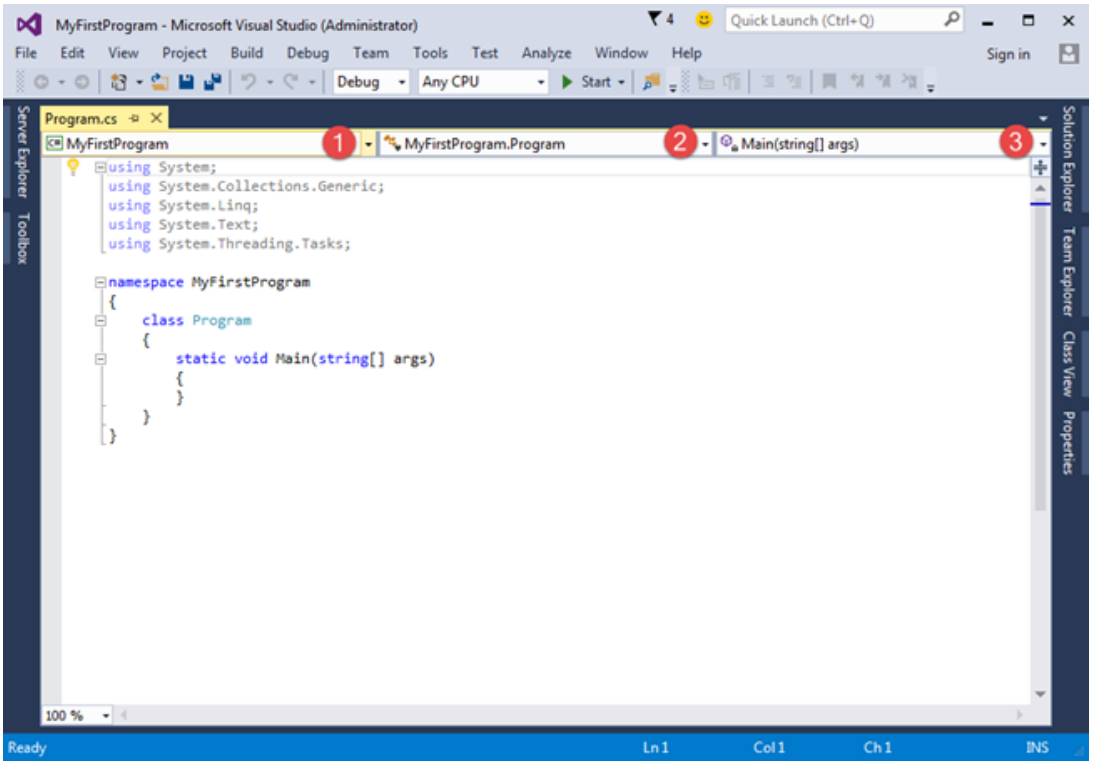
ساخت یک برنامه ساده

اجازه بدهید یک برنامه بسیار ساده به زبان سی‌شارپ بنویسیم. این برنامه یک پیغام را در محیط کنسول نمایش می‌دهد. در این درس می‌خواهم ساختار و دستور زبان یک برنامه ساده سی‌شارپ را توضیح دهم. برنامه Visual Studio Community را اجرا کنید. از مسیر File > New Project یک پروژه جدید ایجاد کنید. حال با یک صفحه مواجه می‌شوید که از شما می‌خواهد نام پروژه‌تان را انتخاب و آن را ایجاد کنید (شکل زیر):



گزینه Console Application را انتخاب کرده و نام پروژه‌تان را MyFirstProgram بگذارید. یک برنامه‌ای تحت Console Application داس در محیط ویندوز است و فاقد محیط گرافیکی می‌باشد. بهتر است برنامه خود را در محیط کنسول بنویسید تا بیشتر با مفهوم برنامه‌نویسی آشنا شوید. بعد از اینکه آموزش مبانی زبان سی‌شارپ به پایان رسید، برنامه نویسی در محیط ویندوز و بخش بصری آن را آموزش خواهیم داد.

بعد از فشردن دکمه OK، برنامه Visual Studio یک solution در یک فولدر موقتی ایجاد می‌کند. یک solution مجموعه‌ای از پروژه‌هاست، اما در بیشتر تمرینات شامل یک پروژه می‌باشد. فایل solution دارای پسوند sln می‌باشد و شامل جزییاتی در مورد پروژه‌ها و فایل‌های وابسته به آن می‌باشد. پروژه جدید همچنین حاوی یک فایل با پسوند csproj. می‌باشد که آن نیز شامل جزییاتی در مورد پروژه‌ها و فایل‌های وابسته به آن می‌باشد. حال می‌خواهیم شما را با محیط کد نویسی آشنا کنیم.



محیط کدنویسی جایی است که ما کدها را در آن تایپ می‌کنیم. کدها در محیط کدنویسی به صورت رنگی تایپ می‌شوند. در نتیجه تشخیص بخشهای مختلف کد را راحت می‌کند. منوی سمت چپ (شماره ۱) شامل نام پروژه‌ای که ایجاد کرده‌اید، منوی وسط (شماره ۲) شامل لیست کلاس‌ها، ساختارها، انواع شمارشی و منوی سمت راست (شماره ۳) شامل اعضای کلاس‌ها، ساختارها، انواع شمارشی و... می‌باشد. نگران اصطلاحاتی که به کار بردیم نباشید آنها را در فصول بعد توضیح خواهم داد. همه فایل‌های دارای کد در سی‌شارپ 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:

```
1  namespace MyFirstProgram
2  {
3      class Program
4      {
5          static void Main()
6          {
7              System.Console.WriteLine("Welcome to Visual C# Tutorials!");
8          }
9      }
10 }
```

ساختار یک برنامه در سی‌شارپ

مثال بالا ساده‌ترین برنامه‌ای است که شما می‌توانید در سی‌شارپ بنویسید. هدف از مثال بالا، نمایش یک پیغام در صفحه نمایش است. هر زبان برنامه‌نویسی 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 می‌کند. درباره فضای نام در درس‌های آینده توضیح خواهیم داد. در خط دوم آکولاد ({) نوشته شده است. آکولاد برای تعریف یک بلوک کد به کار می‌رود. سی‌شارپ یک زبان ساخت یافته است که شامل کدهای زیاد و ساختارهای فراوانی می‌باشد. هر آکولاد باز ({) در سی‌شارپ باید دارای یک آکولاد بسته (}) نیز باشد. همه کدهای نوشته شده از خط ۲ تا خط ۱۰ یک بلوک کد یا بدنه فضای نام است. در خط ۳ یک کلاس تعریف شده است. درباره کلاس‌ها در فصل‌های آینده توضیح خواهیم داد.

در مثال بالا کدهای شما باید در داخل یک کلاس نوشته شود. بدنه کلاس شامل کدهای نوشته شده از خط ۴ تا ۹ می‌باشد. خط ۵ متد Main() یا متد اصلی نامیده می‌شود. هر متد شامل یک سری کد است که وقتی اجرا می‌شوند که متد را صدا بزنیم. درباره 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 می‌شود. درباره متد Main() در فصول بعدی توضیح خواهیم داد. متد Main() و سایر متدها دارای آکولاد و کدهایی در داخل آنها می‌باشند و وقتی کدها اجرا می‌شوند که متدها را صدا بزنیم. هر خط کد در سی‌شارپ به یک سمیکالن (;) ختم می‌شود. اگر سمیکالن در آخر خط فراموش شود، برنامه با خطا مواجه می‌شود. مثالی از یک خط کد در سی‌شارپ 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 می‌دهد. از متد WriteLine() برای چاپ یک رشته استفاده می‌شود. 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 Console که آن نیز به نوبه خود در داخل فضای نام MyFirstProgram قرار دارد را نشان می‌دهد. توضیحات بیشتر در درس‌های آینده آمده است. سی‌شارپ فضای خالی و خطوط جدید را نادیده می‌گیرد. بنابراین شما می‌توانید همه برنامه را در یک خط بنویسید. اما اینکار خواندن و اشکال زدایی برنامه را مشکل می‌کند. یکی از خطاهای معمول در برنامه‌نویسی فراموش کردن سمیکالن در پایان هر خط کد است. به مثال زیر توجه کنید:

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

سی‌شارپ فضای خالی بالا را نادیده می‌گیرد و از کد بالا اشکال نمی‌گیرد. اما از کد زیر ایراد می‌گیرد:

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

به سمیکالن آخر خط اول توجه کنید. برنامه با خطای نحوی مواجه می‌شود، چون دو خط کد مربوط به یک برنامه هستند و شما فقط باید یک سمیکالن در آخر آن قرار دهید. همیشه به یاد داشته باشید که سی‌شارپ به بزرگی و کوچکی حروف حساس است. یعنی به طور مثال MAN و man در سی‌شارپ با هم فرق دارند. رشته‌ها و توضیحات از این قاعده مستثنی هستند که در درس‌های آینده توضیح خواهیم داد. مثلاً کدهای زیر با خطا مواجه می‌شوند و اجرا نمی‌شوند:

```
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 می‌کند. اما کد زیر کاملاً بدون خطا است:

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

همیشه کدهای خود را در داخل آکولاد بنویسید.

```
{
   statement1;
}
```

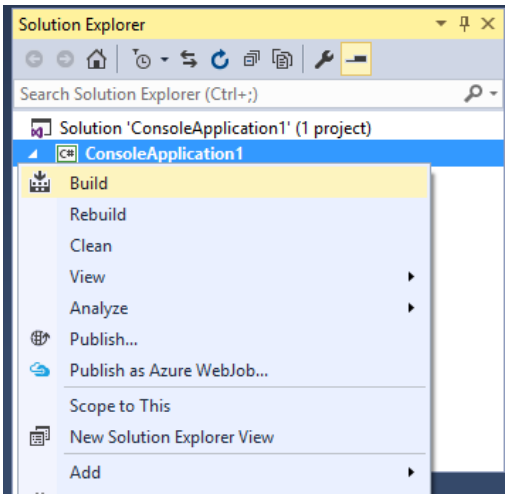
این کار باعث می‌شود که کدنویسی شما بهتر به چشم بیاید و تشخیص خطاها راحت تر باشد. یکی از ویژگی‌های مهم سی‌شارپ نشان دادن کدها به صورت تو رفتگی است. بدین معنی که کدها را به صورت تو رفتگی از هم تفکیک می‌کند و این در خوانایی برنامه بسیار مؤثر است.

ذخیره پروژه و برنامه

برای ذخیره پروژه و برنامه می‌توانید به مسیر File > Save All بروید یا از کلیدهای میانبر Ctrl+Shift+S استفاده کنید. همچنین می‌توانید از قسمت Toolbar بر روی شکل  کلیک کنید. برای ذخیره یک فایل ساده می‌توانید به مسیر File > Save (FileName) رفته یا از کلیدهای میانبر Ctrl+S استفاده کنید. همچنین می‌توانید از قسمت Toolbar بر روی شکل  کلیک کنید. برای باز کردن یک پروژه یا برنامه از منوی File گزینه Open را انتخاب و یا بر روی آیکون  در toolbar کلیک کنید. سپس به محلی که پروژه در آنجا ذخیره شده رفته و فایلی با پسوند sln یا پروژه‌ای با پسوند csproj. را باز کنید.

کامپایل برنامه

قبلاً ذکر شد که کدهای ما قبل از اینکه آنها را اجرا کنیم، ابتدا به زبان میانی ترجمه می‌شوند. برای کامپایل برنامه از منوی Debug گزینه Build Solution را انتخاب کنید یا دکمه F6 را بر روی صفحه کلید فشاردهید. این کار همه پروژه‌های داخل solution را کامپایل می‌کند. برای کامپایل یک قسمت از solution به Solution Explorer رفته و بر روی آن قسمت راست کلیک کرده و از منوی باز شده گزینه build را انتخاب کنید. مانند شکل زیر:



اجرای برنامه

وقتی ما برنامه مان را اجرا می‌کنیم سی‌شارپ به صورت اتوماتیک کدهای ما را به زبان میانی کامپایل می‌کند. 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 می‌گیرد. با اجرای برنامه در حالت Non-Debug سریعاً برنامه اجرا می‌شود و شما با زدن یک دکمه از برنامه خارج می‌شوید. در حالت پیش فرض حالت Non-Debug مخفی است و برای استفاده از آن می‌توان از منوی Debug گزینه Start Without Debuging را انتخاب کرد یا از دکمه‌های 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 جزء خروجی به حساب نمی‌آید و فقط نشان دهنده آن است که برنامه در حالت Non-Debug اجرا شده است و شما می‌توانید با زدن یک کلید از برنامه خارج شوید. دسترسی به حالت Debug Mode آسان تر است و به صورت پیش‌فرض برنامه‌ها در این حالت اجرا می‌شوند. از این حالت برای رفع خطاها و اشکال زدایی برنامه‌ها استفاده می‌شود که در درس‌های آینده توضیح خواهیم داد.

شما همچنین می‌توانید از Break Points و قسمت Help برنامه در مواقعی که با خطا مواجه می‌شوید استفاده کنید. برای اجرای برنامه با حالت Debug Mode می‌توانید از منوی Debug گزینه Start Debugging را انتخاب کرده و یا دکمه F5 را فشار دهید. همچنین می‌توانید بر روی شکل 

در toolbar کلیک کنید. اگر از حالت Debug Mode استفاده کنید برنامه نمایش داده شده و فوراً ناپدید می‌شود. برای جلوگیری از این اتفاق شما می‌توانید 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 می‌کنیم. مشاهده می‌کنید که برنامه متوقف شده و از شما در خواست ورودی می‌کند، به سادگی و با زدن دکمه Enter از برنامه خارج شوید. من از حالت Non-Debug به این علت استفاده کرده‌ام 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 استثناءها رسیدیم از حالت Debug استفاده می‌کنیم.

وارد کردن فضای نام در برنامه

فضای نام (Namespace) در برگیرنده کدهایی است که شما در برنامه‌تان از آنها استفاده می‌کنید. در برنامه فوق ما یک فضای نام در برنامه مان با نام MyFirstProgram داریم، اما دات‌نت دارای هزاران فضای نام می‌باشد. یکی از این فضاهای نامی، فضای نام System است که شامل کدهایی است که در یک برنامه ابتدایی C# به کار می‌روند. کلاس Console که ما از آن در برنامه بالا استفاده کردیم در این فضای نام قرار دارد.

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

اینکه قبل از استفاده از هر کلاس ابتدا فضای نام آن را مانند کد بالا بنویسیم کمی خسته کننده است. خوشبختانه دات‌نت به ما اجازه می‌دهد که برای جلوگیری از تکرار مکررات، فضاهای نامی را که قرار است در برنامه استفاده کنیم با استفاده از دستور using در ابتدای برنامه وارد نماییم:

```
using namespace;
```

دستور بالا نحوه وارد کردن یک فضای نام در برنامه را نشان می‌دهد. در نتیجه به جای آنکه به صورت زیر ابتدا نام فضای نام و سپس نام کلاس را بنویسیم:

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

می‌توانیم فضای نام را با دستوری که ذکر شد وارد برنامه کرده و کد بالا را به صورت خلاصه شده زیر بنویسیم:

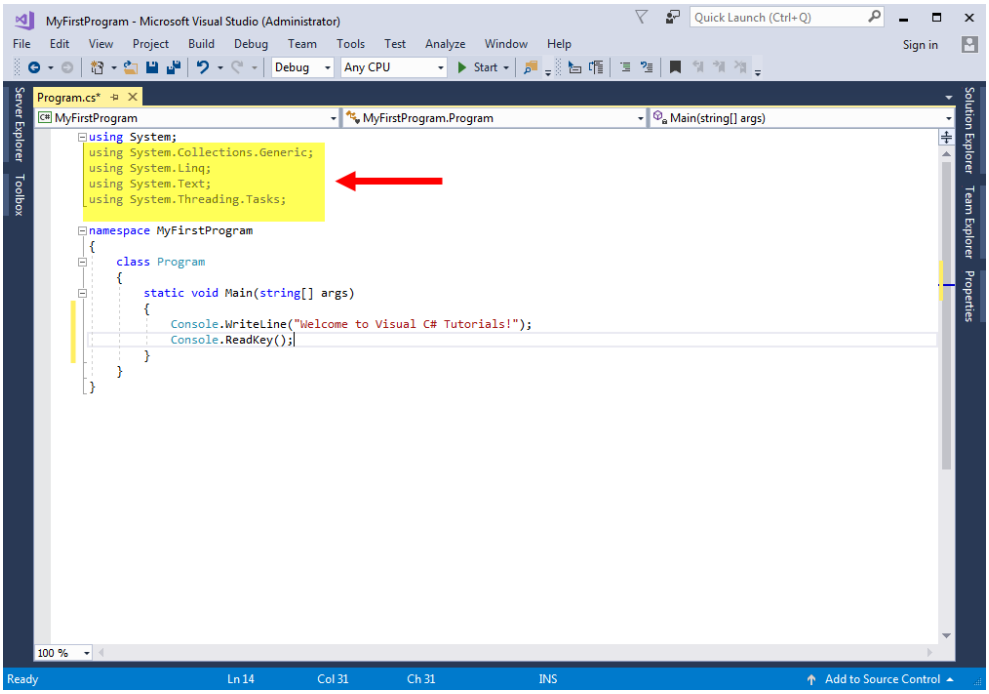
```
Console.WriteLine("Hello World!");
```

دستورات using که باعث وارد شدن فضاهای نامی به برنامه می‌شوند عموماً در ابتدای برنامه و قبل از همه کدها نوشته می‌شوند، پس برنامه‌ی این درس را می‌توان به صورت زیر نوشت:

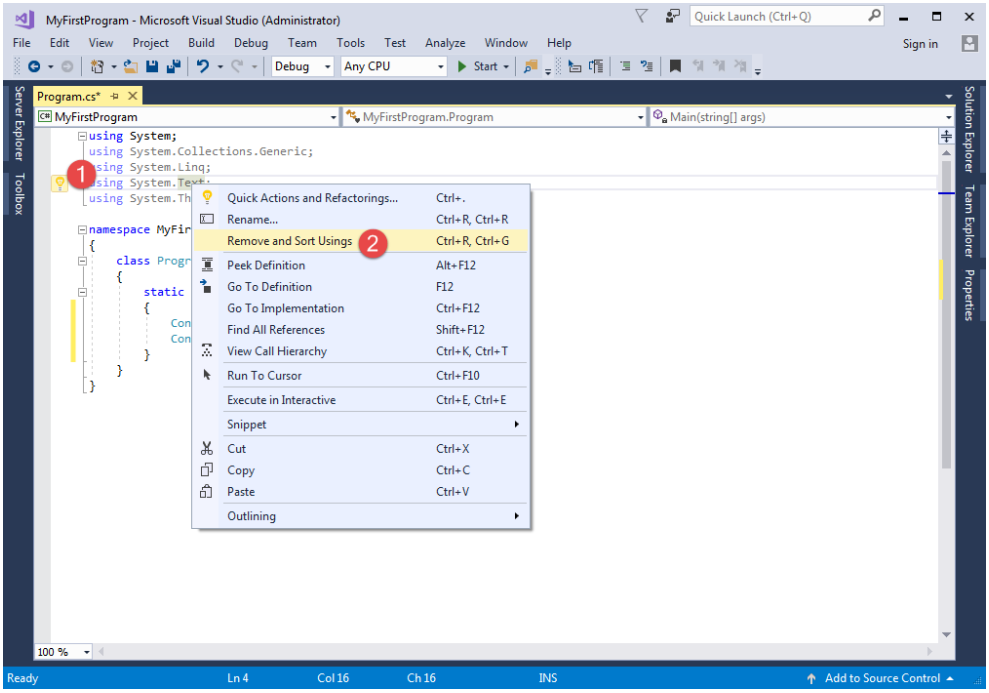
```
using System;

namespace MyFirstProgram
{
```

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

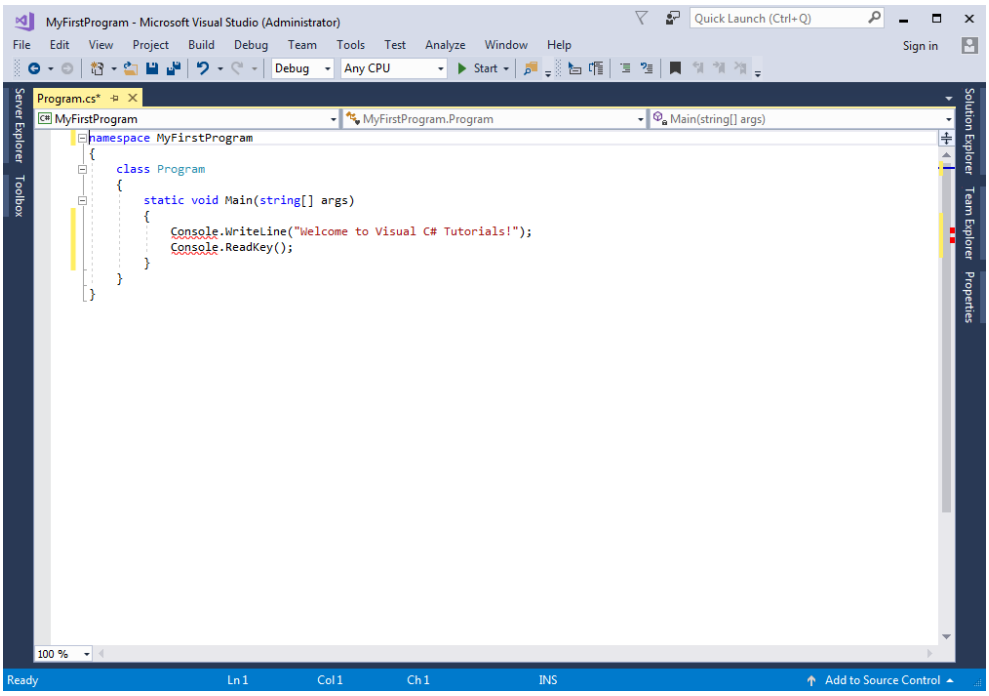
هنگامی که یک برنامه در ویژوال استودیو ایجاد می کنید، اگر وجود برخی از فضاهای نام الزامی نباشد، ویژوال استودیو ۲۰۱۷ آنها را به صورت کم رنگ نمایش می دهد، و شما می توانید بدون هیچ مشکلی این فضاهای نام را پاک کنید:



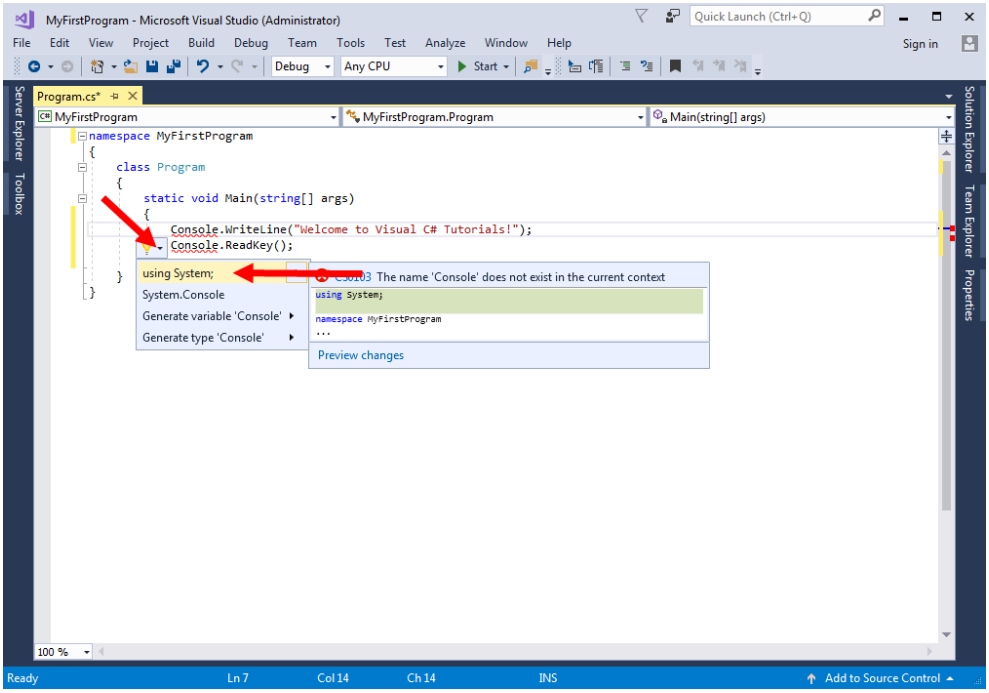
در نسخه های قبلی ویژوال استودیو هم برای پاک کردن فضاهای نام غیر قابل استفاده، ابتدا بر روی یکی از آنها راست کلیک کرده و سپس بر روی Remove and Sort Usings کلیک کنید:



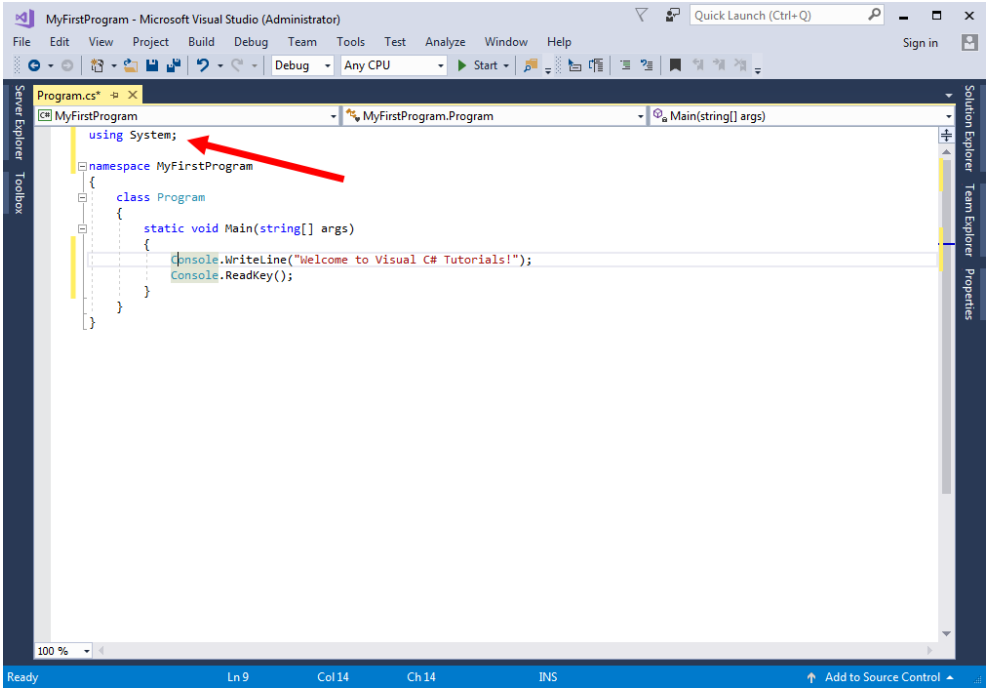
اگر از کلاسی استفاده کنید که از قبل فضای نام مربوط به آن را وارد برنامه نکرده باشید در زیر آن کلاس خط قرمز کشیده می شود :



برای رفع این مشکل، اگر از قبل نام فضای مربوطه را بلد باشید که باید آن را در قسمت فضای نام وارد کنید. در غیر اینصورت، بر روی نام کلاس با ماوس کمی مکث کنید تا یک پنجره popup ظاهر شده و آن را به شما معرفی کند :



در این صورت با کلیک بر روی آن، ویژوال استودیو به طور خودکار فضای نام را وارد برنامه می کند :



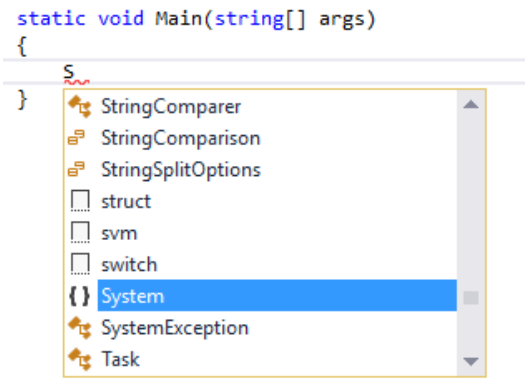
در مورد فضای نام در درس های آینده بیشتر توضیح می دهیم. حال که با خصوصیات و ساختار اولیه سی شارپ آشنا شدید در درسهای آینده مطالب بیشتری از این زبان برنامه نویسی قدرتمند خواهید آموخت.

استفاده از IntelliSense

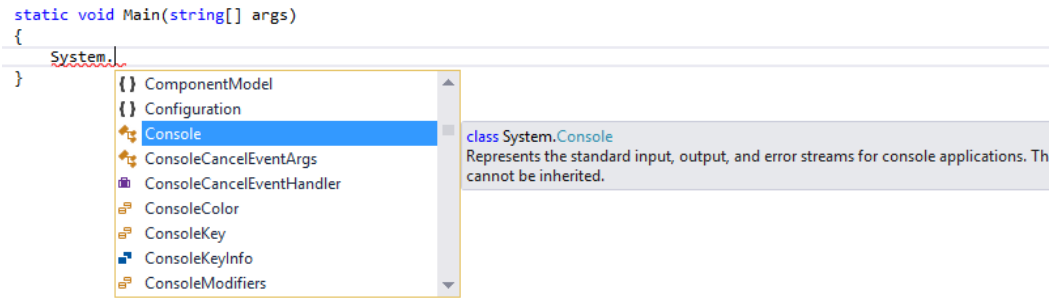
شاید یکی از ویژگی‌های مهم Visual Studio، اینتلی سنس باشد. IntelliSense ما را قادر می‌سازد که به سرعت به کلاس‌ها، متدها و...دسترسی پیدا کنیم. وقتی که شما در محیط کدنویسی حرفی را تایپ کنید IntelliSense فوراً فعال می‌شود. کد زیر را در داخل متد Main() بنویسید.

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

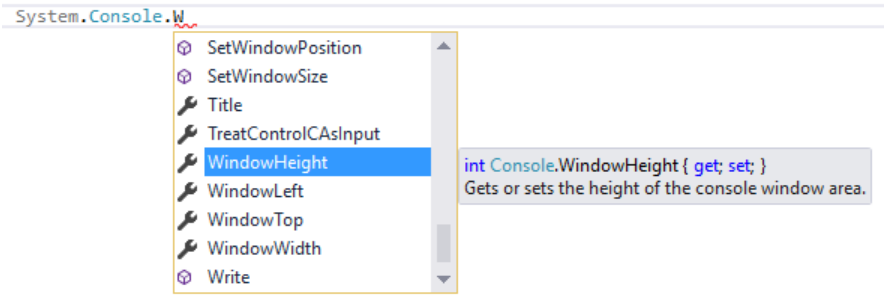
اولین حرف را تایپ کنید تا IntelliSense فعال شود.



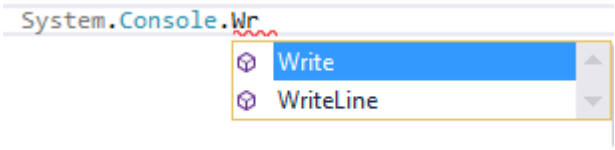
IntelliSense لیستی از کلمات به شما پیشنهاد می‌دهد که بیشترین تشابه را با نوشته شما دارند. شما می‌توانید با زدن دکمه Tab گزینه مورد نظرتان را انتخاب کنید. با تایپ نقطه (.) شما با لیست پیشنهادی دیگری مواجه می‌شوید.



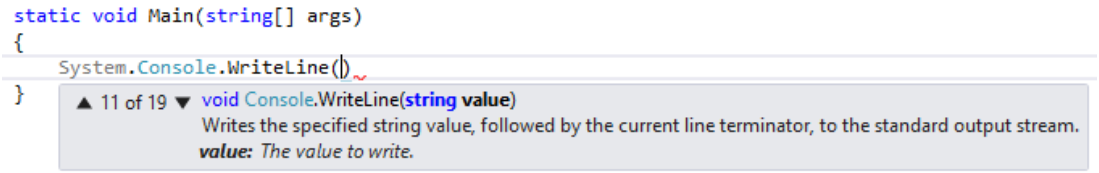
اگر بر روی گزینه‌ای که می‌خواهید انتخاب کنید لحظه‌ای مکث کنید، توضیحی در رابطه با آن مشاهده خواهید کرد، مانند شکل بالا. هر چه که به پایان کد نزدیک می‌شوید لیست پیشنهادی محدود تر می‌شود. برای مثال با تایپ حرف W، IntelliSense فقط کلماتی که با حرف W شروع می شوند، را نمایش می‌دهد.



با تایپ حرف‌های بیشتر لیست محدودتر شده و فقط دو کلمه را نشان می‌دهد.



اگر IntelliSense نتواند چیزی را که شما تایپ کرده‌اید پیدا کند، هیچ چیزی را نمایش نمی‌دهد. برای ظاهر کردن IntelliSense کافیست دکمه ترکیبی Ctrl+Space را فشار دهید. برای انتخاب یکی از متدهایی که دارای چند حالت هستند، می‌توان با استفاده از دکمه‌های مکان نما (بالا و پایین) یکی از حالت‌ها را انتخاب کرد. مثلاً متد Writeline() همانطور که در شکل زیر مشاهده می‌کنید دارای ۱۹ حالت نمایش پیغام در صفحه است.

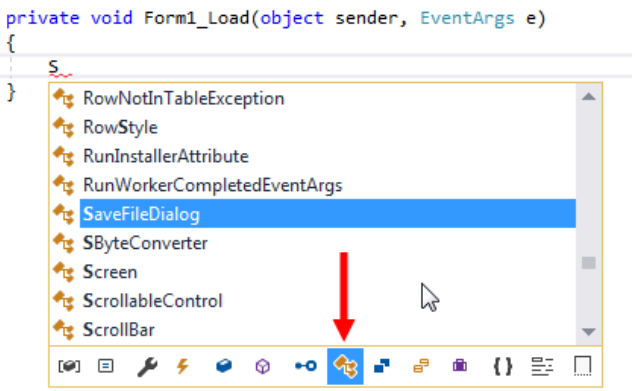


IntelliSense به طور هوشمند کدهایی را به شما پیشنهاد می‌دهد و در نتیجه زمان نوشتن کد را کاهش می‌دهد. در ویژوال استودیو هر جزء دارای یک آیکون منحصر به فرد می باشد. در زیر لیست آیکون های ویزوال استودیو آمده است:

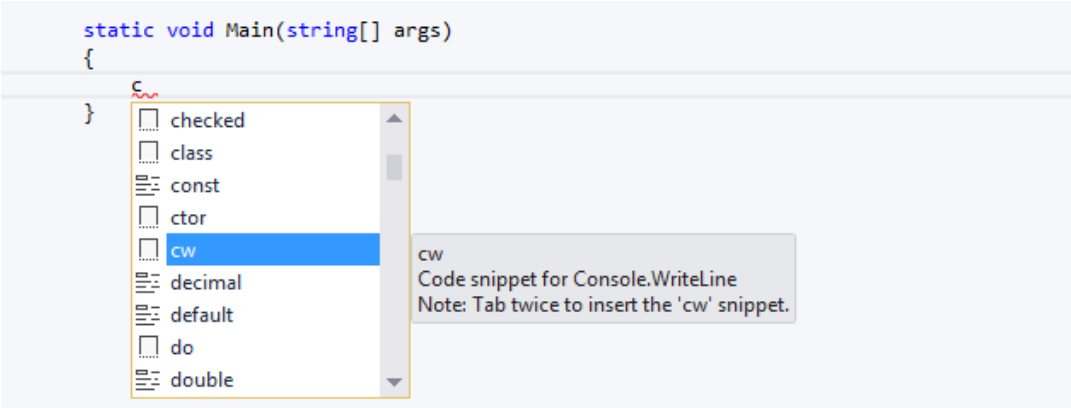
آیکون	مربوط به
	پارامترها و متغیرهای محلی (Locals and Parameters)
	ثابت (Constant)
	خاصیت (Property)
	رویداد (Event)
	فیلد (Field)

	متد (Method)
	رابط (Interface)
	کلاس (Class)
	ساختار (Structure)
	نوع شمارشی (Enum)
	نماینده (Delegate)
	فضای نام (Namespace)
	کلمه کلیدی (Keyword)
	کد کوتاه (Code Snippet)

نگران اسامی ذکر شده در جدول بالا نباشید. آنها را در درس های آینده توضیح خواهیم داد. یکی از قابلیت های جدید که در ویژوال استودیو ۲۰۱۷ اضافه شده است، مرتب کردن لیست IntelliSense می باشد. فرض کنید که شما می خواهید همه کلاس هایی دارای حرف S هستند را در لیست داشته باشید. برای این کار کافیست بر روی آیکون کلاس در IntelliSense کلیک کنید:



همانطور که در شکل بالا مشاهده می کنید همه کلاس هایی که دارای حرف S هستند، لیست می شوند. در زیر یکی دیگر از امکانات ویژوال استودیو که باعث راحتی در کدنویسی می شوند، Code Snippet ها هستند. Code Snippet ها در واقعا مخفف برخی کلمات یا عبارات در ویژوال استودیو هستند. مثلا به جای نوشتن عبارت System.Console.WriteLine(); می توانید cw را نوشته و سپس دو بار دکمه Tab را بزنید تا ویژوال استودیو عبارت مذکور را برای شما کامل کند:



لیست Code Snippet های ویژوال استودیو در لینک زیر آمده است:

```
http://www.w3-farsi.com/?p=2973
```

رفع خطاها

بیشتر اوقات هنگام برنامه‌نویسی با خطا مواجه می‌شویم. تقریباً همه برنامه‌هایی که امروزه می‌بینید حداقل از داشتن یک خطا رنج می‌برند. خطاها می‌توانند برنامه شما را با مشکل مواجه کنند. در سی‌شارپ سه نوع خطا وجود دارد:

خطای کامپایلری

این نوع خطا از اجرای برنامه شما جلوگیری می‌کند. این خطاها شامل خطای دستور زبان می‌باشد. این بدین معنی است که شما قواعد کد نویسی را رعایت نکرده‌اید. یکی دیگر از موارد وقوع این خطا هنگامی است که شما از چیزی استفاده می‌کنید 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 می‌آید. رفع این نوع خطاها بسیار سخت است چون شما برای یافتن آنها باید کد را تست کنید. نمونه‌ای از یک خطای منطقی برنامه‌ای است که دو عدد را جمع می‌کند ولی حاصل تفریق دو عدد را نشان می‌دهد. در این حالت ممکن است برنامه نویس علامت ریاضی را اشتباه تایپ کرده باشد.

استثناء

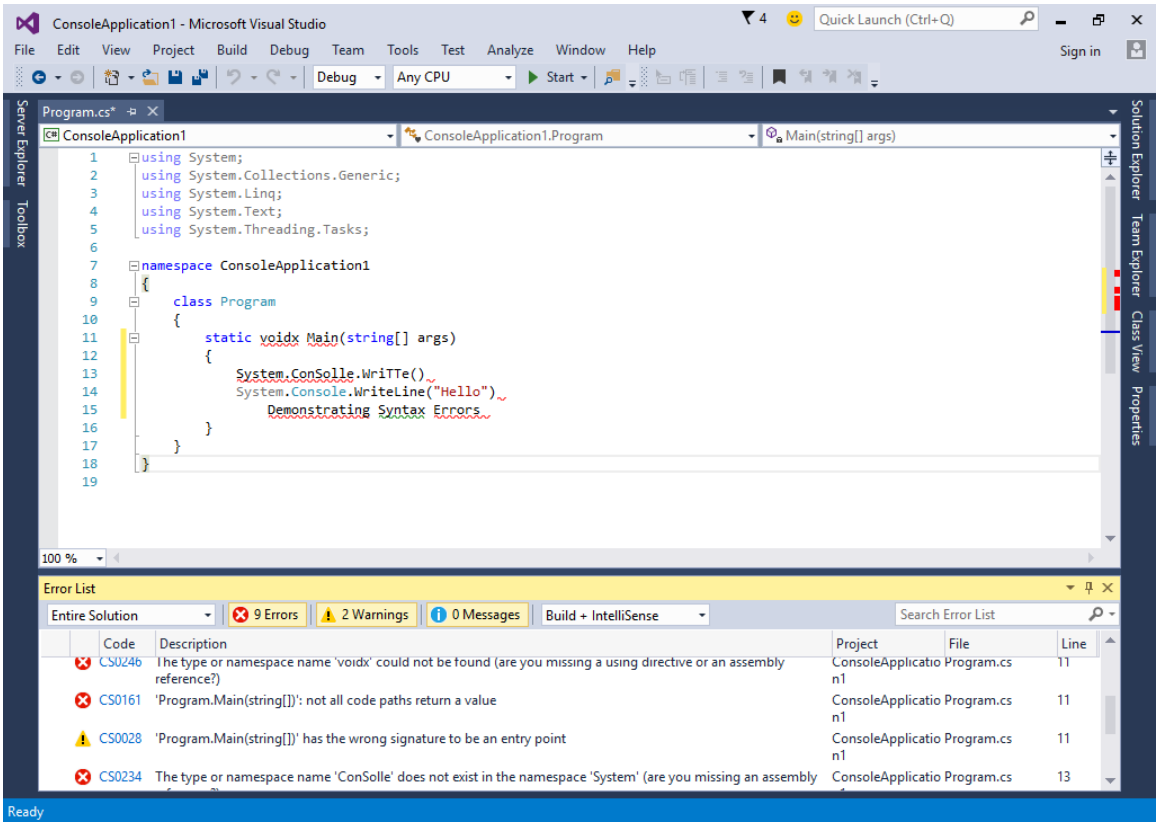
این نوع خطاها هنگامی رخ می‌دهند که برنامه در حال اجراست. این خطا هنگامی روی می‌دهد که کاربر یک ورودی نامعتبر به برنامه بدهد و برنامه نتواند آن را پردازش کند. ویژوال استودیو و ویژوال سی‌شارپ دارای ابزارهایی برای پیدا کردن و برطرف کردن خطاها هستند. وقتی در محیط کدنویسی در حال تایپ کد هستیم یکی از ویژگی‌های ویژوال استودیو تشخیص خطاهای ممکن قبل از اجرای برنامه است. زیر کدهایی که دارای خطای کامپایلری هستند خط قرمز کشیده می‌شود.

```
namespace ConsoleApplication1
{
    class Program
    {
        static voidx Main(string[] args)
        {
            System.ConSolle.WriTTe()
            System.Console.WriteLine("Hello")
                Demonstrating Syntax Errors
        }
    }
}
```

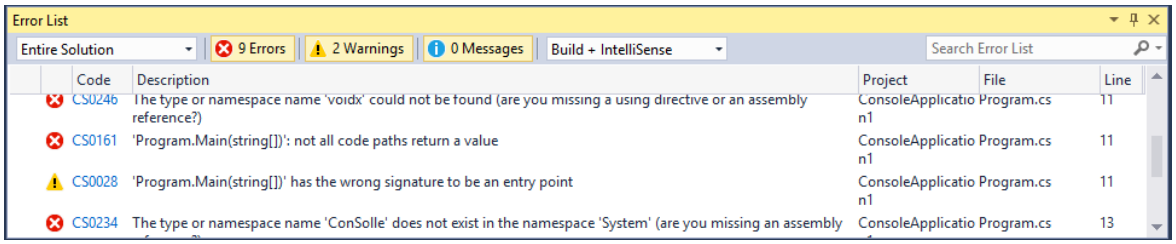
هنگامی که شما با ماوس روی این خطوط توقف کنید توضیحات خطا را مشاهده می‌کنید. شما ممکن است با خط سبز هم مواجه شوید که نشان دهنده اخطار در کد است ولی به شما اجازه اجرای برنامه را می‌دهند. به عنوان مثال ممکن است شما یک متغیر را تعریف کنید ولی در طول برنامه از آن استفاده نکنید (در درس‌های آینده توضیح خواهیم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

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int number;
        }
    }
}
```

در باره رفع خطاها در آینده توضیح بیشتری می‌دهیم. ErrorList (لیست خطاها) که در شکل زیر با فلش قرمز نشان داده شده است به شما امکان مشاهده خطاها، هشدارها و رفع آنها را می‌دهد. برای باز کردن Error List می‌توانید به مسیر View > Other Windows > Error List بروید.



همانطور که در شکل زیر مشاهده می‌کنید هرگاه برنامه شما با خطا مواجه شود لیست خطاها در `Error List` نمایش داده می‌شود.

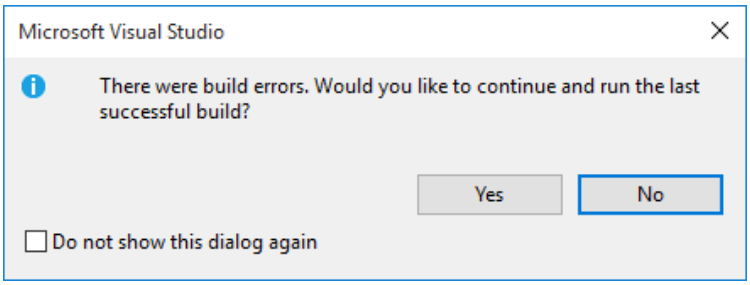


در شکل بالا تعدادی خطا همراه با راه حل رفع آنها در `Error List` نمایش داده شده است. `Error List` دارای چندین ستون است که به طور کامل جزییات خطاها را نمایش می‌دهند.

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 پروژه‌ای که دارای خطاست.

اگر برنامه شما دارای خطا باشد و آن را اجرا کنید با پنجره زیر روبه‌رو می‌شوید:



مربع کوچک داخل پنجره بالا را تیک بزنید، تا دفعات بعد که برنامه شما با خطا مواجه شد، دیگر این پنجره به عنوان هشدار نشان داده نشود. با کلیک بر روی دکمه Yes برنامه با وجود خطا نیز اجرا می‌شود. اما با کلیک بر روی دکمه NO اجرای برنامه متوقف می‌شود 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 می‌شود.

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 می‌شوید و در کل یک مثال است:

خطا	توضیح	راه حل
expected;	در پایان دستور علامت سیمیکان (;) قرار نداده‌اید	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 جزء دستورات سی‌شارپ 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	نوع یا فضای نام متغیر sample تعریف نشده است	باید یک کلاس یا فضای نام، به نام sample ایجاد کنید.

a using directive or an assembly reference?)		
'MyMethod()': not all code paths return a value	بدین معناست 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 نمی‌تواند 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 آنها در درس‌های آینده آمده است.

توضیحات

وقتی که کدی تایپ می‌کنید شاید بخواهید که متنی جهت یادآوری وظیفه آن کد به آن اضافه کنید. در سی‌شارپ (و بیشتر زبان‌های برنامه نویسی) می‌توان این کار را با استفاده از توضیحات انجام داد. توضیحات متونی هستند که توسط کامپایلر نادیده گرفته می‌شوند و به عنوان بخشی از کد محسوب نمی‌شوند. هدف اصلی از ایجاد توضیحات، بالا بردن خوانایی و تشخیص نقش کدهای نوشته شده توسط شما، برای دیگران است. فرض کنید که می‌خواهید در مورد یک کد خاص، توضیح بدهید، می‌توانید توضیحات را در بالای کد یا کنار آن بنویسید. از توضیحات برای مستند سازی برنامه هم استفاده می‌شود. 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 }
```

```
Hello World!
```

در کد بالا، خط ۷ یک توضیح درباره خط ۸ است که به کاربر اعلام می‌کند که وظیفه خط ۸ چیست؟ با اجرای کد بالا فقط جمله Hello World چاپ شده و خط ۷ در خروجی نمایش داده نمی‌شود چون کامپایلر توضیحات را نادیده می‌گیرد. توضیحات بر سه نوع‌اند:

```
// single line comment
```

```
/* multi
   line
```

```
   comment */
```

```
/// <summary>
/// This is XML comments
/// </summary>
```

توضیحات تک خطی همانگونه که از نامش پیداست، برای توضیحاتی در حد یک خط به کار می‌روند. این توضیحات با علامت // شروع می‌شوند و هر نوشته ای که در سمت راست آن قرار بگیرد جزء توضیحات به حساب می‌آید. این نوع توضیحات معمولاً در بالا یا کنار کد قرار می‌گیرند. اگر توضیح در باره یک کد به بیش از یک خط نیاز باشد از توضیحات چند خطی استفاده می‌شود. توضیحات چند خطی با /* شروع و با */ پایان می‌یابند. هر نوشته ای که بین این دو علامت قرار بگیرد جزء توضیحات محسوب می‌شود. نوع دیگری از توضیحات، توضیحات XML نامیده می‌شوند. این نوع با سه اسلش (///) نشان داده می‌شوند. از این نوع برای مستند سازی برنامه استفاده می‌شود و در درس‌های آینده در مورد آنها توضیح خواهیم داد.

کاراکترهای کنترلی

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

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

```
Hello
World!
```

مشاهده کردید که کامپایلر بعد از مواجهه با کاراکتر کنترلی \n نشانگر ماوس را به خط بعد برده و بقیه رشته را در خط بعد نمایش می‌دهد. متد WriteLine() هم مانند کاراکتر کنترلی \n یک خط جدید ایجاد می‌کند، البته بدین صورت که در انتهای رشته یک کاراکتر کنترلی \n اضافه می‌کند:

```
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 ماوس را در همان خط نگه می‌دارد و خط جدید ایجاد نمی‌کند.

جدول زیر لیست کاراکترهای کنترلی و کارکرد آنها را نشان می‌دهد:

کاراکتر کنترلی	عملکرد	کاراکتر کنترلی	عملکرد
\f	Form Feed	\'	چاپ کوتیشن

\n	خط جدید	\"	چاپ دابل کوتیشن
\r	سر سطر رفتن	\\	چاپ بک اسلش
\t	حرکت به صورت افقی	\0	چاپ فضای خالی
\v	حرکت به صورت عمودی	\a	صدای بیپ
\u	چاپ کاراکتر یونیکد	\b	حرکت به عقب

ما برای استفاده از کاراکترهای کنترلی از بک اسلش (\) استفاده می‌کنیم. از آنجاییکه علامت \ معنای خاصی به رشته‌ها می‌دهد، 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 رشته‌ها استفاده می‌کنیم، برای چاپ آن از "\ استفاده می‌کنیم:

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

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

همچنین برای چاپ کوتیشن (') از '\ استفاده می‌کنیم:

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

```
The programmer's heaven.
```

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

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

```
Left    Right
```

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

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

```
Kitten
```

مثلاً در مثال بالا کاراکتر K بعد از کاراکتر کنترلی ۴\ آمده است. کاراکتر کنترلی، حرف K را به ابتدای سطر برده و جایگزین حرف M می‌کند. برای چاپ کاراکترهای یونیکد می‌توان از u\ استفاده کرد. برای استفاده از u\، مقدار در مبنای ۱۶ کاراکتر را درست بعد از علامت u\ قرار می‌دهیم. 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 ۱۶ 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 می‌دهد. بیشترین خطا زمانی اتفاق می‌افتد که برنامه نویس برای چاپ اسلش (\) از \\ استفاده می‌کند.

علامت @

علامت @ به شما اجازه می‌دهد که کاراکترهای کنترلی را رد کرده و رشته‌ای خواناتر و طبیعی تر ایجاد کنید. وقتی از کاراکترهای کنترلی در یک رشته استفاده می‌شود، ممکن است برای تایپ مثلاً 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 می‌کند که شما می‌خواهید یک کاراکتر کنترلی را تایپ کنید، کاراکتر بعد از علامت \ را پردازش می‌کند و چون کاراکتر کنترلی وجود ندارد خطا به وجود می‌آید. 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 عمداً از بک اسلش (\) برای اثبات گفته بالا استفاده کرده‌ایم. کامپایلر خطا ایجاد می‌کند و به شما می‌گوید که کاراکتر کنترلی d\ قابل تشخیص نیست، چون چنین کاراکتر کنترلی وجود ندارد. زمانی وضعیت بدتر خواهد شد که کاراکتر بعد از بک اسلش کاراکتری باشد که هم جزء یک کلمه باشد و هم جزء 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 نمی‌خواهید از علامت بک اسلش برای نشان دادن یک کاراکتر 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 @ معمولاً زمانی استفاده می‌شود که شما بخواهید مسیر یک دایرکتوری را به عنوان رشته داشته باشید. چون دایرکتوری‌ها دارای تعداد زیادی بک اسلش هستند و طبیعتاً 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 می‌توانید 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 کاراکتر کنترلی در خروجی می‌شود.

استفاده از علامت @ برای نگهداری از قالب بندی رشته‌ها

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

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

یکی از راه‌های 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 قبلاً مشاهده کردید خط جدید ایجاد می‌کند و در مثال بالا باعث می‌شود که جمله به چند خط تقسیم شود. از علامت + هم برای ترکیب رشته‌ها استفاده می‌شود. 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 می‌خواهید رشته در خط بعد نمایش داده شود دکمه Enter را فشار دهید.

متغیرها

متغیر، مکانی از حافظه است که شما می‌توانید مقادیری را در آن ذخیره کنید. می‌توان آن را به عنوان یک ظرف تصور کرد که داده‌های خود را در آن قرار داده‌اید. محتویات این ظرف می‌تواند پاک شود یا تغییر کند. هر متغیر دارای یک نام نیز هست. که از طریق آن می‌توان متغیر را از دیگر متغیرها تشخیص داد و به مقدار آن دسترسی پیدا کرد. همچنین دارای یک مقدار می‌باشد که می‌تواند توسط کاربر انتخاب شده باشد یا نتیجه یک محاسبه باشد. مقدار متغیر می‌تواند تهی نیز باشد. متغیر دارای نوع نیز هست بدین معنی که نوع آن با نوع داده‌ای که در آن ذخیره می‌شود

یکی است. متغیر دارای عمر نیز هست که از روی آن می‌توان 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 می‌گوید که متغیر در چه جای برنامه برای شما قابل دسترسی است.

ما از متغیرها به عنوان یک انبار موقتی برای ذخیره داده استفاده می‌کنیم. هنگامی که یک برنامه ایجاد می‌کنیم احتیاج به یک مکان برای ذخیره داده، مقادیر یا داده‌هایی که توسط کاربر وارد می‌شوند داریم. این مکان همان متغیر است. برای این از کلمه متغیر استفاده می‌شود چون ما می‌توانیم بسته به نوع شرایط، هر جا که لازم باشد مقدار آن را تغییر دهیم. متغیرها موقتی هستند و فقط موقعی مورد استفاده قرار می‌گیرند که برنامه در حال اجراست و وقتی شما برنامه را می‌بندید محتویات متغیرها نیز پاک می‌شود. قبلاً ذکر شد که به وسیله نام متغیر می‌توان 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.
- نمی‌تواند شامل کاراکترهای غیرمجاز مانند #، ?، ^ و $ باشد.
- نمی‌توان از کلمات رزرو شده در سی‌شارپ برای نام متغیر استفاده کرد.
- نام متغیر نباید دارای فضای خالی (spaces) باشد.
- اسامی متغیرها نسبت به بزرگی و کوچکی حروف حساس هستند. در سی‌شارپ دو حرف مانند a و A دو کاراکتر مختلف به حساب می‌آیند.

دو متغیر با نام‌های myNumber و MyNumber دو متغیر مختلف محسوب می‌شوند چون یکی از آنها با حرف کوچک m و دیگری با حرف بزرگ M شروع می‌شود. شما نمی‌توانید 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 و نوع آن همان نوع داده‌ای است که در خود ذخیره می‌کند. معمول‌ترین انواع داده int، double، string، char، float، decimal می‌باشند. برای مثال شما برای قرار دادن یک عدد صحیح در متغیر باید از نوع int استفاده کنید.

انواع ساده

انواع ساده، انواعی از داده‌ها هستند که شامل اعداد، کاراکترها، رشته‌ها و مقادیر بولی می‌باشند. به انواع ساده انواع اصلی نیز گفته می‌شود چون از آنها برای ساخت انواع پیچیده تری مانند کلاس‌ها و ساختارها استفاده می‌شود. انواع ساده دارای مجموعه مشخصی از مقادیر هستند و محدوده خاصی از اعداد را در خود ذخیره می‌کنند. در جدول زیر انواع ساده و محدود آنها آمده است:

نوع	محدوده
sbyte	اعداد صحیح بین ۱۲۸- تا ۱۲۷
byte	اعداد صحیح بین ۰ تا ۲۵۵
short	اعداد صحیح بین ۳۲۷۶۸- تا ۳۲۷۶۷

ushort	اعداد صحیح بین ۰ تا ۶۵۵۳۵
int	اعداد صحیح بین ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷
uint	اعداد صحیح بین ۰ تا ۴۲۹۴۹۶۷۲۹۵
long	اعداد صحیح بین ۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۰۸- تا ۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۸۰۷
ulong	اعداد صحیح بین ۰ تا ۱۸۴۴۶۷۴۴۰۷۳۷۰۹۵۵۱۶۱۵

به حرف u در ابتدای برخی از انواع داده‌ها مثلاً 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 می‌توانند در خود ذخیره کنند، را نشان می‌دهد:

نوع	محدوده	دقت
float	۳٫۴۰۲۸۲۳E۳۸- تا ۳٫۴۰۲۸۲۳E۳۸	۷ رقم
double	۱٫۷۹۷۶۹۳۱۳۴۸۶۲۳۲E۳۰۸- تا ۱٫۷۹۷۶۹۳۱۳۴۸۶۲۳۲E۳۰۸	۱۵-۱۶ رقم
decimal	۷۹۲۲۸۱۶۲۵۱۴۲۶۴۳۳۷۵۹۳۵۴۳۹۵۰۳۳۵- تا ۷۹۲۲۸۱۶۲۵۱۴۲۶۴۳۳۷۵۹۳۵۴۳۹۵۰۳۳۵	۲۸-۲۹ رقم

برای به خاطر سپردن آنها باید از نماد علمی استفاده شود. نوع دیگری از انواع ساده برای ذخیره داده‌های غیر عددی به کار می‌روند و در جدول زیر نمایش داده شده‌اند:

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

نوع char برای ذخیره کاراکترهای یونیکد استفاده می‌شود. کاراکترها باید داخل یک کوتیشن ساده قرار بگیرند مانند ('a'). نوع bool فقط می‌تواند مقادیر درست (true) یا نادرست (false) را در خود ذخیره کند و بیشتر در برنامه‌هایی که دارای ساختار تصمیم گیری هستند، مورد استفاده قرار می‌گیرد. نوع string برای ذخیره گروهی از کاراکترها مانند یک پیغام استفاده می‌شود. 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 ۸-۱۴ متغیرهایی با نوع و نام متفاوت تعریف شده‌اند. ابتدا باید نوع داده‌هایی را که این متغیرها قرار است در خود ذخیره کنند را مشخص کنیم و سپس یک نام برای آنها در نظر بگیریم و در آخر سمیکالن 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_type identifier;
```

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 می‌دهد. برای تعریف چند متغیر از یک نوع می‌توان به صورت زیر عمل کرد:

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

مثال

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

در مثال بالا ۵ متغیر از نوع صحیح و ۳ متغیر از نوع رشته تعریف شده است. توجه داشته باشید که بین متغیرها باید علامت کاما (,) باشد.

نامگذاری متغیرها

نام متغیر باید با یک حرف یا زیرخط و به دنبال آن حرف یا عدد شروع شود. نمی‌توان از کاراکترهای خاص مانند #، %، & یا عدد برای شروع نام متغیر استفاده کرد مانند 2numbers. نام متغیر نباید دارای فاصله باشد. برای نام‌های 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 که برای متغیرهای دو کلمه ای به کار می‌رود، اولین کلمه با حرف کوچک نوشته می‌شود و سایر کلمات با حرف بزرگ شروع می‌شوند. مانند myNumber. توجه کنید که اولین حرف کلمه Number با حرف بزرگ شروع شده است. مثال دیگر کلمه numberOfStudents است. اگر 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 می‌شوند. این متغیرها فقط در داخل متد Main() قابل دسترسی هستند. محدوده یک متغیر مشخص می‌کند که متغیر در کجای کد قابل دسترسی است. هنگامیکه برنامه به پایان متد Main() می‌رسد متغیرها از محدوده خارج و بدون استفاده می‌شوند. محدوده متغیرها انواعی دارد که در درس‌های بعدی با آنها آشنا می‌شوید. تشخیص محدوده متغیر بسیار مهم است. چون به وسیله آن می‌فهمید که در کجای کد می‌توان از متغیر استفاده کرد. باید یاد آور شد که دو متغیر در یک محدوده نمی‌توانند دارای نام یکسان باشند. مثلاً کد زیر در برنامه ایجاد خطا می‌کند:

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

از آنجاییکه سی‌شارپ به بزرگی و کوچک بودن حروف حساس است می‌توان از این خاصیت برای تعریف چند متغیر همنام 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

می‌توان فوراً بعد از تعریف متغیرها مقادیری را به آنها اختصاص داد. این عمل را مقداردهی می‌نامند. در زیر نحوه مقدار دهی متغیرها نشان داده شده است:

```
data_type identifier = value;
```

به عنوان مثال:

```
int myNumber = 7;
```

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

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

به عنوان مثال:

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

تعریف متغیر با مقدار دهی متغیرها متفاوت است. تعریف متغیر یعنی انتخاب 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 متغیری که هنوز تعریف نشده نمی‌توانید مقدار بدهید. شما فقط می‌توانید از متغیرهایی استفاده کنید که هم تعریف و هم مقدار دهی شده باشند. مثلاً متغیرهای بالا همه قابل استفاده هستند. در این مثال num1 و num2 هر دو تعریف شده‌اند و مقادیری از نوع صحیح به آنها اختصاص داده شده است. اگر نوع داده با نوع متغیر یکی نباشد برنامه پیغام خطا می‌دهد.

جانگهدار (Placeholders)

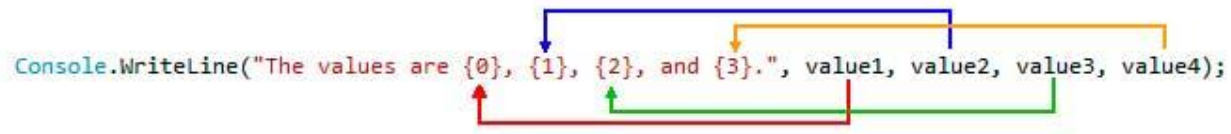
به متد WriteLine() در خطوط (۳۲-۲۶) توجه کنید. این متد دو آرگومان قبول می‌کند. آرگومانها اطلاعاتی هستند که متد با استفاده از آنها کاری انجام می‌دهد. آرگومانها به وسیله کاما از هم جدا می‌شوند. آرگومان اول، یک رشته قالب بندی شده است و آرگومان دوم مقداری است که توسط رشته قالب بندی شده مورد استفاده قرار می‌گیرد.

```
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);
```

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 می‌تواند از صفر تا n باشد. به این اعداد جانگهدار (Placeholder) می‌گویند. این اعداد بوسیله مقدار آرگومان بعد جایگزین می‌شوند. به عنوان مثال جانگهدار {۰} به این معناست که اولین آرگومان (مقدار) بعد از رشته قالب بندی شده در آن قرار می‌گیرد.

متد WriteLine() عملاً می‌تواند هر تعداد آرگومان قبول کند اولین آرگومان همان رشته قالب بندی شده است که جا نگهدار در آن قرار دارد و دومین آرگومان مقداری است که جایگزین جانگهدار می‌شود. در مثال زیر از چهار جا نگهدار استفاده شده است:

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



جا نگهدارها از صفر شروع می‌شوند. تعداد جانگهدارها باید با تعداد آرگومانهای بعد از رشته قالب بندی شده برابر باشد. برای مثال اگر شما چهار جا نگهدار 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 می‌شود. در ابتدا فهمیدن این مفهوم برای کسانی که تازه برنامه‌نویسی را شروع کرده‌اند 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 ۱ 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 System قرار دارند، استفاده می‌کنیم. پس به جای اینکه جمله زیر را به طور کامل بنویسیم:

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

می‌توانیم آن را ساده تر کرده و به صورت زیر بنویسیم:

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

در مورد فضای نام در درس‌های آینده توضیح خواهیم داد.

ثابت‌ها

ثابت‌ها انواعی از متغیرها هستند که مقدار آنها در طول برنامه تغییر نمی‌کند. ثابت‌ها حتماً باید مقدار دهی اولیه شوند و اگر مقدار دهی آنها فراموش شود در برنامه خطا به وجود می‌آید. بعد از این که به ثابت‌ها مقدار اولیه اختصاص داده شد، هرگز در زمان اجرای برنامه نمی‌توان آن را تغییر داد. برای تعریف ثابت‌ها باید از کلمه کلیدی const استفاده کرد. معمولاً نام ثابت‌ها را طبق قرارداد با حروف بزرگ می‌نویسند 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 می‌بینید که مقدار دادن به یک ثابت، که قبلاً مقدار دهی شده برنامه را با خطا مواجه می‌کند. نکته‌ی دیگری که نباید فراموش شود این است که، نباید مقدار ثابت را با مقدار دیگر متغیرهای تعریف شده در برنامه برابر قرار داد. به مثال زیر توجه کنید:

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

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

تبدیل ضمنی

تبدیل ضمنی متغیرها یک نوع تبدیل است که به طور خودکار توسط کامپایلر انجام می‌شود. یک متغیر از یک نوع داده می‌تواند به طور ضمنی به یک نوع دیگر تبدیل شود به شرطی که مقدار آن از مقدار داده‌ای که می‌خواهد به آن تبدیل شود کمتر باشد. به عنوان مثال نوع داده‌ای byte می‌تواند مقادیر ۰ تا ۲۵۵ را در خود ذخیره کند و نوع داده‌ای int مقادیر ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷ را شامل می‌شود. پس می‌توانید یک متغیر از نوع byte را به یک نوع int تبدیل کنید:

```
byte number1 = 5;
int number2 = number1;
```

در مثال بالا مقدار number1 برابر ۵ است در نتیجه متغیر number2 که یک متغیر از نوع صحیح است، می‌تواند مقدار number1 را در خود ذخیره کند. چون نوع int از نوع byte بزرگ‌تر است، پس متغیر number1 که یک متغیر از نوع byte است می‌تواند 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 می‌شویم. اگر چه مقدار ۵ متغیر number1 در محدوده مقادیر byte یعنی اعداد بین ۲۵۵-۰ قرار دارد اما متغیری از نوع byte حافظه کمتری نسبت به متغیری از نوع int اشغال می‌کند. نوع byte شامل ۸ بیت یا ۸ رقم دودویی است، در حالی که نوع int شامل ۳۲ بیت یا رقم دودویی است. یک عدد باینری عددی متشکل از اعداد ۰ و ۱ است. برای مثال عدد ۵ در کامپیوتر به عدد باینری ۱۰۱ ترجمه می‌شود. بنابراین وقتی ما عدد ۵ را در یک متغیر از نوع بایت ذخیره می‌کنیم عددی به صورت زیر نمایش داده می‌شود:

```
00000101
```

و وقتی آن را در یک متغیر از نوع صحیح ذخیره می‌کنیم به صورت زیر نمایش داده می‌شود:

```
00000000000000000000000000000101
```

بنابراین قرار دادن یک مقدار int در یک متغیر byte درست مانند این است که ما سعی کنیم که یک توپ فوتبال را در یک سوراخ کوچک گلف جای دهیم. برای قرار دادن یک مقدار int در یک متغیر از نوع byte می‌توان از تبدیل صریح استفاده کرد که در درس‌های آینده توضیح داده می‌شود. نکته دیگری که نباید فراموش شود این است که شما نمی‌توانید اعداد با ممیز اعشار را به یک نوع int تبدیل کنید چون این کار باعث از بین رفتن بخش اعشاری این اعداد می‌شود.

```
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 کاملاً متفاوت توسط کامپایلر ترجمه می‌شوند. نوع char به عنوان یک کاراکتر و نوع ushort به عنوان یک عدد ترجمه می‌شود.

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

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

```
c
99
```

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

نوع	قابلیت تبدیل به انواع
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

نکته‌ای دیگر که معمولاً ابهام بر انگیز است تعین نوع داده است. برای مثال ما چطور بدانیم که مثلاً عدد ۷ 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 می‌گیرد و در حالت پیش‌فرض کامپایلر اعداد دسیمال (decimal) را اعداد double در نظر می‌گیرد. شما می‌توانید 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 می‌کند 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 قبلاً 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 مثلاً از نوع double را به یک نوع int تبدیل می‌کنیم مقدار اعداد بعد از ممیز از بین می‌روند:

```
double number1 = 5.25;

int number2 = (int)number1;

Console.WriteLine(number2)
```

```
5
```

خروجی کد بالا عدد ۵ است چون نوع داده‌ای int نمی‌تواند 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 می‌افتد؟ مانند تبدیل زیر که می‌خواهیم متغیر number1 را که دارای مقدار ۳۰۰ است را به نوع Byte که محدود اعداد بین ۲۵۵-۰ را پوشش می‌دهد، تبدیل کنیم.

```
int number1 = 300;

byte number2 = (byte)number1;

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

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

خروجی کد بالا عدد ۴۴ است. Byte فقط می‌تواند شامل اعداد ۰ تا ۲۵۵ باشد و نمی‌تواند مقدار ۳۰۰ را در خود ذخیره کند. حال می‌خواهیم ببینیم که چرا به جای عدد ۳۰۰، عدد ۴۴ را در خروجی نمایش داده می‌شود. این کار به تعداد بیت‌ها بستگی دارد. یک byte دارای ۸ بیت است درحالی که int دارای ۳۲ بیت است. حال اگر به مقدار باینری ۲ عدد توجه کنید متوجه می‌شوید که چرا خروجی عدد ۴۴ است.

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

خروجی بالا نشان می‌دهد که بیشترین مقدار byte که عدد ۲۵۵ است، می‌تواند فقط شامل ۸ بیت باشد (11111111) بنابراین فقط ۸ بیت اول مقدار int به متغیر byte انتقال می‌یابد که شامل (00101100) یا عدد ۴۴ در مبنای ۱۰ است. قرار ندادن یک مقدار مناسب در داخل یک متغیر باعث ایجاد یک سرریز (overflow) می‌شود. یک مورد آن سرریز ریاضی نام دارد که در مثال زیر مشاهده می‌کنید:

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

گرچه در این تبدیل ما داده‌هایی را از دست می‌دهیم، اما کامپایلر کد ما را قبول می‌کند. برای اینکه برنامه هنگام وقوع سرریز پیغام خطا بدهد می‌توان 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 می‌توانید از اجرای برنامه جلوگیری کنید.

تبدیل با استفاده از کلاس Convert

.NET Framework دارای یک کلاس استاتیک است، که می‌توان 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 می‌باشد. در جدول زیر متدها ذکر شده‌اند:

دستور	نتیجه
Convert.ToBoolean(val)	مقدار val به نوع bool تبدیل می‌شود.
Convert.ToByte(val)	مقدار val به نوع byte تبدیل می‌شود.
Convert.ToChar(val)	مقدار val به نوع char تبدیل می‌شود.
Convert.ToDecimal(val)	مقدار val به نوع decimal تبدیل می‌شود.
Convert.ToDouble(val)	مقدار val به نوع double تبدیل می‌شود.
Convert.ToInt16(val)	مقدار val به نوع short تبدیل می‌شود.
Convert.ToInt32(val)	مقدار val به نوع int تبدیل می‌شود.
Convert.ToInt64(val)	مقدار val به نوع long تبدیل می‌شود.
Convert.ToSByte(val)	مقدار val به نوع ushort تبدیل می‌شود.

`Convert.ToSingle(val)`	مقدار `val` به نوع `float` تبدیل می‌شود.
`Convert.ToString(val)`	مقدار `val` به نوع `string` تبدیل می‌شود.
`Convert.ToUInt16(val)`	مقدار `val` به نوع `ushort` تبدیل می‌شود.
`Convert.ToUInt32(val)`	مقدار `val` به نوع `uint` تبدیل می‌شود.
`Convert.ToUInt64(val)`	مقدار `val` به نوع `ulong` تبدیل می‌شود.

در برنامه زیر یک نمونه از تبدیل متغیرها 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 داده‌ای می‌تواند باشد، اما باید مطمئن شد که به نوع داده‌ای مورد نظر تبدیل شود.

عبارات و عملگرها

ابتدا با دو کلمه آشنا شوید:

- عملگر: نمادهایی هستند که اعمال خاص انجام می‌دهند.
- عملوند: مقادیری که عملگرها بر روی آنها عملی انجام می‌دهند.

مثلاً X+Y یک عبارت است که در آن X و Y عملوند و علامت + عملگر به حساب می‌آیند. زبان‌های برنامه‌نویسی جدید دارای عملگرهایی هستند که از اجزاء معمول زبان به حساب می‌آیند. سی‌شارپ دارای عملگرهای مختلفی از جمله عملگرهای ریاضی، تخصیصی، مقایس‌های، منطقی و بیتی می‌باشد. از عملگرهای ساده ریاضی می‌توان به عملگر جمع و تفریق اشاره کرد. سه نوع عملگر در سی‌شارپ وجود دارد:

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

انواع مختلف عملگر که در این بخش مورد بحث قرار می‌گیرند عبارت‌اند از:

- عملگرهای ریاضی
- عملگرهای تخصیصی
- عملگرهای مقایسه ای

- عملگرهای منطقی
- عملگرهای بیتی

عملگرهای ریاضی

سی‌شارپ از عملگرهای ریاضی برای انجام محاسبات استفاده می‌کند. جدول زیر عملگرهای ریاضی سی‌شارپ را نشان می‌دهد:

عملگر	دسته	مثال	نتیجه
+	Binary	var1 = var2 + var3;	Var1 برابر است با حاصل جمع var2 و var3
-	Binary	var1 = var2 – var3;	Var1 برابر است با حاصل تفریق var2 و var3
*	Binary	var1 = var2 * var3;	Var1 برابر است با حاصلضرب var2 در var3
/	Binary	var1 = var2 / var3;	Var1 برابر است با حاصل تقسیم var2 بر var3
%	Binary	var1 = var2 % var3;	Var1 برابر است با باقیمانده تقسیم var2 و var3
+	Unary	var1 = +var2;	Var1 برابر است با مقدار var2
-	Unary	var1 = -var2	Var1 برابر است با مقدار var2 ضربدر ۱-

در مثال بالا از نوع عددی استفاده شده است. اما استفاده از عملگرهای ریاضی برای نوع رشته‌ای نتیجه متفاوتی دارد. همچنین در جمع دو کاراکتر کامپایلر معادل عددی آنها را نشان می‌دهد. اگر از عملگر + برای رشته‌ها استفاده کنیم دو رشته را با هم ترکیب کرده و به هم می‌چسباند. دیگر عملگرهای سی‌شارپ عملگرهای کاهش و افزایش هستند. این عملگرها مقدار ۱ را از متغیرها کم یا به آنها اضافه می‌کنند. از این متغیرها اغلب در حلقه‌ها استفاده می‌شود:

عملگر	دسته	مثال	نتیجه
++	Unary	var1 = ++var2;	مقدار var1 برابر است با var2 بعلاوه ۱. از متغیر var2 یک واحد اضافه می‌شود.
--	Unary	var1 = --var2;	مقدار var1 برابر است با var2 منهای ۱. از متغیر var2 یک واحد کم می‌شود.
++	Unary	var1 = var2++;	مقدار var1 برابر است با var2. به متغیر var2 یک واحد اضافه می‌شود.
--	Unary	var1 = var2--;	مقدار var1 برابر است با var2.

			از متغیر var2 یک واحد کم می‌شود.

به این نکته توجه داشته باشید که محل قرارگیری عملگر در نتیجه محاسبات تأثیر دارد. اگر عملگر قبل از متغیر var2 بیاید افزایش یا کاهش var1 اتفاق می‌افتد و var2 تغییر نمی‌کند. چنانچه عملگرها بعد از متغیر var2 قرار بگیرند ابتدا var1 برابر var2 می‌شود و سپس متغیر var2 افزایش یا کاهش می‌یابد. به مثال‌های زیر توجه کنید:

```
using System;

namespace ConsoleApplication5
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 0;
            int y = 1;

            x = ++y;

            Console.WriteLine("x= {0}",x);
            Console.WriteLine("y= {0}", y);
            Console.ReadLine();
        }
    }
}
```

```
x=2
y=2
```

```
using System;

using System;

namespace ConsoleApplication5
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 0;
            int y = 1;

            x = --y;

            Console.WriteLine("x= {0}",x);
            Console.WriteLine("y= {0}", y);
            Console.ReadLine();
        }
    }
}
```

```
x=0
y=0
```

همانطور که در دو مثال بالا مشاهده می‌کنید، درج عملگرهای -- و ++ قبل از عملوند y باعث می‌شود که ابتدا یک واحد از y کم و یا یک واحد به y اضافه شود و سپس نتیجه در عملوند x قرار بگیرد. حال به دو مثال زیر توجه کنید:

```
using System;

namespace ConsoleApplication5
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 0;
            int y = 1;

            x = y--;

            Console.WriteLine("x= {0}",x);
            Console.WriteLine("y= {0}", y);
            Console.ReadLine();
        }
    }
}
```

```
x=1
y=0
```

```
using System;

namespace ConsoleApplication5
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 0;
            int y = 1;

            x = y++;

            Console.WriteLine("x= {0}",x);
            Console.WriteLine("y= {0}", y);
            Console.ReadLine();
        }
    }
}
```

```
x=1
y=2
```

همانطور که در دو مثال بالا مشاهده می‌کنید، درج عملگرهای -- و ++ بعد از عملوند y باعث می‌شود که ابتدا مقدار y در داخل متغیر x قرار بگیرد و سپس یک واحد از y کم و یا یک واحد به آن اضافه شود. حال می‌توانیم با ایجاد یک برنامه نحوه عملکرد عملگرهای ریاضی در سی‌شارپ را یاد بگیریم:

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           //Variable declarations
8           int num1, num2;
9           string msg1, msg2;
10
```

```
11          //Assign test values
12          num1 = 5;
13          num2 = 3;
14
15          //Demonstrate use of mathematical operators
16          Console.WriteLine("The sum of {0} and {1} is {2}.",
17                            num1, num2, (num1 + num2));
18          Console.WriteLine("The difference of {0} and {1} is {2}.",
19                            num1, num2, (num1 - num2));
20          Console.WriteLine("The product of {0} and {1} is {2}.",
21                            num1, num2, (num1 * num2));
22          Console.WriteLine("The quotient of {0} and {1} is {2:F2}.",
23                            num1, num2, ((double)num1 / num2));
24          Console.WriteLine("The remainder of {0} divided by {1} is {2}",
25                            num1, num2, (num1 % num2));
26
27          //Demonstrate concatenation on strings using the + operator
28          msg1 = "Hello ";
29          msg2 = "World!";
30          Console.WriteLine(msg1 + msg2);
31      }
32  }
```

```
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 می‌دهد. در این برنامه از متد Writeline() برای نشان دادن نتایج در سطرهای متفاوت استفاده شده است. در این مثال با یک نکته عجیب مواجه می‌شویم 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 می‌کنید اگر فقط یکی از اعداد را به نوع اعشاری double تبدیل کنیم (در مثال می‌بینید) حاصل به صورت اعشار نشان داده می‌شود. برای اینکه ارقام کسری بعد از عدد حاصل دو رقم باشند از {2:F2} استفاده می‌کنیم. F به معنای فرمت بندی می‌باشد 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 می‌نویسیم. سی‌شارپ خط جدید، فاصله و فضای خالی را نادیده می‌گیرد.

در خط ۲۹ مشاهده می‌کنید که دو رشته به وسیله عملگر + به هم متصل شده‌اند. نتیجه استفاده از عملگر + برای چسباندن دو کلمه "Hello " و "World!" رشته "Hello World!" خواهد بود. به فاصله خالی بعد از کلمه Hello توجه کنید اگر آن را حذف کنید از خروجی برنامه نیز حذف می‌شود.

عملگرهای تخصیصی (جایگزینی)

نوع دیگر از عملگرهای سی‌شارپ عملگرهای جایگزینی نام دارند. این عملگرها مقدار متغیر سمت راست خود را در متغیر سمت چپ قرار می‌دهند. جدول زیر انواع عملگرهای تخصیصی در سی‌شارپ را نشان می‌دهد:

عملگر	مثال	نتیجه
=	var1 = var2;	مقدار var1 برابر است با مقدار var2.

+=	var1 += var2;	مقدار var1 برابر است با حاصل جمع var1 و var2.
-=	var1 -= var2;	مقدار var1 برابر است با حاصل تفریق var1 و var2.
*=	var1 *= var2;	مقدار var1 برابر است با حاصل ضرب var1 در var2.
/=	var1 /= var2;	مقدار var1 برابر است با حاصل تقسیم var1 بر var2.
%=	var1 %= var2;	مقدار var1 برابر است با باقیمانده تقسیم var1 بر var2.

از عملگر =+ برای اتصال دو رشته نیز می‌توان استفاده کرد. استفاده از این نوع عملگرها در واقع یک نوع خلاصه نویسی در کد است. مثلاً شکل اصلی کد var1 += var2 به صورت var1 = var1 + var2 می‌باشد. این حالت کدنویسی زمانی کارایی خود را نشان می‌دهد که نام متغیرها طولانی باشد. برنامه زیر چگونگی استفاده از عملگرهای تخصیصی و تأثیر آنها را بر متغیرها نشان می‌دهد.

```
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 سه عملگر تخصیصی استفاده شده است. ابتدا یک متغیر و مقدار ۱۰ با استفاده از عملگر = به آن اختصاص داده شده است. سپس به آن با استفاده از عملگر =+ مقدار ۱۰ اضافه شده است. و در آخر به وسیله عملگر =- عدد ۱۰ از آن کم شده است.

عملگرهای مقایسه ای

از عملگرهای مقایسه ای برای مقایسه مقادیر استفاده می‌شود. نتیجه این مقادیر یک مقدار بولی (منطقی) است. این عملگرها اگر نتیجه مقایسه دو مقدار درست باشد، مقدار true و اگر نتیجه مقایسه اشتباه باشد، مقدار false را نشان می‌دهند. این عملگرها به طور معمول در دستورات شرطی به کار می‌روند. به این ترتیب که باعث ادامه یا توقف دستور شرطی می‌شوند. جدول زیر عملگرهای مقایسه ای در سی‌شارپ را نشان می‌دهد:

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 var2 بزرگ‌تر از مقدار var3 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 این عملگرها را نشان می‌دهد:

```
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 می‌خواهیم با هم مقایسه کنیم را ایجاد کرده و به آنها مقادیری اختصاص می‌دهیم. سپس با استفاده از یک عملگر مقایسه ای آنها را با هم مقایسه کرده و نتیجه را چاپ می‌کنیم. 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 می‌دهد. عملگر == عملگر مقایسه ای است که دو مقدار را با هم مقایسه می‌کند مانند x==y و اینطور خوانده می‌شود x برابر است با y.

عملگرهای منطقی

عملگرهای منطقی بر روی عبارات منطقی عمل می‌کنند و نتیجه آنها نیز یک مقدار بولی است. از این عملگرها اغلب برای شرط‌های پیچیده استفاده می‌شود. همانطور که قبلاً یاد گرفتید مقادیر بولی می‌توانند false یا true باشند. فرض کنید که var2 و var3 دو مقدار بولی هستند.

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

عملگر منطقی (&&)AND

اگر مقادیر دو طرف این عملگر، true باشند، عملگر AND مقدار true را بر می‌گرداند. در غیر اینصورت اگر یکی از مقادیر یا هر دوی آنها false باشند، مقدار false را بر می‌گرداند. در زیر جدول درستی عملگر AND نشان داده شده است:

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

برای درک بهتر تأثیر عملگر AND یادآوری می‌کنیم که این عملگر فقط در صورتی مقدار true را نشان می‌دهد 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 ۱۸ و salary کوچک‌تر از ۱۰۰۰ باشد.

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

عملگر AND زمانی کارآمد است که ما با محدوده خاصی از اعداد سرو کار داریم. مثلاً عبارت 10 <= x <= 100 بدین معنی است که x می‌تواند مقداری شامل اعداد ۱۰ تا ۱۰۰ را بگیرد. حال برای انتخاب اعداد خارج از این محدوده می‌توان 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 می‌گرداند. 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 می‌کنید 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 ۷۵ یا نمره نهایی امتحان آن ۱۰۰ 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 عبارت بولی را نفی می‌کند. مثلاً اگر عبارت یا مقدار true باشد آنرا false و اگر false باشد آنرا true می‌کند. جدول زیر عملکرد اپراتور NOT را نشان می‌دهد:

X	!X
true	false
false	true

نتیجه کد زیر در صورتی درست است که age (سن) بزرگ‌تر یا مساوی ۱۸ نباشد.

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

عملگرهای بیتی

عملگرهای بیتی به شما اجازه می‌دهند که شکل باینری انواع داده‌ها را دستکاری کنید. برای درک بهتر این درس توصیه می‌شود که شما سیستم باینری و نحوه تبدیل اعداد اعشاری به باینری را از لینک زیر یاد بگیرید:

```
http://www.w3-farsi.com/?p=5698
```

در سیستم باینری (دودویی) که کامپیوتر از آن استفاده می‌کند وضعیت هر چیز یا خاموش است یا روشن. برای نشان دادن حالت روشن از عدد ۱ و برای نشان دادن حالت خاموش از عدد ۰ استفاده می‌شود. بنابراین اعداد باینری فقط می‌توانند صفر یا یک باشند. اعداد باینری را اعداد در مبنای ۲ و اعداد اعشاری را اعداد در مبنای ۱۰ می‌گویند. یک بیت نشان دهنده یک رقم باینری است و هر بایت نشان دهنده ۸ بیت است. به عنوان مثال برای یک داده از نوع int به ۳۲ بیت یا ۴ بایت فضا برای ذخیره آن نیاز داریم، این بدین معناست که اعداد از ۳۲ رقم ۰ و ۱ برای ذخیره استفاده می‌کنند. برای مثال عدد ۱۰۰ وقتی به عنوان یک متغیر از نوع int ذخیره می‌شود در کامپیوتر به صورت زیر خوانده می‌شود:

```
00000000000000000000000001100100
```

عدد ۱۰۰ در مبنای ده معادل عدد ۱۱۰۰۱۰۰ در مبنای ۲ است. در اینجا ۷ رقم سمت راست نشان دهنده عدد ۱۰۰ در مبنای ۲ است و مابقی صفرهای سمت چپ 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 می‌شوند. عملگرهای بیتی سی‌شارپ در جدول زیر نشان داده شده‌اند:

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 – تخصیصی AND	Binary	x &= y;
|=	بیتی – تخصیصی OR	Binary	x |= y;
^=	بیتی – تخصیصی XOR	Binary	x ^= y;

عملگر بیتی AND(&)

عملگر بیتی AND مانند کاری شبیه عملگر منطقی AND انجام می‌دهد با این تفاوت که این عملگر بر روی بیت‌ها کار می‌کند. اگر مقادیر دو طرف آن ۱ باشد، مقدار ۱ را بر می‌گرداند و اگر یکی یا هر دو طرف آن صفر باشد مقدار صفر را بر می‌گرداند. 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 می‌کنید، نتیجه عملکرد عملگر AND بر روی دو مقدار ۵ و ۳ عدد ۱ می‌شود. اجازه بدهید ببینیم که چطور این نتیجه را به دست می‌آید:

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

ابتدا دو عدد ۵ و ۳ به معادل باینری‌شان تبدیل می‌شوند. از آنجاییکه هر عدد صحیح (int) ۳۲ بیت است از صفر برای پر کردن بیت‌های خالی استفاده می‌کنیم. با استفاده از جدول درستی عملگر بیتی AND می‌توان فهمید که چرا نتیجه عدد یک می‌شود.

عملگر بیتی OR(|)

اگر مقادیر دو طرف عملگر بیتی OR هر دو صفر باشند نتیجه ۰ در غیر اینصورت ۱ 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 ۰ است که عملوندهای دو طرف آن ۰ باشند. اگر فقط یکی از دو عملوند ۱ باشد، نتیجه ۱ 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 (۷ و ۹) استفاده می‌کنیم، نتیجه ۱۵ می‌شود. حال بررسی می‌کنیم که چرا این نتیجه به دست آمده است؟

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

با استفاده از جدول درستی عملگر بیتی OR می‌توان نتیجه استفاده از این عملگر را تشخیص داد. عدد ۱۱۱۱ باینری معادل عدد ۱۵ 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 ۰ یا هر دو ۱ باشند نتیجه ۰، در غیر اینصورت نتیجه ۱ می‌شود. در مثال زیر تأثیر عملگر بیتی XOR را بر روی دو مقدار مشاهده می‌کنید:

```
int result = 5 ^ 7;

Console.WriteLine(result);
```

```
2
```

در زیر معادل باینری اعداد بالا (۵ و ۷) نشان داده شده است.

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

با نگاه کردن به جدول درستی عملگر بیتی XOR می‌توان فهمید که چرا نتیجه عدد ۲ می‌شود.

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 می‌کند. 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 برنامه‌تان استفاده کنید. کدهای مربوط به هر سبک هم در جدول زیر آمده است:

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

توجه کنید که مقدار اولیه ۰ بدین معنی است که می‌خواهید از سبک regular (عادی) استفاده کنید.

```
int fontStyle = 0;
```

برای نشان دادن فونت‌ها به صورت کلفت (Bold) از عملگر بیتی OR استفاده می‌شود. توجه کنید که برای فونت Bold باید کد ۱ را به کار برید.

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

برای استفاده از سبک Italic باید از عملگر بیتی OR و کد ۲ استفاده شود.

```
fontStyle |= 2;
```

برای استفاده از سایر سبک‌ها می‌توان به روش‌های 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، می‌توانید 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 می‌دهند 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 می‌کنند. عملوند سمت چپ این عملگرها حالت باینری یک مقدار و عملوند سمت راست تعداد جابه جایی بیت‌ها را نشان می‌دهد.

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 می‌کند. به عنوان مثال:

```
int result = 10 << 2;

Console.WriteLine(result);
```

```
40
```

در مثال بالا ما بیت‌های مقدار ۱۰ را دو مکان به سمت چپ منتقل کرده‌ایم، حال بیایید تأثیر این انتقال را بررسی کنیم:

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

مشاهده می‌کنید که همه بیت‌ها به اندازه دو واحد به سمت چپ منتقل شده‌اند. در این انتقال دو صفر از صفرهای سمت چپ کم می‌شود و در عوض دو صفر به سمت راست اضافه می‌شود.

عملگر تغییر مکان به سمت راست

این عملگر شبیه به عملگر تغییر مکان به سمت چپ است با این تفاوت که بیت‌ها را به سمت راست جا به جا می‌کند. به عنوان مثال:

```
int result = 100 >> 4;

Console.WriteLine(result);
```

```
6
```

با استفاده از عملگرتغییر مکان به سمت راست بیت‌های مقدار ۱۰۰ را به اندازه ۴ واحد به سمت چپ جا به جا می‌کنیم. اجازه بدهید تأثیر این جا به جایی را مورد بررسی قرار دهیم:

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

هر بیت به اندازه ۴ واحد به سمت راست منتقل می‌شود، بنابراین ۴ بیت اول سمت راست حذف شده و چهار صفر به سمت چپ اضافه می‌شود.

تقدم عملگرها

تقدم عملگرها مشخص می‌کند که در محاسباتی که بیش از دو عملوند دارند، ابتدا کدام عملگر اثرش را اعمال کند. عملگرها در سی‌شارپ 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 ۹ خواهد شد (۳=۲+۱ سپس ۹=۳×۳ و در آخر ۹/۱=۹). اما کامپایلر با توجه به تقدم عملگرها محاسبات را انجام می‌دهد. برای مثال عمل ضرب و تقسیم نسبت به جمع و تفریق تقدم دارند. بنابراین در مثال فوق ابتدا عدد ۲ ضربدر ۳ و سپس نتیجه آنها تقسیم بر ۱ می‌شود که نتیجه ۶ به دست می‌آید. در آخر عدد ۶ با ۱ جمع می‌شود و عدد ۷ حاصل می‌شود. در جدول زیر تقدم برخی از عملگرهای سی‌شارپ آمده است:

عملگرها	تقدم
++, --, (used as prefixes); +, - (unary)	بالاترین
*, /, %	
+, -	
<<, >>	
<, >, <=, >=	
==, !=	
&	
^	
|	
&&	
||	
=, *=, /=, %=, +=, -=	
++, -- (used as suffixes)	پایین‌ترین

ابتدا عملگرهای با بالاترین و سپس عملگرهای با پایین‌ترین حق تقدم در محاسبات تأثیر می‌گذارند. 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 ۴ می‌شود و سپس مقدار جدید با عدد ۳ جمع می‌شود و در نهایت عدد ۷ به دست می‌آید. در عبارت دوم مقدار عددی ۳ به مقدار number اضافه می‌شود و عدد ۶ به دست می‌آید. سپس این مقدار در متغیر number2 قرار می‌گیرد. و در نهایت مقدار number به ۴ افزایش می‌یابد. برای ایجاد خوانایی در تقدم عملگرها و انجام محاسباتی که در آنها از عملگرهای زیادی استفاده می‌شود از پرانتز استفاده می‌کنیم:

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

در مثال بالا ابتدا هر کدام از عباراتی که داخل پرانتز هستند مورد محاسبه قرار می‌گیرند. به نکته‌ای در مورد عبارتی که در داخل پرانتز سوم قرار دارد توجه کنید. در این عبارت ابتدا مقدار داخلی‌ترین پرانتز مورد محاسبه قرار می‌گیرد یعنی مقدار ۶ ضربدر ۷ شده و سپس از ۵ کم می‌شود. اگر دو یا چند عملگر با حق تقدم یکسان موجود باشد ابتدا باید هر کدام از عملگرها را که در ابتدای عبارت می‌آیند مورد ارزیابی قرار دهید. به عنوان مثال:

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

هر دو عملگر * و / دارای حق تقدم یکسانی هستند. بنابراین شما باید از چپ به راست آنها را در محاسبات تأثیر دهید. یعنی ابتدا ۳ را ضربدر ۲ می‌کنید و سپس عدد ۸ را بر ۴ تقسیم می‌کنید. در نهایت نتیجه دو عبارت را جمع کرده و در متغیر number قرار می‌دهید.

گرفتن ورودی از کاربر

چارچوب دات‌نت تعدادی متد برای گرفتن ورودی از کاربر در اختیار شما قرار می‌دهد. حال می‌خواهیم در باره متد ReadLine() یکی دیگر از متدهای کلاس Console بحث کنیم که یک مقدار رشته‌ای را از کاربر دریافت می‌کند. متد ReadLine() فقط مقدار رشته‌ای را که توسط کاربر نوشته می‌شود را بر می‌گرداند. همانطور که از نام این متد پیداست، تمام کاراکترهایی را که شما در محیط کنسول تایپ می‌کنید تا زمانی که دکمه Enter را می‌زنید می‌خواند. هر چه که در محیط کنسول تایپ می‌شود از نوع رشته است. برای تبدیل نوع رشته به انواع دیگر می‌توانید 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  }
```

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

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

ابتدا ۳ متغیر را برای ذخیره داده در برنامه تعریف می‌کنیم (خطوط ۷ و ۸ و ۹). برنامه از کاربر می‌خواهد که نام خود را وارد کند (خط ۱۱). در خط ۱۲ شما به عنوان کاربر نام خود را وارد می‌کنید. مقدار متغیر نام، برابر مقداری است که توسط متد ReadLine() خوانده می‌شود. از آنجاییکه نام از نوع رشته است و مقداری که از متد ReadLine() خوانده می‌شود هم از نوع رشته است در نتیجه نیازی به تبدیل انواع نداریم.

سپس برنامه از ما سن را سؤال می‌کند (خط ۱۳). 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 می‌کنیم (خط ۱۴). مقدار بازگشتی از این متد در متغیر سن قرار می‌گیرد. چون متغیر قد (height) را از نوع double تعریف کرده‌ایم برای تبدیل رشته دریافتی از محیط کنسول به نوع double باید از متد ToDouble() مربوط به کلاس Convert استفاده کنیم (خط ۱۶). علاوه بر آنچه گفته شد شما می‌توانید از متد Parse() برای تبدیل‌های 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 می‌شود، یعنی رشته‌ای که توسط کاربر تایپ می‌شود، باید فقط عدد باشد.

ساختارهای تصمیم

تقریباً همه زبان‌های برنامه‌نویسی به شما اجازه اجرای کد را در شرایط مطمئن می‌دهند. 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 در صورتیکه مقدار یک متغیر با یک عدد برابر باشد، سپس یک پیغام چاپ شود آن وقت با مشکل مواجه خواهید شد. سی‌شارپ راه‌های مختلفی برای رفع این نوع مشکلات ارائه می‌دهد. 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

می‌توان با استفاده از دستور if و یک شرط خاص که باعث ایجاد یک کد می‌شود یک منطق به برنامه خود اضافه کنید. دستور if ساده‌ترین دستور شرطی است که برنامه می‌گوید 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 می‌شود. اگر شرط برقرار باشد یعنی درست باشد سپس کد اجرا می‌شود. شرط یک عبارت مقایسه ای است. می‌توان 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 ۱۰ و Goodbye World را اگر مقدار number از ۱۰ بزرگ‌تر باشد در صفحه نمایش می‌دهد.

```
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  }
```

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

در خط ۸ یک متغیر با نام number تعریف و مقدار ۵ به آن اختصاص داده شده است. وقتی به اولین دستور if در خط ۱۱ می‌رسیم برنامه تشخیص می‌دهد که مقدار number از ۱۰ کمتر است؟ یعنی ۵ کوچک‌تر از ۱۰ است؟ منطقی است که نتیجه مقایسه درست می‌باشد. بنابراین خط ۱۲ اجرا و پیغام Hello World چاپ می‌شود. حال مقدار number را به ۱۵ تغییر می‌دهیم (خط ۱۶). وقتی به دومین دستور if در خط ۱۹ می‌رسیم برنامه مقدار number را با ۱۰ مقایسه می‌کند و چون مقدار number یعنی ۱۵ از ۱۰ بزرگ‌تر است برنامه پیغام Goodbye World را چاپ می‌کند (خط ۲۰). به این نکته توجه کنید که دستور if را می‌توان در یک خط نوشت:

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

شما می‌توانید چندین دستور 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 جزء 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 ۱۰ بزرگ‌تر باشد دو پیغام چاپ می‌شود. حال اگر به عنوان مثال آکولاد را حذف کنیم و مقدار x از ۱۰ 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 می‌شود 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?)");
```

می‌بینید که دستور دوم (خط ۳) در مثال بالا جزء دستور if نیست. اینجاست که چون ما فرض را بر این گذاشته‌ایم که مقدار x از ۱۰ کوچک‌تر است پس خط This is still part of the if statement (Really?) چاپ می‌شود. در نتیجه اهمیت وجود آکولاد مشخص می‌شود. 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 می‌کند. یکی از خطاهای معمول کسانی که برنامه‌نویسی را تازه شروع کرده‌اند قرار دادن سمیکالن 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 می‌شود. همیشه به یاد داشته باشید که قرار گرفتن سمیکالن 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);
        }
    }
}
```

```
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 کرده‌ایم. ابتدا دو عدد که قرار است با هم مقایسه شوند را به عنوان ورودی از کاربر می‌گیریم. اعداد با هم مقایسه می‌شوند و اگر شرط درست بود پیغامی چاپ می‌شود. به این نکته توجه داشته باشید که شرط‌ها مقادیر بولی هستند، بنابراین شما می‌توانید نتیجه یک عبارت را در داخل یک متغیر 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 ۲۰۰۰ باشد سپس حاصل عبارت در متغیر isNewMillenium ذخیره می‌شود. می‌توان 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 می‌رود 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 نمی‌توان به تنهایی استفاده کرد بلکه حتماً 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 می‌شود 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 ۱۰ کمتر باشد کد داخل بلوک if اجرا می‌شود و اگر مقدار number را تغییر دهیم و به مقداری بزرگ‌تر از ۱۰ تغییر دهیم شرط نادرست می‌شود و کد داخل بلوک else اجرا می‌شود. مانند بلوک if نباید به آخر کلمه کلیدی else سمیکالن اضافه شود.

عملگر شرطی

عملگر شرطی (:?) در سی‌شارپ مانند دستور شرطی if…else عمل می‌کند. در زیر نحوه استفاده از این عملگر آمده است:

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

عملگر شرطی تنها عملگر سه تایی سی‌شارپ 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 می‌دهد. خط یک به صورت زیر ترجمه می‌شود: اگر مقدار pet1 برابر با puppy بود، مقدار dog را در type1 قرار بده در غیر اینصورت مقدار cat را type1 قرار بده. خط دو به صورت زیر ترجمه می‌شود: اگر مقدار pet2 برابر با kitten بود، مقدار cat را در type2 قرار بده در غیر اینصورت مقدار dog. حال برنامه بالا را با استفاده از دستور if else می‌نویسیم:

```
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 می‌آورد.

دستور if چندگانه

اگر بخواهید چند شرط را بررسی کنید چکار می‌کنید؟ می‌توانید 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 استفاده از دستور else if را یاد گرفتید باید بدانید که مانند else، else if نیز به دستور if وابسته است. دستور else if وقتی اجرا می‌شود که اولین دستور if اشتباه باشد. حال اگر else if اشتباه باشد دستور else if بعدی اجرا می‌شود. و اگر آن نیز اجرا نشود در نهایت دستور else اجرا می‌شود. برنامه زیر نحوه استفاده از دستور else if را نشان می‌دهد:

```
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.");
        }
    }
}
```

```
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 می‌کنید پیغام‌های مختلفی چاپ می‌شود. اگر عددی که شما تایپ می‌کنید در داخل حالت‌های انتخاب نباشد، کد مربوط به بلوک `else` اجرا می‌شود.

دستور if تو در تو

می‌توان از دستور if تو در تو در سی‌شارپ 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.");
            }
        }
```

```
34          else
35          {
36              Console.WriteLine("You are still too young.");
37          }
38      }
39  }
```

```
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 سنتان سؤال می‌کند (خط ۹). در خط ۱۲ درباره جنستان از شما سؤال می‌کند. سپس به اولین دستور `if` می‌رسد (خط ۱۶).

در این قسمت اگر سن شما بیشتر از ۱۲ سال باشد برنامه وارد بدنه دستور `if` می‌شود در غیر اینصورت وارد بلوک `else` (خط ۳۴) مربوط به همین دستور `if` می‌شود. حال فرض کنیم که سن شما بیشتر از ۱۲ سال است و شما وارد بدنه اولین `if` شده‌اید. در بدنه اولین `if` دو دستور `if` دیگر را مشاهده می‌کنید. اگر سن کمتر ۲۰ باشد شما وارد بدنه `if` دوم می‌شوید و اگر نباشد به قسمت `else` متناظر با آن می‌روید (خط ۲۹). دوباره فرض می‌کنیم که سن شما کمتر از ۲۰ باشد، در اینصورت وارد بدنه `if` دوم شده و با یک `if` دیگر مواجه می‌شوید (خط ۲۰). در اینجا جنسیت شما مورد بررسی قرار می‌گیرد که اگر برابر `"male"` باشد کدهای داخل بدنه سومین `if` اجرا می‌شود در غیر اینصورت قسمت `else` مربوط به این `if` اجرا می‌شود (خط ۲۴). پیشنهاد می‌شود که از `if` تو در تو در برنامه کمتر استفاده کنید چون خوانایی برنامه را پایین می‌آورد.

استفاده از عملگرهای منطقی

عملگرهای منطقی به شما اجازه می‌دهند که چندین شرط را با هم ترکیب کنید. این عملگرها حداقل دو شرط را درگیر می‌کنند و در آخر یک مقدار بولی را بر می‌گردانند. در جدول زیر برخی از عملگرهای منطقی آمده است:

عملگر	تلفظ	مثال	تأثیر
&&	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 مقدارشان `false` 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 ۲ و مقدار y کوچک‌تر از ۱۰ 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 (||) تأثیر 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. می‌توان 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 می‌کنیم. در اینجا ابتدا عبارت `((z < 10) || (y > 3)` مورد بررسی قرار می‌گیرد (به علت تقدم عملگرها). سپس نتیجه آن بوسیله عملگر AND با نتیجه `(x == 1)` مقایسه می‌شود. حال بیایید نحوه استفاده از عملگرهای منطقی در برنامه را مورد بررسی قرار دهیم:

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           int age;
8           string gender;
9
10          Console.Write("Enter your age: ");
11          age = Convert.ToInt32(Console.ReadLine());
12
13          Console.Write("Enter your gender (male/female): ");
14          gender = Console.ReadLine();
15
16          if (age > 12 && age < 20)
17          {
18              if (gender == "male")
19              {
20                  Console.WriteLine("You are a teenage boy.");
21              }
22              else
23              {
24                  Console.WriteLine("You are a teenage girl.");
25              }
26          }
27          else
28          {
29              Console.WriteLine("You are not a teenager.");
30          }
31      }
32  }
```

```
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 می‌دهد (خط ۱۶). وقتی به دستور if می‌رسید (خط ۱۶) برنامه سن شما را چک می‌کند. اگر سن شما بزرگ‌تر از ۱۲ و کوچک‌تر از ۲۰ باشد (سنتان بین ۱۲ و ۲۰ باشد) یعنی مقدار هر دو true باشد سپس کدهای داخل بلوک if اجرا می‌شوند. اگر نتیجه یکی از شروط false باشد کدهای داخل بلوک else اجرا می‌شود. عملگر AND عملوند سمت چپ را مورد بررسی قرار می‌دهد. اگر مقدار آن false باشد دیگر عملوند سمت راست را بررسی نمی‌کند و مقدار false را بر می‌گرداند. بر عکس عملگر || عملوند سمت چپ را مورد بررسی قرار می‌دهد و اگر مقدار آن true باشد سپس عملوند سمت راست را نادیده می‌گیرد و مقدار true را بر می‌گرداند. نکته مهم اینجاست که شما می‌توانید 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 می‌روند این است که دو عملوند را بدون در نظر گرفتن مقدار عملوند سمت چپ مورد بررسی قرار می‌دهند. به عنوان مثال حتی اگر مقدار عملوند سمت چپ false باشد، عملوند سمت چپ به وسیله عملگر بیتی AND(&) ارزیابی می‌شود. 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 می‌کند. 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 می‌کند.

دستور Switch

در سی‌شارپ ساختاری به نام switch وجود دارد که به شما اجازه می‌دهد 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، رشته‌ها و یا کاراکترها را قبول می‌کند. 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 می‌دهید. این مقدار با هر یک از عبارت‌های case داخل بلوک switch مقایسه می‌شود. 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 می‌شود که باعث می‌شود برنامه از دستور switch خارج شده و دستورات بعد از آن اجرا شوند. اگر این کلمه کلیدی از قلم بیوفتد برنامه با خطا مواجه می‌شود. دستور switch یک بخش default دارد. این دستور در صورتی اجرا می‌شود 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 نمی‌افتد. مکان این دستور هم مهم نیست اما بر طبق تعریف آن را در پایان دستورات می‌نویسند. به مثالی در مورد دستور switch توجه کنید:

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           int choice;
8
9           Console.WriteLine("What's your favorite pet?");
10          Console.WriteLine("[1] Dog");
11          Console.WriteLine("[2] Cat");
12          Console.WriteLine("[3] Rabbit");
13          Console.WriteLine("[4] Turtle");
14          Console.WriteLine("[5] Fish");
15          Console.WriteLine("[6] Not in the choices");
16          Console.Write("\nEnter your choice: ");
17
18          choice = Convert.ToInt32(Console.ReadLine());
19
20          switch (choice)
21          {
22              case 1:
23                  Console.WriteLine("Your favorite pet is Dog.");
24                  break;
25              case 2:
26                  Console.WriteLine("Your favorite pet is Cat.");
27                  break;
28              case 3:
29                  Console.WriteLine("Your favorite pet is Rabbit.");
30                  break;
31              case 4:
32                  Console.WriteLine("Your favorite pet is Turtle.");
33                  break;
34              case 5:
35                  Console.WriteLine("Your favorite pet is Fish.");
36                  break;
37              case 6:
38                  Console.WriteLine("Your favorite pet is not in the choices.");
39                  break;
40              default:
41                  Console.WriteLine("You don't have a favorite pet.");
42                  break;
43          }
44      }
```

```
45  }
```

```
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 علاقه‌تان را می‌دهد. به اسم هر حیوان یک عدد نسبت داده شده است. شما عدد را وارد می‌کنید و این عدد در دستور switch با مقادیر case مقایسه می‌شود و با هر کدام از آن مقادیر که برابر بود پیغام مناسب نمایش داده خواهد شد. اگر هم با هیچ کدام از مقادیر caseها برابر نبود دستور default اجرا می‌شود. یکی دیگر از ویژگی‌های دستور switch این است که شما می‌توانید از دو یا چند case برای نشان داده یک مجموعه کد استفاده کنید. در مثال زیر اگر مقدار number، ۱، ۲ یا ۳ باشد یک کد اجرا می‌شود. توجه کنید که caseها 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 قبلاً ذکر شد دستور switch معادل دستور if تو در تو است. برنامه بالا را به صورت زیر نیز می‌توان 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 دقیقاً نتیجه‌ای مانند دستور switch دارد. دستور default معادل دستور else می‌باشد. حال از بین این دو دستور (switch و if else) کدامیک را انتخاب کنیم. از دستور switch موقعی استفاده می‌کنیم که مقداری که می‌خواهیم با دیگر مقادیر مقایسه شود ثابت باشد. مثلاً 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 می‌کنید که با اینکه مقدار x عدد ۵ است و به طور واضح با متغیر myNumber مقایسه شده است برنامه خطا می‌دهد 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 می‌شود. توجه کنید که بعد از تعریف یک ثابت نمی‌توان مقدار آن را در طول برنامه تغییر داد. به یاد داشته باشید که باید ثابت‌ها را حتماً مقداردهی کنید. دستور switch یک مقدار را با مقادیر Caseها مقایسه می‌کند 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 می‌دهند 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. مثلاً شما مجبورید ۱۰ 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 می‌توانید با کپی کردن این تعداد کد را راحت بنویسید ولی این کار در کل کیفیت کدنویسی را پایین می‌آورد. برای نوشتن کدهای بالا استفاده از حلقه‌ها بهتر است. ساختارهای تکرار در سی‌شارپ عبارت‌اند از:

- `while`
- `do while`
- `for`

حلقه While

ابتدایی‌ترین ساختار تکرار در سی‌شارپ حلقه `While` است. ابتدا یک شرط را مورد بررسی قرار می‌دهد و تا زمانیکه شرط برقرار باشد کدهای درون بلوک اجرا می‌شوند. ساختار حلقه `while` به صورت زیر است:

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

می‌بینید 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 می‌شوند. اگر شرط غلط یا `false` باشد وقتی که برنامه به حلقه `while` برسد هیچکدام از کدها را اجرا نمی‌کند. 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 می‌گیرد 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  }
```

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

برنامه بالا ۱۰ بار پیغام Hello World! را چاپ می‌کند. اگر از حلقه در مثال بالا استفاده نمی‌کردیم مجبور بودیم تمام ۱۰ خط را تایپ کنیم. اجازه دهید که نگاهی به کدهای برنامه فوق بیندازیم. ابتدا در خط ۷ یک متغیر تعریف و از آن به عنوان شمارنده حلقه استفاده شده است. سپس به آن مقدار ۱ را اختصاص می‌دهیم چون اگر مقدار نداشته باشد نمی‌توان در شرط از آن استفاده کرد. در خط ۹ حلقه While را وارد می‌کنیم. در حلقه while ابتدا مقدار اولیه شمارنده با ۱۰ مقایسه می‌شود که آیا از ۱۰ کمتر است یا با آن برابر است. نتیجه هر بار مقایسه ورود به بدنه حلقه while و چاپ پیغام است.

همانطور که مشاهده می‌کنید بعد از هر بار مقایسه مقدار شمارنده یک واحد اضافه می‌شود (خط ۱۲). حلقه تا زمانی تکرار می‌شود که مقدار شمارنده از ۱۰ کمتر باشد. اگر مقدار شمارنده ۱ بماند و آن را افزایش ندهیم و یا مقدار شرط هرگز false نشود یک حلقه بینهایت به وجود می‌آید. به این نکته توجه کنید که در شرط بالا به جای علامت > از => استفاده شده است. اگر از علامت > استفاده می‌کردیم کد ما ۹ بار تکرار می‌شد چون مقدار اولیه ۱ است و هنگامی که شرط به ۱۰ برسد false می‌شود چون 10 > 10 نیست. اگر می‌خواهید 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 می‌شود و سپس شرط مورد بررسی قرار می‌گیرد. ساختار حلقه do while به صورت زیر است:

```
do
{
   code to repeat;
}
while(condition);
```

همانطور که مشاهده می‌کنید شرط در آخر ساختار قرار دارد. این بدین معنی است که کدهای داخل بدنه حداقل یکبار اجرا می‌شوند. برخلاف حلقه while که اگر شرط نادرست باشد، دستورات داخل بدنه اجرا نمی‌شوند. یکی از موارد 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 می‌کنید 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 می‌دهد و فقط دارای چند خصوصیت اضافی است.

ساختار حلقه for به صورت زیر است:

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

مقدار اولیه (initialization) اولین مقداری است که به شمارنده حلقه می‌دهیم. شمارنده فقط در داخل حلقه for قابل دسترسی است. شرط (condition) در اینجا مقدار شمارنده را با یک مقدار دیگر مقایسه می‌کند و تعیین می‌کند که حلقه ادامه یابد یا نه. عملگر (operation) که مقدار اولیه متغیر را کاهش یا افزایش می‌دهد. 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);
            }
        }
    }
}
```

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

برنامه بالا اعداد ۱ تا ۱۰ را با استفاده از حلقه for می‌شمارد. ابتدا یک متغیر به عنوان شمارنده تعریف می‌کنیم و آن را با مقدار ۱ مقدار دهی اولیه می‌کنیم. سپس با استفاده از شرط آن را با مقدار ۱۰ مقایسه می‌کنیم که آیا کمتر است یا مساوی؟ توجه کنید که قسمت سوم حلقه (i++) فوراً اجرا نمی‌شود. کد اجرا می‌شود و ابتدا رشته Number و سپس مقدار جاری i یعنی ۱ را چاپ می‌کند. آنگاه یک واحد به مقدار i اضافه شده و مقدار i برابر ۲ می‌شود و بار دیگر i با عدد ۱۰ مقایسه می‌شود و این حلقه تا زمانی که مقدار شرط true شود ادامه می‌یابد. حال اگر بخواهید معکوس برنامه بالا را پیاده سازی کنید یعنی اعداد از بزرگ به کوچک چاپ شوند باید به صورت زیر عمل کنید:

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

کد بالا اعداد را از ۱۰ به ۱ چاپ می‌کند (از بزرگ به کوچک). مقدار اولیه شمارنده را ۱۰ می‌دهیم و با استفاده از عملگر کاهش (--) برنامه‌ای که شمارش معکوس را انجام می‌دهد ایجاد می‌کنیم. می‌توان قسمت شرط و عملگر را به صورت‌های دیگر نیز تغییر داد. به عنوان مثال می‌توان از عملگرهای منطقی در قسمت شرط و از عملگرهای تخصیصی در قسمت عملگر افزایش یا کاهش استفاده کرد. همچنین می‌توانید 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 می‌کنید باید آنها را با استفاده از کاما از هم جدا کنید.

حلقه‌های تو در تو (Nested Loops)

سی‌شارپ به شما اجازه می‌دهد که از حلقه‌ها به صورت تو در تو استفاده کنید. اگر یک حلقه در داخل حلقه دیگر قرار بگیرد، به آن حلقه تو در تو گفته می‌شود. در این نوع حلقه‌ها، به ازای اجرای یک بار حلقه بیرونی، حلقه داخلی به طور کامل اجرا می‌شود. در زیر نحوه ایجاد حلقه تو در تو آمده است:

```
for (init; condition; increment)
{
   for (init; condition; increment)
   {
      //statement(s);
   }
   //statement(s);
}
```

```
while(condition)
{
   while(condition)
   {
      //statement(s);
   }
   //statement(s);
}
```

```
do
{
   //statement(s);
   do
   {
      //statement(s);
   }
   while(condition);

}
while(condition);
```

نکته ای که در مورد حلقه‌های تو در تو وجود دارد این است که می‌توان از یک نوع حلقه در داخل نوع دیگر استفاده کرد. مثلاً می‌توان از حلقه for در داخل حلقه while استفاده نمود. در مثال زیر نحوه استفاده از این حلقه‌ها ذکر شده است. فرض کنید که می‌خواهید یک مستطیل با ۳ سطر و ۵ ستون ایجاد کنید:

```
1   using System;
2
3   namespace NestedLoopsDemo
4   {
5       class Program
6       {
7           static void Main(string[] args)
8           {
9               for (int i = 1; i <= 4; i++)
10              {
11                  for (int j = 1; j <= 5; j++)
```

```
12                  {
13                      Console.Write(" * ");
14                  }
15                  Console.Write("\n");
16              }
17          }
18      }
19  }
```

```
 * * * * *
 * * * * *
 * * * * *
 * * * * *
```

در کد بالا به ازای یک بار اجرای حلقه for اول (خط ۹)، حلقه for دوم (۱۴-۱۱) به طور کامل اجرا می‌شود. یعنی وقتی مقدار i برابر عدد ۱ می‌شود، علامت * توسط حلقه دوم ۵ بار چاپ می‌شود، وقتی i برابر ۲ می‌شود، دوباره علامت * پنج بار چاپ می‌شود و .... در کل منظور از دو حلقه for این است که در ۴ سطر علامت * در ۵ ستون چاپ شود یا ۴ سطر ایجاد شود و در هر سطر ۵ بار علامت * چاپ شود. خط ۱۵ هم برای ایجاد خط جدید است. یعنی وقتی حلقه داخلی به طور کامل اجرا شد، یک خط جدید ایجاد می‌شود و علامت‌های * در خطوط جدید چاپ می‌شوند. البته به جای این خط می‌توان Console.WriteLine(); را هم نوشت.

خارج شدن از حلقه با استفاده از break و continue

گاهی اوقات با وجود درست بودن شرط می‌خواهیم حلقه متوقف شود. سؤال 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 می‌توان بخشی از حلقه را رد کرد و به مرحله بعد رفت. برنامه زیر نحوه استفاده از continue و break را نشان می‌دهد:

```
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 حلقه‌های while و do…while استفاده می‌شد، نتیجه یکسانی به دست می‌آمد. همانطور که در شرط برنامه (خط ۱۱) آمده است وقتی که مقدار x به عدد ۵ رسید سپس دستور break اجرا می‌شود (خط ۱۲). حلقه بلافاصله متوقف می‌شود حتی اگر شرط 10 > x برقرار باشد. از طرف دیگر در خط ۲۲ حلقه for فقط برای یک تکرار خاص متوقف شده و سپس ادامه می‌یابد (وقتی مقدار x برابر ۵ شود حلقه از ۵ رد شده و مقدار ۵ را چاپ نمی‌کند و بقیه مقادیر چاپ می‌شوند).

آرایه‌ها

آرایه نوعی متغیر است که لیستی از آدرس‌های مجموعه‌ای از داده‌های هم نوع را در خود ذخیره می‌کند. تعریف چندین متغیر از یک نوع برای هدفی یکسان بسیار خسته کننده است. مثلاً اگر بخواهید صد متغیر از نوع اعداد صحیح تعریف کرده و از آنها استفاده کنید. مطمئناً تعریف این همه متغیر بسیار کسالت آور و خسته کننده است. اما با استفاده از آرایه می‌توان 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];
```

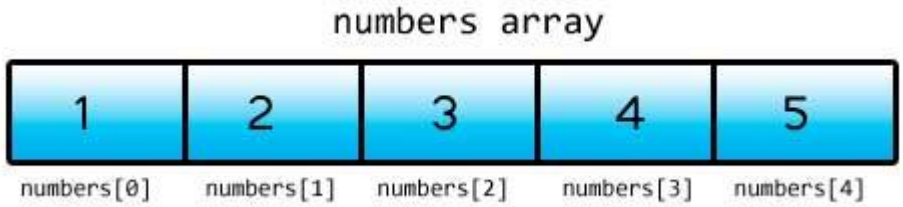
Datatype نوع داده‌هایی را نشان می‌دهد که آرایه در خود ذخیره می‌کند. کروشه که بعد از نوع داده قرار می‌گیرد و نشان دهنده استفاده از آرایه است. arrayName که نام آرایه را نشان می‌دهد. هنگام نامگذاری آرایه بهتر است که نام آرایه نشان دهنده نوع آرایه باشد. به عنوان مثال برای نامگذاری آرایه‌هایی که اعداد را در خود ذخیره می‌کند از کلمه number استفاده کنید. طول آرایه که به کامپایلر می‌گوید 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 می‌شود. برای تعریف یک آرایه که ۵ مقدار از نوع اعداد صحیح در خود ذخیره می‌کند باید به صورت زیر عمل کنیم:

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

در این مثال ۵ آدرس از فضای حافظه کامپیوتر شما برای ذخیره ۵ مقدار رزرو می‌شود. حال چطور مقادیرمان را در هر یک از این آدرس‌ها ذخیره کنیم؟ برای دسترسی و اصلاح مقادیر آرایه از اندیس یا مکان آنها استفاده می‌شود.

```
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 می‌شود. به عنوان مثال شما یک آرایه ۵ عضوی دارید، اندیس آرایه از ۰ تا ۴ می‌باشد چون طول آرایه ۵ است، پس ۱-۵ برابر است با ۴. این بدان معناست که اندیس ۰ نشان دهنده اولین عضو آرایه است و اندیس ۱ 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 برنامه‌نویسی کرده‌اند، معمولاً در گذاشتن اندیس دچار اشتباه می‌شوند و مثلاً ممکن است در مثال بالا اندیس‌ها را از ۱ شروع کنند. اگر بخواهید به یکی از اجزای آرایه با استفاده از اندیسی دسترسی پیدا کنید که در محدوده اندیس‌های آرایه شما نباشد با پیغام خطای IndexOutOfRangeException مواجه می‌شوید و بدین معنی است که شما آدرسی را می‌خواهید که وجود ندارد. یکی دیگر از راه‌های تعریف سریع و مقدار دهی یک آرایه به صورت زیر است:

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

در این روش شما می‌توانید فوراً 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 می‌دهد. شما می‌توانید 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 ۵ است و ما ۵ مقدار را در آن قرار می‌دهیم. اگر تعداد مقادیری که در آرایه قرار می‌دهیم کمتر یا بیشتر از طول آرایه باشد با خطا مواجه می‌شویم. یکی دیگر از راه‌های تعریف آرایه در زیر آمده است. شما می‌توانید 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 ۱۰ مقدار را به آرایه اختصاص داده‌ایم. نکته اینجاست که طول آرایه را تعریف نکرده‌ایم. در این حالت کامپایلر بعد از شمردن تعداد مقادیر داخل آکولاد طول آرایه را تشخیص می‌دهد. به این نکته توجه کنید که اگر برای آرایه طولی در نظر نگیرید، باید برای آن مقدار تعریف کنید، در غیر این صورت با خطا مواجه می‌شوید:

```
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 می‌توان مقادیر را در داخل آکولاد قرار داد. کامپایلر به صورت اتوماتیک با شمارش مقادیر، طول آرایه را تشخیص می‌دهد.

دستیابی به مقادیر آرایه با استفاده از حلقه for

در زیر مثالی در مورد استفاده از آرایه‌ها آمده است. در این برنامه ۵ مقدار از کاربر گرفته شده و میانگین آنها حساب می‌شود:

```
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  }
```

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

در خط ۷ یک آرایه تعریف شده است که می‌تواند ۵ عدد صحیح را در خود ذخیره کند. خطوط ۸ و ۹ متغیرهایی تعریف شده‌اند که از آنها برای محاسبه میانگین و جمع کل استفاده می‌شود. توجه کنید که مقدار اولیه total صفر است تا از بروز خطا هنگام اضافه شدن مقدار به آن جلوگیری شود. در خطوط ۱۱ تا ۱۵ حلقه for برای تکرار و گرفتن ورودی از کاربر تعریف شده است. از خاصیت طول (Length) آرایه برای تشخیص تعداد اجزای آرایه استفاده می‌شود. اگر چه می‌توانستیم به سادگی در حلقه for مقدار ۵ را برای شرط قرار دهیم، ولی استفاده از خاصیت Length آرایه کار راحت تری است و می‌توانیم طول آرایه را تغییر دهیم و شرط حلقه for با تغییر جدید هماهنگ شود. در خط ۱۴ ورودی دریافت شده از کاربر به نوع int تبدیل و در آرایه ذخیره می‌شود. اندیس استفاده شده در number (خط ۱۴) مقدار i جاری در حلقه است. برای مثال در ابتدای حلقه

مقدار i صفر است، بنابراین وقتی در خط ۱۴ اولین داده از کاربر گرفته می‌شود، اندیس آن برابر صفر می‌شود. در تکرار بعدی i یک واحد اضافه می‌شود و در نتیجه در خط ۱۴ و بعد از ورود دومین داده توسط کاربر اندیس آن برابر یک می‌شود. این حالت تا زمانی که شرط در حلقه for برقرار است ادامه می‌یابد. در خطوط ۲۰-۱۷ از حلقه for دیگر برای دسترسی به مقدار هر یک از داده‌های آرایه استفاده شده است. در این حلقه نیز مانند حلقه قبل از مقدار متغیر شمارنده به عنوان اندیس استفاده می‌کنیم.

هر یک از اجزای عددی آرایه به متغیر total اضافه می‌شوند. بعد از پایان حلقه می‌توانیم میانگین اعداد را حساب کنیم (خط ۲۲). مقدار total یا جمع کل را بر تعداد اجزای آرایه (تعداد عددها) تقسیم می‌کنیم. برای دسترسی به تعداد اجزای آرایه می‌توان از خاصیت length آرایه استفاده کرد. توجه کنید که در اینجا ما مقدار خاصیت length را به نوع double تبدیل کرده‌ایم، بنابراین نتیجه عبارت یک مقدار از نوع double خواهد شد و دارای بخش کسری می‌باشد. حال اگر عملوندهای تقسیم را به نوع double تبدیل نکنیم، نتیجه تقسیم یک عدد از نوع صحیح خواهد شد و دارای بخش کسری نیست. خط ۲۴ مقدار میانگین را در صفحه نمایش چاپ می‌کند. طول آرایه بعد از مقدار دهی نمی‌تواند تغییر کند. به عنوان مثال اگر یک آرایه را که شامل ۵ جزء است مقدار دهی کنید دیگر نمی‌توانید آن را مثلاً به ۱۰ جزء تغییر اندازه دهید. البته تعداد خاصی از کلاس‌ها مانند آرایه‌ها عمل می‌کنند و توانایی تغییر تعداد اجزای تشکیل دهنده خود را دارند. آرایه‌ها 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 foreach یکی دیگر از ساختارهای تکرار در سی‌شارپ می‌باشد که مخصوصاً برای آرایه‌ها، لیست‌ها و مجموعه‌ها طراحی شده است. حلقه foreach با هر بار گردش در بین اجزاء، مقادیر هر یک از آنها را در داخل یک متغیر موقتی قرار می‌دهد و شما می‌توانید 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 می‌کند.temporaryVar باید دارای نوع باشد تا بتواند مقادیر آرایه را در خود ذخیره کند. به عنوان مثال اگر 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 می‌نویسیم. در زیر نحوه استفاده از حلقه foreach آمده است:

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           int[] numbers = { 1, 2, 3, 4, 5 };
8
9           foreach (int n in numbers)
10          {
11              Console.WriteLine("Number {0}", n);
```

```
        }
    }
}
```

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

در برنامه بالا آرایه ای با ۵ جزء تعریف شده و مقادیر ۱ تا ۵ در آنها قرار داده شده است (خط ۷). در خط ۹ حلقه foreach شروع می‌شود. ما یک متغیر موقتی تعریف کرده‌ایم که اعداد آرایه را در خود ذخیره می‌کند. در هر بار تکرار از حلقه foreach متغیر موقتی n، مقادیر عددی را از آرایه استخراج می‌کند. حلقه foreach مقادیر اولین تا آخرین جزء آرایه را در اختیار ما قرار می‌دهد.

حلقه foreach برای دریافت هر یک از مقادیر آرایه کاربرد دارد. بعد از گرفتن مقدار یکی از اجزای آرایه، مقدار متغیر موقتی را چاپ می‌کنیم (خط ۱۱). حلقه foreach یک ضعف دارد و آن این است که این حلقه ما را قادر می‌سازد که به داده‌ها دسترسی یابیم و یا آنها را بخوانیم ولی اجازه اصلاح اجزاء آرایه را نمی‌دهد. برای درک این مطلب در مثال زیر سعی شده است که مقدار هر یک از اجزا آرایه یک واحد افزایش یابد:

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

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

اگر برنامه را اجرا کنید با خطا مواجه می‌شوید. برای اصلاح هر یک از اجزا آرایه می‌توان از حلقه for استفاده کرد.

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

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

آرایه‌های چند بعدی

آرایه‌های چند بعدی آرایه‌هایی هستند که برای دسترسی به هر یک از عناصر آنها باید از چندین اندیس استفاده کنیم. یک آرایه چند بعدی را می‌توان مانند یک جدول با تعدادی ستون و ردیف تصور کنید. با افزایش اندیس‌ها اندازه ابعاد آرایه نیز افزایش می‌یابد و آرایه‌های چند بعدی با بیش از دو اندیس به وجود می‌آیند. 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 می‌شود:

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

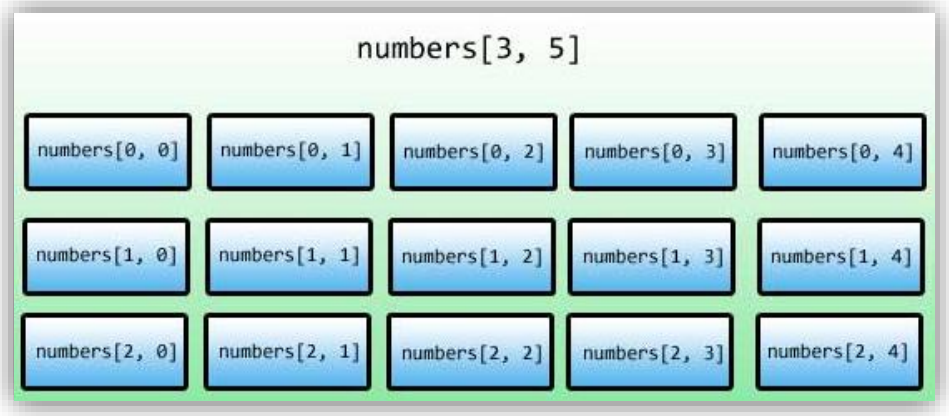
می‌توان یک آرایه با تعداد زیادی بعد ایجاد کرد به شرطی که هر بعد دارای طول مشخصی باشد. به دلیل اینکه آرایه‌های سه بعدی یا آرایه‌های با بیشتر از دو بعد بسیار کمتر مورد استفاده قرار می‌گیرند اجازه بدهید که در این درس بر روی آرایه‌های دو بعدی تمرکز کنیم. در تعریف این نوع

آرایه، ابتدا نوع آرایه یعنی اینکه آرایه چه نوعی از انواع داده را در خود ذخیره می‌کند را مشخص می‌کنیم. سپس یک جفت کروشه و در داخل کروشه‌ها یک کاما قرار می‌دهیم.

به تعداد کاماهایی که در داخل کروشه می‌گذارید، توجه کنید. اگر آرایه ما دو بعدی است باید ۱ کاما و اگر سه بعدی است باید ۲ کاما قرار دهیم. سپس یک نام برای آرایه انتخاب کرده و بعد تعریف آنرا با گذاشتن کلمه new، نوع داده و طول آن کامل می‌کنیم. 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 می‌توان 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 می‌گوید که فضای کافی به عناصر آرایه اختصاص بده (در این مثال ۱۵ خانه). در شکل زیر مکان هر عنصر در یک آرایه دو بعدی نشان داده شده است.



مقدار ۳ را به x، چون ۳ سطر و مقدار ۵ را به Y چون ۵ ستون داریم، اختصاص می‌دهیم. چطور یک آرایه چند بعدی را مقدار دهی کنیم؟ چند راه برای مقدار دهی به آرایه‌ها وجود دارد.

```
datatype[,] arrayName = new datatype[x, y] { { r0c0, r0c1, ... r0cY },
                                             { r1c0, r1c1, ... r1cY },
                                                       .
                                                       .
                                                       .
                                             { rXc0, rXc1, ... rXcY } };
```

برای راحتی کار می‌توان از نوشتن قسمت new datatype[,] صرف نظر کرد.

```
datatype[,] arrayName = { { r0c0, r0c1, ... r0cY },
                          { r1c0, r1c1, ... r1cY },
                                    .
                                    .
                                    .
                          { rXc0, rXc1, ... rXcY } };
```

به عنوان مثال:

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

و یا می‌توان 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 می‌کنید برای دسترسی به هر یک از عناصر در یک آرایه دو بعدی به سادگی می‌توان از اندیس‌های X و Y و یک جفت کروشه مانند مثال استفاده کرد.

گردش در میان عناصر آرایه‌های چند بعدی

گردش در میان عناصر آرایه‌های 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.

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           int[,] numbers = { { 1, 2, 3, 4, 5 },
8                              { 6, 7, 8, 9, 10 },
9                              { 11, 12, 13, 14, 15 }
10                           };
11
12          foreach (int number in numbers)
13          {
14              Console.Write(number + " ");
15          }
16      }
17  }
```

```
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 نمی‌توانیم انتهای ردیف‌ها را مشخص کنیم. برنامه زیر نشان می‌دهد که چطور از حلقه for برای خواندن همه مقادیر آرایه و تعیین انتهای ردیف‌ها استفاده کنید.

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
```

```
7           int[,] numbers = { { 1, 2, 3, 4, 5 },
8                              { 6, 7, 8, 9, 10 },
9                              { 11, 12, 13, 14, 15 }
10                           };
11
12          for (int row = 0; row < numbers.GetLength(0); row++)
13          {
14              for (int col = 0; col < numbers.GetLength(1); col++)
15              {
16                  Console.Write(numbers[row, col] + " ");
17              }
18
19              //Go to the next line
20              Console.WriteLine();
21          }
22      }
23  }
```

```
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 نمی‌توان به مقادیر دسترسی یافت بلکه به یک حلقه for تو در تو نیاز داریم. در اولین حلقه for (خط ۱۲) یک متغیر تعریف شده است که در میان ردیف‌های آرایه (row) گردش می‌کند. این حلقه تا زمانی ادامه می‌یابد که مقدار ردیف کمتر از طول اولین بعد باشد. در این مثال از متد GetLength() کلاس Array استفاده کرده‌ایم. این متد طول آرایه را در یک بعد خاص نشان می‌دهد و دارای یک پارامتر است که همان بعد آرایه می‌باشد. به عنوان مثال برای به دست آوردن طول اولین بعد آرایه مقدار صفر را به این متد ارسال می‌کنیم چون شمارش ابعاد یک آرایه از صفر تا یک واحد کمتر از تعداد ابعاد انجام می‌شود.

در داخل اولین حلقه for حلقه for دیگری تعریف شده است (خط ۱۴). 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 ۱ را به آن ارسال می‌کنیم 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 می‌شود. سپس مقدار هر عنصر از آرایه را با استفاده از حلقه نشان می‌دهیم، اگر مقدار ردیف (row) برابر ۰ و مقدار ستون (col) برابر ۰ باشد مقدار عنصری که در ستون ۱ و ردیف ۱ (numbers[0, 0]) قرار دارد نشان داده خواهد شد که در مثال بالا عدد ۱ است.

بعد از اینکه دومین حلقه تکرار به پایان رسید، فوراً دستورات بعد از آن اجرا خواهند شد، که در اینجا دستور Console.WriteLine() که به برنامه اطلاع می‌دهد که به خط بعد برود. سپس حلقه با اضافه کردن یک واحد به مقدار row این فرایند را دوباره تکرار می‌کند.

سپس دومین حلقه for اجرا شده و مقادیر دومین ردیف نمایش داده می‌شود. این فرایند تا زمانی اجرا می‌شود که مقدار row کمتر از طول اولین بعد باشد. حال بیایید آنچه را از قبل یاد گرفته‌ایم در یک برنامه به کار بریم. این برنامه نمره چهار درس مربوط به سه دانش آموز را از ما می‌گیرد و معدل سه دانش آموز را حساب می‌کند.

```
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] = Convert.ToDouble(Console.ReadLine());
20                  total += studentGrades[student, grade];
21              }
22
23              Console.WriteLine("Average is {0:F2}", (total / studentGrades.GetLength(1)));
24              Console.WriteLine();
25          }
26      }
27  }
```

```
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 ۷). همچنین یک متغیر به نام total تعریف می‌کنیم که مقدار محاسبه شده معدل هر دانش آموز را در آن قرار دهیم. حال وارد حلقه for تو در تو می‌شویم (خط ۱۰). در اولین حلقه for یک متغیر به نام sudent برای تشخیص پایه درسی هر دانش آموز تعریف کرده‌ایم. از متد GetLength() هم برای تشخیص تعداد دانش آموزان استفاده شده است. وارد بدنه حلقه for می‌شویم. در خط ۱۲ مقدار متغیر total را برابر صفر قرار می‌دهیم. بعداً مشاهده می‌کنید که چرا این کار را انجام دادیم. سپس برنامه یک پیغام را نشان می‌دهد و از شما می‌خواهد که شماره دانش آموز را وارد کنید (student + 1). عدد ۱ را به student اضافه کرده‌ایم تا به جای نمایش Student 0، با Student 1 شروع شود، تا طبیعی تر به نظر برسد.

سپس به دومین حلقه for در خط ۱۶ می‌رسیم. در این حلقه یک متغیر شمارنده به نام grade تعریف می‌کنیم که طول دومین بعد آرایه را با استفاده از فراخوانی متد GetLength(1) به دست می‌آورد. این طول تعداد نمراتی را که برنامه از سؤال می‌کند را نشان می‌دهد. برنامه چهار نمره

مربوط به دانش آموز را می‌گیرد. هر وقت که برنامه یک نمره را از کاربر دریافت می‌کند، نمره به متغیر total اضافه می‌شود. وقتی همه نمره‌ها وارد شدند، متغیر total هم جمع همه نمرات را نشان می‌دهد. در خطوط ۲۴-۲۳ معدل دانش آموز نشان داده می‌شود. به فرمت {0:F2} توجه کنید. این فرمت معدل را تا دو رقم اعشار نشان می‌دهد. معدل از تقسیم کردن total (جمع) بر تعداد نمرات به دست می‌آید. از متد GetLength(1) هم برای به دست آوردن تعداد نمرات استفاده می‌شود.

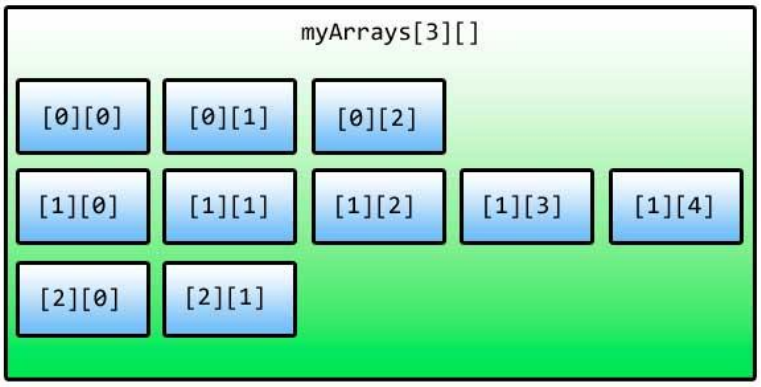
آرایه‌های دندانه دار

آرایه‌های دندانه دار نوعی از آرایه‌های چند بعدی هستند که شامل ردیف‌هایی با تعداد ستون‌های مختلف‌اند. آرایه چند بعدی ساده، آرایه ای به شکل مستطیل است، چون تعداد ستون‌های آن یکسان است ولی آرایه دندانه دار دارای سطرهایی با تعداد ستون‌های متفاوت است. بنابراین می‌توان یک آرایه دندانه دار را آرایه ای از آرایه‌ها فرض کرد. در زیر نحوه تعریف آرایه‌های چند بعدی آمده است.

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

مقدار دهی به آرایه‌های 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 می‌کنیم، و بعد نوع داده‌ای آرایه و سپس دو جفت کروشه که در جفت کروشه اول تعداد ردیف‌ها قرار دارد را تعریف می‌کنیم. حال تعداد ستون‌های هر ردیف را با استفاده از سه ردیفی که در دسترس است و با استفاده از اندیس‌های آنها مانند یک آرایه ساده مقدار دهی می‌کنیم. می‌توان به ستون‌های هر ردیف مجموعه‌ای 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 آرایه‌های 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 می‌توان 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 می‌توان 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 می‌توان از ستون‌ها و ردیف‌های آن استفاده کرد:

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

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

از یک حلقه foreach ساده نمی‌توان برای دسترسی به اجزای این آرایه‌ها استفاده کرد.

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

اگر از حلقه foreach استفاده کنیم با خطا مواجه می‌شویم، چون عناصر این نوع آرایه‌ها، 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 ردیف‌های آرایه را به دست می‌آوریم. در حلقه for دوم نیز با استفاده از خاصیت Length عنصر ردیف جاری تعداد ستون‌ها را به دست می‌آوریم. سپس با استفاده از اندیس، عناصر آرایه را چاپ می‌کنیم.

متدها

متدها به شما اجازه می‌دهند که یک رفتار یا وظیفه را تعریف کنید و مجموعه‌ای از کدها هستند که در هر جای برنامه می‌توان از آنها استفاده کرد. متدها دارای آرگومانهایی هستند که وظیفه متد را مشخص می‌کنند. متد در داخل کلاس تعریف می‌شود. نمی‌توان یک متد را در داخل متد دیگر تعریف کرد. وقتی که شما در برنامه یک متد را صدا می‌زنید برنامه به قسمت تعریف متد رفته و کدهای آن را اجرا می‌کند. در سی‌شارپ متدی وجود دارد که نقطه آغاز هر برنامه است و بدون آن برنامه‌ها نمی‌دانند باید از کجا شروع شوند، این متد `Main()` نام دارد. پارامترها همان چیزهایی هستند که متد منتظر دریافت آنها است. آرگومانها مقادیری هستند که به پارامترها ارسال می‌شوند. گاهی اوقات دو کلمه پارامتر و آرگومان به یک منظور به کار می‌روند. ساده‌ترین 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 }
```

```
Hello World!
```

در خطوط ۸-۵ یک متد تعریف کرده‌ایم. مکان تعریف آن در داخل کلاس مهم نیست. به عنوان مثال می‌توانید آن را زیر متد `Main()` تعریف کنید. می‌توان این متد را در داخل متد دیگر صدا زد (فراخوانی کرد). متد دیگر ما در اینجا متد `Main()` است که می‌توانیم در داخل آن نام متدی که برای چاپ یک پیغام تعریف کرده‌ایم (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 می‌توان از متد استفاده کرد بدون اینکه از کلاس نمونه‌ای ساخته شود. متد `Main()` همواره باید به صورت `static` تعریف شود چون برنامه فوراً و بدون نمونه سازی از کلاس از آن استفاده می‌کند. وقتی به مبحث برنامه‌نویسی شیء گرا رسیدید به طور دقیق کلمه `static` مورد بحث قرار می‌گیرد. برنامه `class` (مثال بالا) زمانی اجرا می‌شود که برنامه دو متدی را که تعریف کرده‌ایم را اجرا کند و متد `Main()` به صورت `static` تعریف شود. درباره این کلمه کلیدی در درس‌های آینده مطالب بیشتری می‌آموزیم. 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 می‌شود) استفاده کرده‌ایم. این روش نامگذاری قراردادی است و می‌توان از این روش استفاده نکرد، اما پیشنهاد می‌شود که از این روش برای تشخیص متدها استفاده کنید. بهتر است در نامگذاری متدها از کلماتی استفاده شود که کار آن متد را مشخص می‌کند مثلاً نام‌هایی مانند GoToBed یا OpenDoor.

همچنین به عنوان مثال اگر مقدار برگشتی متد یک مقدار بولی باشد، می‌توانید اسم متد خود را به صورت یک کلمه سوالی انتخاب کنید، مانند IsLeapyear یا IsTeenager ... . ولی از گذاشتن علامت سؤال در آخر اسم متد خودداری کنید. دو پرانتزی که بعد از نام می‌آید نشان دهنده آن است که نام متعلق به یک متد است. در این مثال در داخل پرانتزها هیچ چیزی نوشته نشده چون پارامتری ندارد. در درس‌های آینده در مورد متدها بیشتر توضیح می‌دهیم. بعد از پرانتزها دو آکولاد قرار می‌دهیم که بدنه متد را تشکیل می‌دهد و کدهایی را که می‌خواهیم اجرا شوند را در داخل این آکولادها می‌نویسیم.

در داخل متد Main()، متدی که در خط ۱۲ ایجاد کرده‌ایم را صدا می‌زنیم. 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 آرگومانها را به ترتیب در داخل پرانتزها قرار دهید. در این مورد نیز در درس‌های آینده توضیح بیشتری می‌دهیم. با صدا زدن یک متد کدهای داخل بدنه آن اجرا می‌شوند. برای اجرای متد PrintMessage() برنامه از متد Main() به محل تعریف متد PrintMessage() می‌رود. مثلاً وقتی ما متد PrintMessage() را در خط ۱۲ صدا می‌زنیم، برنامه از خط ۱۲ به خط ۷، یعنی جایی که متد تعریف شده می‌رود. اکنون ما یک متد در کلاس Program داریم که همه متدهای این کلاس می‌توانند آن را صدا بزنند.

مقدار برگشتی از یک متد

متدها می‌توانند مقدار برگشتی از هر نوع داده‌ای داشته باشند. این مقادیر می‌توانند 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 می‌زنید و از او می‌خواهید 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 داده‌ای مقدار برگشتی را مشخص می‌کند (int، bool، ...). در داخل بدنه متد کلمه کلیدی return و بعد از آن یک مقدار یا عبارتی که نتیجه آن یک مقدار است، را می‌نویسیم. 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 می‌دهد.

```
1  using System;
2
3  public class Program
```

```
{
    static int CalculateSum()
    {
        int firstNumber = 10;
        int secondNumber = 5;

        int sum = firstNumber + secondNumber;

        return sum;
    }

    public static void Main()
    {
        int result = CalculateSum();

        Console.WriteLine("Sum is {0}.", result);
    }
}
```

```
Sum is 15.
```

همانطور که در خط ۵ مثال فوق مشاهده می‌کنید هنگام تعریف متد از کلمه `int` به جای `void` استفاده کرده‌ایم که نشان دهنده آن است که متد ما دارای مقدار برگشتی از نوع اعداد صحیح است. در خطوط ۷ و ۸ دو متغیر تعریف و مقدار دهی شده‌اند. 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 شده‌اند قابل استفاده هستند. در خط ۱۰ جمع دو متغیر در متغیر `sum` قرار می‌گیرد. در خط ۱۲ مقدار برگشتی `sum` توسط دستور `return` فراخوانی می‌شود.

در داخل متد `Main()` یک متغیر به نام `result` در خط ۱۷ تعریف می‌کنیم و متد `CalculateSum()` را فراخوانی می‌کنیم. متد `CalculateSum()` مقدار ۱۵ را بر می‌گرداند که این مقدار در داخل متغیر `result` ذخیره می‌شود. در خط ۱۹ مقدار ذخیره شده در متغیر `result` چاپ می‌شود. 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 ۱۵ است، در حالیکه می‌توانستیم به راحتی یک متغیر تعریف کرده و مقدار ۱۵ را به آن اختصاص دهیم. این متد در صورتی کارآمد است که پارامترهایی به آن اضافه شود که در درس‌های آینده توضیح خواهیم داد. هنگامی که می‌خواهیم 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:

```
using System;

public class Program
{
    static int GetNumber()
    {
        int number;

        Console.Write("Enter a number greater than 10: ");
        number = Convert.ToInt32(Console.ReadLine());

        if (number > 10)
        {
            return number;
        }
        else
        {
            return 0;
```

```
        }
    }

    public static void Main()
    {
        int result = GetNumber();

        Console.WriteLine("Result = {0}.", result);
    }
}
```

```
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 ۱۰ را می‌خواهد. اگر عدد وارد شده توسط کاربر درست نباشد متد مقدار صفر را بر می‌گرداند و اگر قسمت else دستور if و یا دستور return را از آن حذف کنیم، در هنگام اجرای برنامه با پیغام خطا مواجه می‌شویم. چون اگر شرط دستور if نادرست باشد (کاربر مقداری کمتر از ۱۰ را وارد کند) برنامه به قسمت else می‌رود تا مقدار صفر را برگرداند و چون قسمت else حذف شده است برنامه با خطا مواجه می‌شود و همچنین اگر دستور return حذف شود، چون برنامه نیاز به مقدار برگشتی دارد، پیغام خطا می‌دهد. و آخرین مطلبی که در این درس می‌خواهیم به شما آموزش دهیم این است که شما می‌توانید از یک متد که مقدار برگشتی ندارد، خارج شوید. حتی اگر از نوع داده‌ای void در یک متد استفاده می‌کنید، باز هم می‌توانید کلمه کلیدی return را در آن به کار ببرید. استفاده از return باعث خروج از بدنه متد و اجرای کدهای بعد از آن می‌شود.

```
using System;

public class Program
{
    static void TestReturnExit()
    {
        Console.WriteLine("Line 1 inside the method TestReturnExit()");
        Console.WriteLine("Line 2 inside the method TestReturnExit()");

        return;

        //The following lines will not execute
        Console.WriteLine("Line 3 inside the method TestReturnExit()");
        Console.WriteLine("Line 4 inside the method TestReturnExit()");
    }

    public static void Main()
    {
        TestReturnExit();
        Console.WriteLine("Hello World!");
    }
}
```

```
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 می‌شود.

پارامترها و آرگومانها

پارامترها داده‌های خامی هستند که متد آنها را پردازش می‌کند و سپس اطلاعاتی را که به دنبال آن هستید در اختیار شما قرار می‌دهد. فرض کنید پارامترها مانند اطلاعاتی هستند که شما به یک کارمند می‌دهید، که بر طبق آنها کارش را به پایان برساند. یک متد می‌تواند هر تعداد پارامتر داشته باشد. هر پارامتر می‌تواند 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 می‌گیرند. بر اساس کاری که متد انجام می‌دهد می‌توان تعداد پارامترهای زیادی به متد اضافه کرد. بعد از فراخوانی یک متد باید آرگومانهای آن را نیز تأمین کنید. آرگومانها مقادیری هستند که به پارامترها اختصاص داده می‌شوند. ترتیب ارسال آرگومانها به پارامترها مهم است. عدم رعایت ترتیب در ارسال آرگومانها باعث به وجود آمدن خطای منطقی و خطای زمان اجرا می‌شود. اجازه بدهید که یک مثال بزنیم:

```
1   using System;
2
3   public class Program
4   {
5       static int CalculateSum(int number1, int number2)
6       {
7           return number1 + number2;
8       }
9
10      public static void Main()
11      {
12          int num1, num2;
13
14          Console.Write("Enter the first number: ");
15          num1 = Convert.ToInt32(Console.ReadLine());
16          Console.Write("Enter the second number: ");
17          num2 = Convert.ToInt32(Console.ReadLine());
18
19          Console.WriteLine("Sum = {0}", CalculateSum(num1, num2));
20      }
21  }
```

```
Enter the first number: 10
Enter the second number: 5
Sum = 15
```

در برنامه بالا یک متد به نام CalculateSum() (خطوط ۸-۵) تعریف شده است، که وظیفه آن جمع مقدار دو عدد است. چون این متد مقدار دو عدد صحیح را با هم جمع می‌کند پس نوع برگشتی ما نیز باید int باشد. متد دارای دو پارامتر است که اعداد را به آنها ارسال می‌کنیم. به نوع داده‌ای پارامترها توجه کنید. هر دو پارامتر یعنی number1 و number2 مقادیری از نوع اعداد صحیح (int) دریافت می‌کنند. در بدنه متد دستور return نتیجه جمع دو عدد را بر می‌گرداند. در داخل متد Main() برنامه از کاربر دو مقدار را درخواست می‌کند و آنها را داخل متغیرها قرار می‌دهد. حال متد را که آرگومانهای آن را آماده کرده‌ایم فراخوانی می‌کنیم. مقدار num1 به پارامتر اول و مقدار num2 به پارامتر دوم ارسال می‌شود.

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 دقیقاً با ترتیب قرارگیری پارامترهای تعریف شده در متد مطابقت داشته باشد. بعد از ارسال مقادیر ۱۰ و ۵ به پارامترها، پارامترها آنها را دریافت می‌کنند. به این نکته نیز توجه کنید که نام پارامترها طبق قرارداد به شیوه کوهان شتری یا camelCasing (بجز کلمه اول بقیه کلمات با حرف بزرگ شروع می‌شوند) نوشته می‌شود. در داخل بدنه متد (خط ۷) دو مقدار با هم جمع می‌شوند و نتیجه به متد فراخوان (متدی که متد ( ) CalculateSum را فراخوانی می‌کند) ارسال می‌شود.

در درس آینده از یک متغیر برای ذخیره نتیجه محاسبات استفاده می‌کنیم، ولی در اینجا مشاهده می‌کنید که می‌توان به سادگی نتیجه جمع را نشان داد (خط ۷). در داخل متد Main() از ما دو عدد که قرار است با هم جمع شوند درخواست می‌شود. در خط ۱۹ متد CalculateSum() را فراخوانی می‌کنیم و دو مقدار صحیح به آن ارسال می‌کنیم. دو عدد صحیح در داخل متد با هم جمع شده و نتیجه آنها برگردانده می‌شود. مقدار برگشت داده شده از متد به وسیله متد WriteLine() از کلاس Console نمایش داده می‌شود (خط ۱۹). در برنامه زیر یک متد تعریف شده است که دارای دو پارامتر از دو نوع داده‌ای 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  }
```

```
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 می‌کند. متد به سادگی دو مقداری که به آن ارسال شده است را نشان می‌دهد. در خط ۱۳ متد را اول با یک رشته و سپس یک عدد خاص فراخوانی می‌کنیم. حال اگر متد به صورت زیر فراخوانی می‌شد:

```
ShowMessageAndNumber(100, "Welcome to Gimme C#!");
```

در برنامه خطا به وجود می‌آمد، چون عدد ۱۰۰ به پارامتری از نوع رشته و رشته‌ی Hello World! به پارامتری از نوع اعداد صحیح ارسال می‌شود. این نشان می‌دهد که ترتیب ارسال آرگومانها به پارامترها هنگام فراخوانی متد مهم است. به مثال ۱ توجه کنید. در آن مثال دو عدد از نوع int به پارامترها ارسال کردیم، که ترتیب ارسال آنها چون هر دو 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 داده‌ای قبول کند، باز هم بهتر است ترتیب بر اساس تعریف پارامترها رعایت شود. به عنوان مثال در اولین فراخوانی متد بالا اشکالی به چشم نمی‌آید، چون سن شخص ۲۰ و قد او ۱۶۰ سانتی متر است. اگر آرگومانها را به ترتیب ارسال نکنیم، سن شخص ۱۶۰ و قد او ۲۰ سانتی متر می‌شود، که به واقعیت نزدیک نیست. دانستن مبانی مقادیر برگشتی و ارسال آرگومانها باعث می‌شود که شما متدهای کارآمدتری تعریف کنید. تکه کد زیر نشان می‌دهد که شما حتی می‌توانید 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 ۵ است و به عنوان آرگومان به متد AnotherMethod() ارسال می‌شود خروجی کد بالا هم عدد ۵ است.

نامیدن آرگومانها

یکی دیگر از راه‌های ارسال آرگومانها استفاده از نام آنهاست. استفاده از نام آرگومانها شما را از به یادآوری و رعایت ترتیب پارامترها هنگام ارسال آرگومانها راحت می‌کند. 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 می‌برد، چون شما می‌توانید ببینید که چه مقادیری به چه پارامترهایی اختصاص داده شده است. نامیدن آرگومانها در سی‌شارپ ۲۰۱۰ مطرح شده است و اگر شما از نسخه‌های قبلی مانند سی‌شارپ ۲۰۰۸ استفاده می‌کنید، نمی‌توانید از این خاصیت استفاده کنید. در زیر نحوه استفاده از نام آرگومانها وقتی که متد فراخوانی می‌شود، 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);
    }
}
```

```
Jack' 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 ۹-۷ از فرمت پول رایج، که با `{0:C}` نشان داده می‌شود، استفاده کرده است که یک داده عددی را به نوع پولی تبدیل می‌کند. خروجی نشان می‌دهد که حتی اگر ما ترتیب آرگومانها در سه متد فراخوانی شده را تغییر دهیم، مقادیر مناسب به پارامترهای مربوطه‌شان اختصاص داده می‌شود. همچنین می‌توان 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.

```
SetSalary(30, andy: 50, mark: 60);


SetSalary(30, mark: 60, andy: 50);


SetSalary(mark: 60, andy: 50, 30);


SetSalary(mark: 60, 30, andy: 50);
```

همانطور که مشاهده می‌کنید ابتدا باید آرگومانهای ثابت هنگام فراخوانی متد ذکر شوند. در اولین و دومین فراخوانی در کد بالا، مقدار ۳۰ را به عنوان اولین آرگومان به اولین پارامتر متد یعنی jack اختصاص می‌دهیم. سومین و چهارمین خط کد بالا اشتباه هستند، چون آرگومانهای دارای نام قبل از آرگومانهای ثابت قرار گرفته‌اند. قرار گرفتن آرگومانهای دارای نام بعد از آرگومانها ثابت، از بروز خطا جلوگیری می‌کند.

ارسال آرگومانها به روش ارجاع

آرگومانها را می‌توان به کمک ارجاع ارسال کرد. این بدان معناست که شما آدرس متغیر را ارسال می‌کنید، 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 می‌کنیم، مقدار اصلی آرگومان در خارج از متد هم تغییر می‌کند. در زیر دستورالعمل پایه ای تعریف پارامترها 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 می‌شود و آرگومانها به آنها ارسال می‌شود 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 شده‌اند و آن اضافه کردن عدد ۱۰ به مقداری است که به آنها ارسال می‌شود. اولین متد (خطوط ۹-۵) دارای یک پارامتر است که نیاز به یک مقدار آرگومان (از نوع int) دارد. وقتی که متد را صدا می‌زنیم و آرگومانی به آن اختصاص می‌دهیم (خط ۲۴)، کپی آرگومان به پارامتر متد ارسال می‌شود. بنابراین مقدار اصلی متغیر خارج از متد هیچ ارتباطی به پارامتر متد ندارد. سپس مقدار ۱۰ را به متغیر پارامتر (number) اضافه کرده و نتیجه را چاپ می‌کنیم.

برای اثبات اینکه متغیر num هیچ تغییری نکرده است، مقدار آن را یکبار دیگر چاپ کرده و مشاهده می‌کنیم که تغییری نکرده است. دومین متد (خطوط ۱۵-۱۱) نیاز به یک مقدار با ارجاع دارد. در این حالت به جای اینکه یک کپی از مقدار به عنوان آرگومان به آن ارسال شود، آدرس متغیر به آن ارسال می‌شود. حال پارامتر به مقدار اصلی متغیر که زمان فراخوانی متد به آن ارسال می‌شود، دسترسی دارد. وقتی که ما مقدار پارامتری که شامل آدرس متغیر اصلی است را تغییر می‌دهیم (خط ۱۳)، در واقع مقدار متغیر اصلی در خارج از متد را تغییر داده‌ایم. در نهایت مقدار اصلی متغیر را وقتی که از متد خارج شدیم را نمایش می‌دهیم و مشاهده می‌شود که مقدار آن واقعاً تغییر کرده است.

پارامترهای out

پارامترهای out، پارامترهایی هستند که متغیرهایی که مقدار دهی اولیه نشده‌اند، را قبول می‌کنند. کلمه کلیدی out زمانی مورد استفاده قرار می‌گیرد 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 می‌دهد. ارسال متغیر مقداردهی نشده به متد زمانی مفید است که شما بخواهید از طریق متد، متغیر را مقداردهی کنید. استفاده از کلمه کلیدی out باعث ارسال آرگومان به روش ارجاع می‌شود نه مقدار. به مثال زیر توجه کنید:

```
1   using System;
2
3   public class Program
4   {
5       static void GiveValue(out int number)
6       {
7           number = 10;
8       }
9
10      public static void Main()
11      {
12          //Uninitialized variable
13          int myNumber;
14
15          GiveValue(out myNumber);
16
17          Console.WriteLine("myNumber = {0}", myNumber);
18      }
```

```
19  }
```

```
myNumber = 10
```

از کلمه کلیدی out برای پارامترهای متد استفاده شده است، بنابراین می‌توانند متغیرهای مقداردهی نشده را قبول کنند. در متد Main()، خط ۱۵ متد را فراخوانی می‌کنیم و قبل از آرگومان کلمه کلیدی out را قرار می‌دهیم. متغیر مقداردهی نشده (myNumber) به متد ارسال می‌شود و در آنجا مقدار ۱۰ به آن اختصاص داده می‌شود (خط ۷). مقدار myNumber در خط ۱۷ نمایش داده می‌شود و مشاهده می‌کنید که مقدارش برابر مقداری است که در داخل متد به آن اختصاص داده شده است (یعنی ۱۰). 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 می‌توان به متد ارسال کرد. این کار در حکم استفاده از کلمه کلیدی ref است. تفاوت ref با out این است که کلمه کلیدی ref به کامپایلر می‌گوید که متغیر مقدار دهی اولیه و بعد به متد ارسال شده است ولی out به کامپایلر می‌گوید که متغیر مقدار دهی اولیه نشده و باید در داخل متد مقدار دهی اولیه شود. زمانی که لازم باشد یک متد دارای چندین خروجی باشد از out استفاده می‌کنیم.

ارسال آرایه به عنوان آرگومان

می‌توان آرایه‌ها 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  }
```

```
1
2
3
4
5
```

مشاهده کردید که به سادگی می‌توان با گذاشتن کروشه بعد از نوع داده‌ای پارامتر، یک متد ایجاد کرد که پارامتر آن، آرایه دریافت می‌کند. وقتی متد در خط ۱۹ فراخوانی می‌شود، آرایه را فقط با استفاده از نام آن و بدون استفاده از اندیس ارسال می‌کنیم. پس آرایه‌ها هم به روش ارجاع به

متدها ارسال می‌شوند. در خطوط ۱۲-۹ از حلقه foreach برای دسترسی به اجزای اصلی آرایه که به عنوان آرگومان به متد ارسال کرده‌ایم، استفاده می‌کنیم. 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);
            }
        }
    }
}
```

```
2
3
4
5
6
```

برنامه بالا یک متد را نشان می‌دهد که یک آرایه را دریافت می‌کند و به هر یک از عناصر آن یک واحد اضافه می‌کند. به این نکته توجه کنید که از حلقه foreach نمی‌توان برای افزایش مقادیر آرایه استفاده کنیم، چون این حلقه برای خواندن مقادیر آرایه مناسب است نه اصلاح آنها. در داخل متد، مقادیر هر یک از اجزای آرایه را افزایش داده‌ایم. سپس از متد خارج شده و نتیجه را نشان می‌دهیم. مشاهده می‌کنید که هر یک از مقادیر اصلی متد هم اصلاح شده‌اند. راه دیگر برای ارسال آرایه به متد، مقداردهی مستقیم به متد فراخوانی شده است. به عنوان مثال:

```
IncrementElements(new int[] { 1, 2, 3, 4, 5 });
```

در این روش ما آرایه ای تعریف نمی‌کنیم، بلکه مجموعه‌ای از مقادیر را به پارامتر ارسال می‌کنیم، که آنها را مانند آرایه قبول کند. از آنجاییکه در این روش آرایه ای تعریف نکرده‌ایم، نمی‌توانیم در متد Main() نتیجه را چاپ کنیم. اگر از چندین پارامتر در متد استفاده می‌کنید، همیشه برای هر یک از پارامترهایی که آرایه قبول می‌کنند 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 می‌کند ولی پارامتر دوم (param2) یک عدد صحیح. حال اگر پارامتر دوم (param2) هم آرایه قبول می‌کرد، باید برای آن هم از کروشه استفاده می‌کردیم:

```
void MyMethod(int[] param1, int[] param2)
{
  //code here
}
```

کلمه کلیدی params

کلمه کلیدی params امکان ارسال تعداد دلخواه پارامترهاي همنوع و ذخیره آنها در یک آرایه ساده را فراهم می‌آورد. کد زیر طریقه استفاده از کلمه کلیدی params را نشان می‌دهد:

```
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 داده‌ای آرایه پارامتر استفاده می‌شود (مثال بالا). حال متد را سه بار با تعداد مختلف آرگومانها فراخوانی می‌کنیم. این آرگومانها در داخل یک پارامتر از نوع آرایه ذخیره می‌شوند. با استفاده از حلقه foreach این آرگومانها را جمع و به متد فراخوان برگشت می‌دهیم.

وقتی از چندین پارامتر در یک متد استفاده می‌کنید، 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 می‌شوید. 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
```

محدوده متغیر

متدها در سی‌شارپ دارای محدوده هستند. محدوده یک متغیر به شما می‌گوید که در کجای برنامه می‌توان از متغیر استفاده کرد و یا متغیر قابل دسترسی است. به عنوان مثال متغیری که در داخل یک متد تعریف می‌شود، فقط در داخل بدنه متد قابل دسترسی است. می‌توان دو متغیر با نام یکسان در دو متد مختلف تعریف کرد. برنامه زیر این ادعا را اثبات می‌کند:

```
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);
    }
}
```

```
number inside method DemonstrateScope() = 5
number inside the Main method = 10
```

مشاهده می‌کنید که حتی اگر ما دو متغیر با نام یکسان تعریف کنیم که دارای محدوده‌های متفاوتی هستند، می‌توان 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 می‌توان به آنها آرگومان ارسال کرد یا نه. این پارامترها دارای مقادیر پیشفرضی هستند. اگر به اینگونه پارامترها، آرگومانی ارسال نشود از مقادیر پیش‌فرض استفاده می‌کنند. 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  }
```

```
Welcome to Visual C# Tutorials!
Learn C# Today!
```

متد PrintMessage() (خطوط ۸-۵) یک پارامتر اختیاری دارد. برای تعریف یک پارامتر اختیاری می‌توان به آسانی و با استفاده از علامت = یک مقدار را به یک پارامتر اختصاص داد (خط ۵). دو بار متد را فراخوانی می‌کنیم. در اولین فراخوانی (خط ۱۲) ما آرگومانی به متد ارسال نمی‌کنیم، بنابراین متد از مقدار پیش‌فرض (Welcome to Visual C# Tutorials!) استفاده می‌کند. در دومین فراخوانی (خط ۱۴) یک پیغام (آرگومان) به متد ارسال می‌کنیم، که جایگزین مقدار پیش‌فرض پارامتر می‌شود. اگر از چندین پارامتر در متد استفاده می‌کنید 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 می‌شوند، باید به پارامترهایی که از لحاظ مکانی در آخر بقیه پارامترها نیستند مقدار اختصاص داد. به یاد داشته باشید که نمی‌توان 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 از نام پارامترها استفاده کنید.

```
SomeMethod(10, optional2: 100);
```

برای استفاده از نام پارامتر، شما به راحتی می‌توانید نام مخصوص پارامتر و بعد از نام علامت کالن (:) و بعد مقدار اختصاص شده به آن را بنویسید، 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 می‌کند.

سربارگذاری متدها

سربارگذاری متدها به شما اجازه می‌دهد که چندین متد با نام یکسان تعریف کنید که دارای امضاء و تعداد پارامترهای مختلف هستند. برنامه از روی آرگومانهایی که شما به متد ارسال می‌کنید، به صورت خودکار تشخیص می‌دهد که کدام متد را فراخوانی کرده‌اید 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 نمی‌شوند. در مثال زیر نمونه‌ای از سربارگذاری متدها 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 }
```

```
Double version of the method was called.
Integer version of the method was called.
```

در برنامه بالا دو متد با نام مشابه تعریف شده‌اند. اگر سربارگذاری متد توسط سی‌شارپ پشتیبانی نمی‌شد، برنامه زمان زیادی برای انتخاب یک متد از بین متدهایی که فراخوانی می‌شوند، لازم داشت. رازی در نوع پارامترهای متد نهفته است. کامپایلر بین دو یا چند متد همنام در صورتی فرق می‌گذارد، که پارامترهای متفاوتی داشته باشند. وقتی یک متد را فراخوانی می‌کنیم، متد نوع آرگومانها را تشخیص می‌دهد. در فراخوانی اول (خط ۱۹) ما یک مقدار `double` را به متد `()ShowMessage` ارسال کرده‌ایم، در نتیجه متد `()ShowMessage` (خطوط ۱۰-۷) که دارای پارامتری از نوع `double` است، اجرا می‌شود.

در بار دوم که متد فراخوانی می‌شود (خط ۲۰)، ما یک مقدار `int` را به متد `()ShowMessage` ارسال می‌کنیم. متد `()ShowMessage` (خطوط ۱۵-۱۲) که دارای پارامتری از نوع `int` است، اجرا می‌شود. معنای اصلی سربارگذاری متد همین است که توضیح داده شد. هدف اصلی از سربارگذاري

متدها این است، که بتوان چندین متد که وظیفه یکسانی انجام می‌دهند را تعریف کرد. تعداد زیادی از متدها در کلاس‌های دات‌نت سربارگذاری می‌شوند، مانند متد WriteLine() از کلاس Console. قبلاً مشاهده کردید که این متد می‌تواند یک آرگومان از نوع رشته دریافت کند و آن را نمایش دهد، و در حالت دیگر می‌تواند دو یا چند آرگومان قبول کند.

بازگشت

بازگشت فرایندی است که در آن متد مدام خود را فراخوانی می‌کند تا زمانی که به یک مقدار مورد نظر برسد. بازگشت یک مبحث پیچیده در برنامه‌نویسی است و تسلط به آن کار راحتی نیست. به این نکته هم توجه کنید، که بازگشت باید در یک نقطه متوقف شود، در غیر اینصورت برای بی نهایت بار، متد، خود را فراخوانی می‌کند. در این درس یک مثال ساده از بازگشت را برای شما توضیح می‌دهیم. فاکتوریل یک عدد صحیح مثبت (n!) شامل حاصل ضرب همه اعداد مثبت صحیح کوچک‌تر یا مساوی آن می‌باشد. به فاکتوریل عدد ۵ توجه کنید.

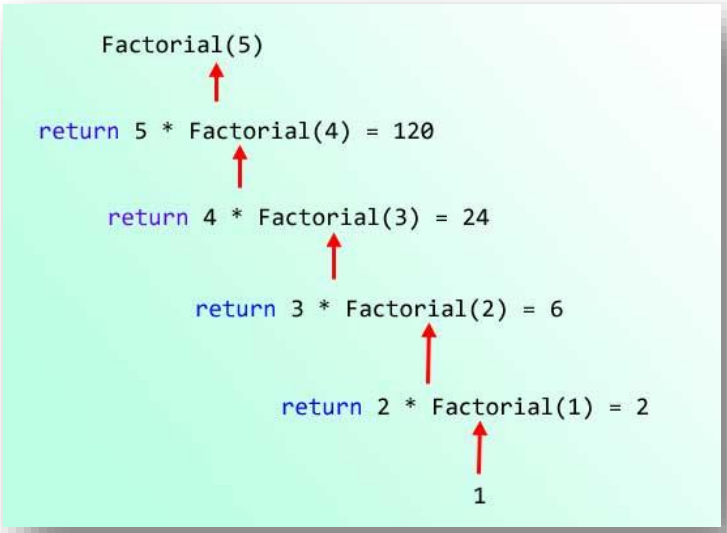
```
5! = 5 * 4 * 3 * 2 * 1 = 120
```

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 ۱ است. در نتیجه از این مقدار برای متوقف کردن بازگشت استفاده می‌کنیم.

```
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  }
```

```
120
```

متد مقدار بزرگی را بر می‌گرداند چون محاسبه فاکتوریل می‌تواند خیلی بزرگ باشد. متد یک آرگومان که یک عدد است و می‌تواند در محاسبه مورد استفاده قرار گیرد را می‌پذیرد. در داخل متد یک دستور if می‌نویسیم و در خط ۷ می‌گوییم که اگر آرگومان ارسال شده برابر ۱ باشد، سپس مقدار ۱ را برگردان در غیر اینصورت به خط بعد برو. این شرط باعث توقف تکرارها نیز می‌شود. در خط ۱۰ مقدار جاری متغیر number در عددی یک واحد کمتر از خودش (number - 1) ضرب می‌شود. در این خط متد Factorial()، خود را فراخوانی می‌کند و آرگومان آن در این خط همان number - 1 است. مثلاً اگر مقدار جاری number، ۱۰ باشد، یعنی اگر ما بخواهیم فاکتوریل عدد ۱۰ را به دست بیاوریم، آرگومان متد Factorial() در اولین ضرب ۹ خواهد بود. فرایند ضرب تا زمانی ادامه می‌یابد که آرگومان ارسال شده با عدد ۱ برابر نشود. شکل زیر فاکتوریل عدد ۵ را نشان می‌دهد.



کد بالا را به وسیله یک حلقه for نیز می‌توان 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 زمینه‌های خاصی در علوم کامپیوتر استفاده می‌شود. استفاده از بازگشت زمانی طبیعی تر به نظر می‌رسد که ما از غیر بازگشتی (Iteration) استفاده کنیم. استفاده از بازگشت حافظه زیادی اشغال می‌کند، پس اگر سرعت برای شما مهم است، از آن استفاده نکنید.

نماینده‌ها (Delegates)

Delegate ها انواعی هستند که مرجع یک متد را در خود ذخیره می‌کنند. همچنین می‌توانند رفتار هر متدی را کپی برداری کنند. برای تعریف یک delegate از کلمه کلیدی delegate استفاده می‌شود. تعریف یک delegate بسیار شبیه به تعریف یک متد است، با این تفاوت که متد بدنه دارد ولی delegate ندارد. Delegate دقیقاً مانند متدها دارای نوع برگشتی و مجموعه‌ای از پارامترها هستند. Delegate ها، می‌گویند که چه نوع متدی را می‌توانند در خود ذخیره کنند. در زیر نحوه تعریف delegate نشان داده شده است:

```
delegate returnType DelegateName(dt param1, dt param2, ... dt paramN);
```

در زیر نحوه استفاده از یک delegate و فواید آن نشان داده شده است:

```
1  using System;
2
3  public class Program
4  {
5      delegate void ArithmeticDelegate(int num1, int num2);
6
```

```
7       static void Add(int x, int y)
8       {
9           Console.WriteLine("Sum is {0}.", x + y);
10      }
11
12      static void Subtract(int x, int y)
13      {
14          Console.WriteLine("Difference is {0}.", x - y);
15      }
16
17      static void Main()
18      {
19          ArithmeticDelegate Operation;
20
21          int num1, num2;
22
23          Console.Write("Enter first number: ");
24          num1 = Convert.ToInt32(Console.ReadLine());
25
26          Console.Write("Enter second number: ");
27          num2 = Convert.ToInt32(Console.ReadLine());
28
29          if (num1 < num2)
30          {
31              Operation = new ArithmeticDelegate(Add);
32          }
33          else
34          {
35              Operation = new ArithmeticDelegate(Subtract);
36          }
37
38          Operation(num1, num2);
39      }
40  }
```

```
Enter first number: 3
Enter second number: 5
Sum is 8
Enter first number: 5
Enter second number: 3
Difference is 2
```

در خط ۵، `delegate` تعریف شده است. از کلمه کلیدی `delegate` برای نشان داده آن استفاده شده است. به دنبال آن نوع برگشتی متدی که قبول می‌کند، هم آمده است. برای نامگذاری `delegate` مانند متدها از روش `Pascal` استفاده می‌کنیم. همچنین برای تشخیص بهتر، بهتر است از کلمه `delegate` در نامگذاری آنها استفاده شود. پارامترهایی که برای `delegate` تعریف می‌کنیم، باید از نظر نوع و تعداد با پارامترهای متدها برابر باشد.

`Delegate` ی که در خط ۵ تعریف شده است، فقط مرجع متدهایی را قبول می‌کند که دارای مقدار برگشتی نیستند و دو پارامتر از نوع `int` دارند. بعد از تعریف `delegate` دو متد با امضای دقیقاً مشابه به عنوان نماینده تعریف می‌کنیم. هر دو متد هیچ مقدار برگشتی ندارند و هر دو، دو آرگومان از نوع `int` قبول می‌کنند. در داخل متد `()Main` یک متغیر از نوع `delegate` ی که قبلاً تعریف کرده‌ایم، تعریف می‌کنیم (خط ۱۹). این متغیر اشاره به متدی دارد که امضای آن با امضای `Delegate` مطابقت دارد. برنامه از کاربر می‌خواهد دو مقدار از نوع `int` را وارد کند. بعد از وارد کردن مقادیر، وارد اولین دستور `if` می‌شویم، چنانچه مقدار اولین عددی که کاربر وارد کرده از دومین عدد وارد شده کمتر باشد، دو عدد با هم جمع

می‌شوند، در غیر اینصورت اگر مقدار اولین عدد بزرگ‌تر یا مساوی دومین عدد باشد، از هم کم می‌شوند. برای ارجاع یک متد به یک delegate به صورت زیر عمل می‌کنیم:

```
variable = new DelegateName(MethodName);
```

وقتی یک delegate را با مرجع یک متد برابر قرار می‌دهیم، باید قبل از نام delegate از کلمه کلیدی new استفاده کنیم (مثال بالا). در داخل پرانتز نام متدی که delegate به آن مراجعه می‌کند، نشان داده شده است. یک راه بسیار ساده تر برابر قرار دادن نام متد با متغیر delegate است:

```
Operation = Add;
Operation = Subtract;
```

به دستور if بر می‌گردیم وقتی شرط درست باشد، delegate را به متد add() و هنگامی که شرط نادرست باشد آن را به متد Subtract() ارجاع می‌دهیم. اجرای delegate باعث اجرای متدی می‌شود که delegate به آن مراجعه می‌کند. اگر قصد داشته باشید که بیش از یک متد را به delegate اضافه کنید باید از عملگر += استفاده نمایید:

```
MyDelegate del = Method1;
del += Method2;
del += Method3;
...
```

کاربرد اصلی delegate ها هنگام کار با رویدادها می‌باشد که در درس‌های آینده توضیح می‌دهیم.

آرگومانهای خط فرمان (Command Line Arguments)

برای اجرای موفق یک برنامه سی‌شارپی باید یک متد مهم به نام متد Main() وجود داشته باشد، که نقطه آغاز برنامه است. این متد باید به صورت public static تعریف شود. همه ما می دانیم که به متدها می‌توان آرگومان ارسال کرد، اما برای متد Main(string[] args) چطور؟ جواب مثبت است. شما می‌توانید از طریق دستور خط فرمان ویندوز یا همان CMD آرگومانهایی را برای این متد ارسال کنید. برای روشن شدن مطلب یک برنامه کنسول به نام Sample ایجاد کنید، سپس کدهای برنامه را به صورت زیر بنویسید:

```
using System;

namespace Sample
{
    class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("First Name is " + args[0]);
            Console.WriteLine("Last  Name is " + args[1]);
            Console.ReadLine();
        }
    }
}
```

برنامه را یک بار اجرا و ذخیره کنید (ممکن است با پیغام خطا مواجه شوید ولی مهم نیست). به پارامتر args توجه کنید. در حقیقت این پارامتر یک آرایه رشته ای است که می‌تواند چندین آرگومان از نوع رشته قبول کند. اگر برنامه‌تان را ایجاد کرده و به فایل با پسوند .exe دسترسی داشته

باشید می‌توانید پارامترهای رشته ای را به متد Main() ارسال کنید. فایلSample.exe را که در پوشه Debug برنامه‌تان است را به یک درایو یا پوشه مشخص که مسیر گیج کننده ای نداشته باشد انتقال دهید. در این مثال ما فایل Sample.exe را مستقیماً در درایو C قرار می‌دهیم. حال CMD ویندوز را اجرا کنید، سپس کدهای زیر را در داخل CMD نوشته و دکمه Enter را بزنید:

```
Microsoft Windows[Version 6.1.7601]
Copyright(c) 2009 Microsoft Corporation.All rights reserved.

C:\Users\VisualCsharp>cd/

C:\>Sample Steven Clark
First Name is Steven
Last  Name is Clark
```

با نوشتن نام فایل، باعث اجرای آن می‌شویم. بعد از نوشتن نام فایل کلمه Steven و سپس Clark را می‌نویسیم. همانطور که در کد مشاهده می‌کنید ما دو متغیر به نام‌های args[0] و args[1] تعریف کرده‌ایم. این دو متغیر به ترتیب خانه‌های اول و دوم آرایه هستند. کلمه Stevenدر متغیر رشته ای args[0] که اولین عنصر آرایه و کلمه Clark را در متغیر رشته ای args[1] که دومین عنصر آرایه است ذخیره و سپس با استفاده از متد WriteLine() آنها را چاپ می‌کنیم. در حقیقت بسیاری از برنامه‌ها از این تکنیک استفاده می‌کنند. شما می‌توانید با ارسال آرگومانهایی به متد Main() نحوه اجرای برنامه را تغییر دهید.

شمارش (Enumeration)

شمارش راهی برای تعریف داده‌هایی است که می‌توانند مقادیر محدودی که شما از قبل تعریف کرده‌اید را بپذیرند. به عنوان مثال شما می‌خواهید یک متغیر تعریف کنید که فقط مقادیر جهت (جغرافیایی) مانند east، west، north و south را در خود ذخیره کند. ابتدا یک enumeration تعریف می‌کنید و برای آن یک اسم انتخاب کرده و بعد از آن تمام مقادیر ممکن که می‌توانند در داخل بدنه آن قرار بگیرند تعریف می‌کنید. به نحوه تعریف یک enumeration توجه کنید:

```
enum enumName
{
   value1,
   value2,
   value3,
   .
   .
   .
   valueN
}
```

ابتدا کلمه کلیدی enum و سپس نام آن را به کار می‌بریم. در سی‌شارپ 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 می‌تواند 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. می‌توانید 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 می‌گیرد.

```
enum Direction
{
    North = 3,
    East = 5,
    South,
    West
}
```

در مثال بالا مشاهده می‌کنید که ما هیچ مقداری برای South در نظر نگرفته‌ایم، بنابراین به صورت خودکار یک واحد بیشتر از East یعنی ۶ و به West یک واحد بیشتر از South یعنی ۷ اختصاص داده می‌شود. همچنین می‌توان مقادیر یکسانی برای عناصر enumeration در نظر گرفت. مثال:

```
enum Direction
{
    North = 3,
    East,
    South = North,
    West
}
```

می‌توانید مقادیر بالا را حدس بزنید؟ مقادیر North، East، South، West به ترتیب ۳، ۴، ۳، ۴ است. وقتی مقدار ۳ را به North می‌دهیم مقدار East برابر ۴ می‌شود. سپس وقتی مقدار South را برابر ۳ قرار دهیم، به صورت اتوماتیک مقدار West برابر ۴ می‌شود. اگر نمی‌خواهید که مقادیر آیتم‌های enumeration شما پیش‌فرض (از نوع int) باشد می‌توانید از نوع مثلاً byte به عنوان نوع داده‌ای آیتم‌های آن اسفاده کنید.

```
enum Direction : byte
{
    North,
    East,
    South,
    West
}
```

نوع داده‌ای byte فقط شامل مقادیر بین ۰ تا ۲۵۵ می‌شود بنابراین تعداد مقادیر که شما می‌توانید به enumeration اضافه کنید، محدود می‌باشد. به نحوه استفاده از enumeration در یک برنامه سی‌شارپ 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  }
```

```
Direction: North
```

ابتدا enumeration را در خطوط ۹-۳ تعریف می‌کنیم. توجه کنید که enumeration را خارج از کلاس قرار داده‌ایم. این کار باعث می‌شود 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 می‌دهیم. در داخل بدنه enumeration نام چهار جهت جغرافیایی وجود دارد که هر یک از آنها با ۱ تا ۴ مقدار دهی شده‌اند. در خط ۱۵ یک متغیر تعریف شده است که مقدار یک جهت را در خود ذخیره می‌کند. نحوه تعریف آن به صورت زیر است:

```
enumType variableName ;
```

در اینجا enumType نوع داده شمارشی (مثلاً Direction یا مسیر) می‌باشد و variableName نیز نامی است که برای آن انتخاب کرده‌ایم که در مثال قبل myDirection است. سپس یک مقدار به متغیر myDirection اختصاص می‌دهیم (خط ۱۷). برای اختصاص یک مقدار به صورت زیر عمل می‌کنیم:

```
variable = enumType.value;
```

ابتدا نوع Enumeration سپس علامت نقطه و بعد مقدار آن (مثلاً North) را می‌نویسیم. می‌توان یک متغیر را فوراً، به روش زیر مقدار دهی کرد:

```
Direction myDirection = Direction.North;
```

حال در خط ۱۹ با استفاده از Console.WriteLine() مقدار myDirection را چاپ می‌کنیم. توجه کنید که با استفاده از متد ToString() مقدار عددی myDirection را به رشته، جهت چاپ تبدیل می‌کنیم. 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 می‌شوند. متغیرهای شمارشی می‌توانند به انواع دیگری مانند int یا string تبدیل شوند. همچنین یک مقدار رشته‌ای می‌تواند به نوع شمارشی معادلش تبدیل شود.

تبدیل انواع شمارشی

می‌توان 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  }
```

```
Value of East is 1

Direction: West
```

در خط ۱۴ مقدار East نوع شمارشی Direction را به متغیر myDirection را به اختصاص داده‌ایم. در حالت پیش‌فرض مقدار East در داخل آیتم‌های این داده شمارشی، ۱ می‌باشد. در خط ۱۵ 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. می‌توان 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 می‌دهد. در خط ۱۹ معکوس این کار را انجام می‌دهیم. در این خط یک مقدار صحیح را به یک مقدار شمارشی تبدیل می‌کنیم. مقدار ۳ را برابر آیتم West قرار می‌دهیم.

برای تبدیل آن از روشی شبیه به تبدیل یک نوع شمارشی به صحیح استفاده می‌کنیم (تبدیل صریح). به این نکته توجه کنید که اگر عددی را که می‌خواهید تبدیل کنید در محدوده انواع شمارشی نباشد، تبدیل انجام می‌شود 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 می‌باشد) خروجی Console خود عدد را نشان می‌دهد. 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 رشته‌ای به یک نوع شمارشی

می‌توان یک نوع رشته‌ای را به نوع شمارشی تبدیل کرد. مثلاً می‌خواهید 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 می‌دهیم. دومین پارامتر، رشته‌ای است که قرار است به نوع شمارشی تبدیل شود. چون مقدار برگشتی از نوع شیء (object) است، بنابراین یک تبدیل مناسب نوع شمارشی لازم است. با این جزییات الان می‌دانیم که چگونه یک رشته را به نوع شمارشی تبدیل کنیم.

```
enumType name = (enumType)Enum.Parse(typeof(enumType), string);
```

اگر رشته‌ای که به متد ارسال می‌کنید جزء آیتم‌های داده شمارشی نباشد، با خطا مواجه می‌شوید.

ساختارها

ساختارها یا struct، انواعی از داده‌ها هستند که، توسط کاربر تعریف می‌شوند (user-define) و می‌توانند دارای فیلد و متد باشند. با ساختارها می‌توان نوع داده‌ای خیلی سفارشی ایجاد کرد. فرض کنید می‌خواهیم داده‌ای 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 می‌کنیم:

```
struct StructName
{
   member1;
   member2;
   member3;
   ...
```

```
    member4;
}
```

برای تعریف ساختار از کلمه کلیدی struct استفاده می‌شود. برای نامگذاری ساختارها از روش نامگذاری Pascal استفاده می‌شود. اعضاء در مثال بالا (member1-4) می‌توانند متغیر باشند یا متد. در زیر مثالی از یک ساختار آمده است:

```
using System;

public struct Employee
{
    public string name;
    public int age;
    public decimal salary;
}

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);

        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);
    }
}
```

```
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 می‌دهیم. در خطوط ۸-۳ یک ساختار تعریف شده است. به کلمه Public در هنگام تعریف توجه کنید. این کلمه کلیدی نشان می‌دهد که Employee می‌تواند در هر جای برنامه قابل دسترسی و استفاده باشد، 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 می‌کنیم. نام

ساختار نیز از روش نامگذاری Pascal پیروی می‌کند. در داخل بدنه ساختار سه فیلد تعریف کرده‌ایم (خطوط ۷-۵). این سه فیلد مشخصات Employee (کارمند) مان را نشان می‌دهند.

مثلاً یک کارمند دارای نام، سن و حقوق ماهانه می‌باشد. همچنین هر سه فیلد به صورت Public تعریف شده‌اند، بنابراین در خارج از ساختار نیز می‌توان آنها را فراخوانی کرد. در خطوط ۱۴ و ۱۵ دو نمونه از ساختار 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 می‌کنید. در خطوط ۱۷ تا ۲۳ به فیلدهای مربوط به هر employee مقادیری اختصاص می‌دهید. 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 می‌نویسیم. وقتی که از عملگر دات استفاده می‌کنیم، این عملگر اجازه دسترسی به اعضای مخصوص آن ساختار یا کلاس را به شما می‌دهد. در خطوط ۲۵ تا ۳۵ نشان داده شده که شما چطور می‌توانید به مقادیر ذخیره شده در هر فیلد دسترسی یابید.

ساختارها انواع مقداری هستند. این بدین معنی است که اگر مثلاً در مثال بالا employee2 را برابر employee1 قرار دهید، employee2 همه مقادیر صفات employee1 را به جای اینکه به آنها مراجعه کند، کپی برداری می‌کند. کلاس یک ساختار ساده است ولی از انواع مرجع به حساب می‌آید. در مورد کلاس در درس‌های آینده توضیح خواهیم داد. می‌توان به ساختار، متد هم اضافه کرد. مثال زیر اصلاح شده مثال قبل است.

```
1   using System;
2
3   public struct Employee
4   {
5       public string name;
6       public int age;
7       public decimal salary;
8
9       public void SayThanks()
10      {
11          Console.WriteLine("{0} thanked you!", name);
12      }
13  }
14
15  public class Program
16  {
17      public static void Main()
18      {
19          Employee employee1;
20          Employee employee2;
21
22          employee1.name = "Jack";
23          employee1.age = 21;
24          employee1.salary = 1000;
25
26          employee2.name = "Mark";
27          employee2.age = 23;
28          employee2.salary = 800;
29
30          Console.WriteLine("Employee 1 Details");
31          Console.WriteLine("Name: {0}", employee1.name);
32          Console.WriteLine("Age: {0}", employee1.age);
33          Console.WriteLine("Salary: {0:C}", employee1.salary);
34
35          employee1.SayThanks();
36
37          Console.WriteLine(); //Seperator
38
```

```
39          Console.WriteLine("Employee 2 Details");
40          Console.WriteLine("Name: {0}", employee2.name);
41          Console.WriteLine("Age: {0}", employee2.age);
42          Console.WriteLine("Salary: {0:C}", employee2.salary);
43
44          employee2.SayThanks();
45      }
46  }
```

```
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 می‌دهد و مقدار فیلد name را گرفته و یک پیام منحصر به فرد برای هر نمونه نشان می‌دهد. 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 می‌کنید (خطوط ۳۵ و ۴۴) پرانتزها را قرار می‌دهیم و در صورتی که متد به آرگومان هم نیاز داشت در داخل پرانتز آنها را می‌نویسیم.