دانشگاه فنی و حرفه ای امام خمینی (ره) قاین

جزوه درس : برنامه سازی پیشرفته

هدف یادگیری زبان برنامه نویسی #c

مدرس: حمیدرضا رضاپور

گام اول آشنایی با زبان#C

آشنایی با زبان#C

زبان #C یک زبان سطح بالا، شی گرا و همه منظوره است که به وسیلهٔ شرکت مایکروسافت هم زمان با پیدایش لایه نرم افزاری جدید آن به نام .NET ابداع و توسعه پیدا کرده است. از نرم افزارهای متنوع و گوناگونی از جمله نرم افزاهای اداری و برنامه های کاربردی تحت وب گرفته تا نرم افزارهایی برای تلفن همراه و بازی های کامپیوتری، با زبان #C و با استفاده از لایه .NET تولید می شود.

زبان #C شباهت زیادی به زبان های C++ و Java دارد و ویژگی هایی را از آنها تقلید کرده، یا بعضی امکانات آنها را بهبود داده است. تلاش شده است که بهترین ویژگی ها گردآوری شود، اما بر خلاف زبان جاوا که متن باز است، #Cدر انحصار و اختیار سازنده آن یعنی شرکت مایکروسافت است. زبان های C++ و Java هر دو به زبان C بر می گردند که در سال ۱۹۷۰ ابداع شد و معروفیت آن به دلیل نوشتن سیستم عامل UNIX به وسیله آن بود. زبانC ، یک زبان حرفه ای است که دست برنامه نویس را برای نوشتن برنامه و دسترسی به سخت افزار، باز می گذارد و دارای انعطاف بسیار زیادی است، به همین دلیل کمتر اشتباهات منطقی برنامه نویس را کنترل می نماید. اما در زبان#C ، در هنگام ترجمه و همچنین اجرای برنامه دقت زیادی بر روی تطبیق و به کارگیری داده ها صورت می گیرد تا از اشتباهات دستوری برنامه نویس یا کاربر جلوگیری نماید.

شروع برنامه نویسی

همان طور که برای تهیه و پخت غذا، به مواد اولیه، لوازم آشپزی و دستور پخت نیاز داریم برای تولید یک برنامه نیز، به یک کامپیوتر یا لپ تاپ، لوازم برنامه نویسی(یک ویرایشگر متنی و یک برنامه مترجم) و همچنین به یک الگوریتم نیاز داریم. اگر یک کامپیوتر با سیستم عامل ویندوز ۱۰ در دسترس باشد تقریباً تمام مواد اولیه و لوازم مورد نیاز را در اختیار داریم.

در این صورت در طی مراحل زیر می توانیم برنامه ای را نوشته، ترجمه کرده و سپس اجرا کنیم.

۱-نوشتن برنامه و ذخیره آن با استفاده از یک ویرایشگر مانند Notepad ویندوز

۲-ترجمه برنامه ذخیره شده به وسیله مترجم زبان C# به نامCSC.EXE

این مترجم با نصب .NET Framework در روی کامپیوتر قرار می گیرد.

۳-اجرای برنامه ترجمه شده

شکل زیر این مراحل را از چپ به راست نشان می دهد:

در انتهای این بخش، در قسمت کار در کارگاه، تمام مراحل بالا را به صورت عملی تجربه خوهید کرد.

اولین برنامه به زبان#C

با یک برنامه ساده به زبان #C آشنا می شویم.

```
class WelcomeToCSharp
{
static void Main( )
{
System.Console.WriteLine(''Welcome To C#!'');
}
}
```

این برنامه کوچک فقط یک پیام خوش آمد گویی بر روی صفحه نمایش نشان می دهد. رنگ های کلمات که در این برنامه مشاهده می کنید، تنها برای کمک به واضح شدن برنامه برای خواننده به کار گرفته شده است و تأثیری بر روی برنامه ندارد. همان طور که در برنامه Notepad آنچه که می نویسد همگی با یک رنگ نوشته می شود.

سؤال: آیا با مشاهده این برنامه و بدون اطلاعات قبلی و یا کمک از دیگران، می توانید حدس بزنید که چه پیامی بر روی صفحه نشان داده می شود؟

برای این که با این برنامه آشنا شویم و یاد بگیریم که چگونه باید به زبان C# برنامه بنویسیم از دو جنبه این برنامه را بررسی خواهیم کرد:

الف) نگاه جزیی تر در حد کلمات و علامت ها

ب) نگاه کلی تر در حد تقسیم بندی یک برنامه به قسمت های مختلف

با نگاهی جزیی تر به برنامه بالا، مشاهده می کنید که این برنامه از تعدادی کلمه و علامت تشکیل شده است. بعضی از کلمات مانندStatic ، Classو Void کلمات شناخته شده برای زبان C# هستند و دارای معنی و مفهوم ثابتی هستند به این نوع کلمات، کلمات کلیدی یا رزرو شده گفته می شود. کلمات رزرو شده به رنگ آبی در این جزوه نوشته شده اند و به تدریج با آنها آشنا می شوید.

بعضی از کلمات دیگر مانند WelcomeToCSharp نامی است که به وسیله برنامه نویس و طبق سلیقه وی انتخاب می شود . به این نام ها شناسه می گویند. برنامه نویس در انتخاب شناسه ها باید ضوابطی را رعایت کند که در گام های بعدی با آن آشنا می شوید.

علامت هایی مانند }، {،)، (و " نیز در این برنامه دیده می شود که معمولاً برای شروع یا پایان یک قسمت استفاده می گردد.

با نگاهی دیگر و کلی تر به برنامه بالا، مشاهده می کنیم که یک برنامه ساده از یک قسمت کلی به نام کلاس تشکیل شده است که با کلمه کلیدی class مشخص می شود و شروع و پایان آن با علامت آکولاد باز و بسته تعیین می گردد. در جلوی کلمه کلیدیClass ، یک نام(شناسه) دلخواه مثلاً WelcomeToCSharp نوشته می شود که بیان کننده کار برنامه است. قسمت کلاس برنامه را در شکل زیر مشاهده کنید.

```
class WelcomeToCSharp
{
}
```

اگر درون کلاس WelcomeToCSharp را نگاه کنیم یک قسمت دیگر را خواهیم دید که چنین شروع شده است:

```
static void Main( )
```

شروع و پایان این قسمت نیز با علامت های آکولاد باز و بسته، مشخص شده است. به این قسمت متد Main می گوییم که بدنه اجرایی برنامه است هر دستوری که در این قسمت نوشته شود به وسیلهٔ کامپیوتر به ترتیب اجرا می شود . دستورهای برنامه خود را در این قسمت می نویسیم.

```
static void Main( )
{
System.Console.WriteLine(''Welcome To C#!'');
}
```

آخرین قسمتی که در برنامه بالا، در داخل متد Main قابل تشخیص است، یک دستور اجرایی است و به کامپیوتر اعلام می کند که چه باید انجام دهد که در این برنامه ، نمایش یک پیام است:

```
System.Console.WriteLine(''Welcome To C#!'');
```

با اجرای دستور بالا، پیام خوش آمدگویی Welcome To C#! بر روی صفحه نمایش، نشان داده می شود.

آیا شما قبلاً جزوهت حدس زده بودید که برنامه بالا، چه پیامی را بر روی صفحه نمایش، نشان می دهد؟!

به وسیله دستور بالا، هر آنچه که داخل علامت های نقل قول'' '' قرار داشته باشد، بر روی صفحه نمایش نشان داده می شود حتی اگر به زبانی غیر از انگلیسی مثلاً فارسی نوشته شده باشد.

توجه داشته باشید که خود علامت های نقل قول بر روی صفحه نمایش، نشان داده نمی شوند. بلکه این علامت ها برای مشخص کردن شروع و پایان عبارتی است که می خواهیم روی صفحه نشان داده شود.

%توجه داشته باشید که زبان C# مانند زبان هایC ، C++و Java نسبت به حروف کوچک و بزرگ حساس است و چنانچه قصد دارید برنامه ای را در کامپیوتر وارد کنید به دیکته و نوع حروف کوچک و بزرگ کلمات توجه داشته باشید. مثلاً کلمات Static و Void باید با حروف کوچک نوشته شود ولی حرف اول کلمه Mainباید حرف بزرگ (M) باشد.

الگوی یک برنامه ساده به زبان#C

یک برنامه کاربردی نوشته شده به زبان#C ، شامل مجموعه ای از کلاس ها است که هر یک از آنها نیز شامل تعدادی متد هستند. اما در یک برنامه ساده مانند برنامه بالا، تنها یک کلاس وجود دارد که در آن نیز فقط یک متد به نام Mian() تعریف می شود که نقطه آغاز اجرای برنامه است و الگوریتم خود را با رعایت قوانین زبان C# در آن می نویسیم. الگو یا ساختار کلی یک برنامه ساده به زبان C# در زیر آمده است. الگوی زیر را به خاطر بسپارید.

classیک نام دلخواه

{

static void Main()

{

دستورات مربوط به انجام یک کار

}

}

کلاس چیست؟

کلاس یک مفهوم اساسی در برنامه نویسی شی گرا است که در قسمت های بعدی جزوه به تفصیل بحث می شود. در اینجا اگر بخواهیم به طور ساده در مورد معنی و مفهوم کلاس صحبت کنیم، بایدبگوییم که کلاس به عنوان یک قالب یا الگویی می باشد که در آن داده هایی تعریف می شود. این داده ها مربوط به یک موضوع

است و عملیاتی که می توان بر روی آنها انجام داد. در زبان C# گنجینه ای از کلاس های مختلف و کاربردی، از قبل تعریف شده و آماده وجود دارد که برنامه نویس کافی است آنها را بشناسد و در برنامه استفاده نماید . Consoleیک کلاس آماده در زبان C# است که عملیات مختلف و ورودی و یا خروجی (بر روی صفحه نمایش و یا صفحه کلید) در آن تعریف شده است.

نحوه تعریف کلاس: در زبان C# این امکان برای برنامه نویس فراهم است که کلاس جدیدی را تعریف کند. همان طور که در زیر مشاهده می کنید از کلمه کلیدی Class برای تعریف و مشخص کردن یک کلاس جدید استفاده می شود. در جلوی کلمهClass ، یک نام دلخواه ذکر می گردد که نام کلاس است. مانند WelcomeToCSharp

class نام کلاس

{

تعریف داده ها

و عملیات بر روی آنها

}

%نام گذاری کلاس: نام یک کلاس به وسیلهٔ برنامه نویس نام گذاری می شود. سعی کنید یک نام با معنی و مطابق با کار برنامه انتخاب کنید. ممکن است این نام از چند کلمه تشکیل شده باشد. بین کلمات نباید فاصله بگذارید ولی برای این که خواندنِ نام به راحتی انجام شود و تشخیص کلمات آسان باشد، از روش پاسکال استفاده کنید که در آن، اولین حرف هر کلمه با حرف بزرگ نوشته می شود.

متد چیست؟

همان طور که گفته شد در داخل کلاس، عملیات بر روی داده ها و یا الگوریتم انجام یک کار تعریف می شود. متد مجموعه ای از دستورات است که برای انجام یک کار لازم است. هر متد مطابق با عملکردش نام گذاری می شود و همچنین دارای یک جفت پرانتز باز و بسته است که در آن ممکن است ورودی هایی ذکر شود که برای انجام کار لازم است.

در برنامه های زبان C# ، ممکن است متدهای زیادی تعریف و یا مورد استفاده قرار گیرند، اما حتماً باید متدی به نام Main() تعریف شده باشد که نقطه آغاز اجرای برنامه است و اجرای یک برنامه از اولین دستور داخل آن شروع می شود.

کلمات Static و Void در قالب کلی متد Main() ویژگی های متد را مشخص می کنند که در قسمت های بعدی توضیح و با آن آشنا می شوید.

```
static void Main()
{
دستور شماره ۱;
دستور شماره ۲;
دستور شماره ۳;
ادامه دستورات;
}
```

در این جزوه مانند برنامه بالا، فقط به تعریف متد Main() می پردازیم و از متدهای آماده در زبان C# استفاده خواهیم کرد.

استفاده از متدهای آماده: تعداد زیادی متد در کلاس های آماده زبان C# وجود دارد که هر یک از آنها، برای انجام کاری در نظر گرفته شده است. مثلاً متد WriteLine() از کلاس Console برای نشان دادن پیام روی صفحه نمایش در نظر گرفته شده است که در برنامه بالا از آن استفاده کردیم:

```
System.Console.WriteLine("Welcome To C#!");
```

همان طور که برای آجزوه دادن منزل خود به دیگران، نام منطقه، خیابان، کوچه و شماره پلاک را ذکر می کنید، برای استفاده از یک متد نیز باید نام فضا یا حوزه، نام کلاس و سپس نام متد را

مشخص کنید و برای جدا کردن آنها از یکدیگر، علامت نقطه بین آنها قرار دهید.

به این ترتیب برای استفاده از متد WriteLine() در برنامه بالا، مشاهده می کنید که فضای نام System و نام کلاس Console و در آخر نام متد نوشته شده است که با علامت نقطه از یکدیگر جدا شده اند. متدهای دیگری نیز در کلاس Console وجود دارد که در این گام با برخی از آنها آشنا می شوید.

برنامه زیر برای نمایش دو پیام بر روی صفحه نوشته شده است:

```
class WelcomeToCSharp
{
static void Main()
{
System.Console.WriteLine("Welcome To C#!");
```

```
// Insert a blank line

System.Console.WriteLine();

System.Console.WriteLine(''This is my first program.'');

}

}
```

برنامه بالا مانند برنامه اول است با این تفاوت که سه خط دیگر به متد Main() اضافه شده است. خط دوم این برنامه فقط یک توضیح برای خواننده برنامه می باشد و توضیح می دهد که خط بعدیِ برنامه چه عملی را انجام می دهد. نشانه توضیحات علامت//)دو بار کلید/ (است و مترجم با دیدن این علامت متوجه می شود که این خط یک توضیح است؛ بنابراین آن را به زبان ماشین ترجمه نمی کند.

خط سوم یک دستور اجرایی است:

```
System.Console.WriteLine();
```

در این خط از متد WriteLine() استفاده شده است با این تفاوت که داخل پرانتز، خالی است. اجرای این دستور سبب می شود که روی صفحه نمایش یک سطر خالی ایجاد شود. دستور آخر، پیام This is my first program.را روی صفحه نمایش نشان می دهد.

با توجه به برنامه دوم، پیامی را که می خواهید نمایش داده شود باید بین علامت های نقل قول " " قرار دهید. برای مثال اگر اسم شما AmirMohammad است و بخواهید روی صفحه نشان داده شود، باید به صورت زیر بنویسید:

```
System.Console.WriteLine(AmirMohammad'');

System.Console.WriteLine(HendiVarkaneh'');
```

حروف، علامت ها و عبارتی که ما بین علامت های نقل قول نوشته می شود را رشته می نامند. این حروف می تواند فارسی، انگلیسی یا به هر زبانی باشد. نام "AmirMohammad" و یا یک رمز عبور

.”Hosna۲۰۲۰“هر یک از حروف و علامت ها را نیز یک کاراکتر می نامند. برای مثال AmirMohammadاز ۱۲ کارکتر و رمز عبور Hosna۲۰۲۰ از ۹ کاراکتر تشکیل شده است. چنانچه فاصله در رشته وجود داشته باشد، فاصله نیز یک کاراکتر محسوب می شود.

%برای درج توضیحات در برنامه، اگر یک خط باشد از علامت // و چنانچه چند خط باشد از علامت */توضیحات/* استفاده می شود.

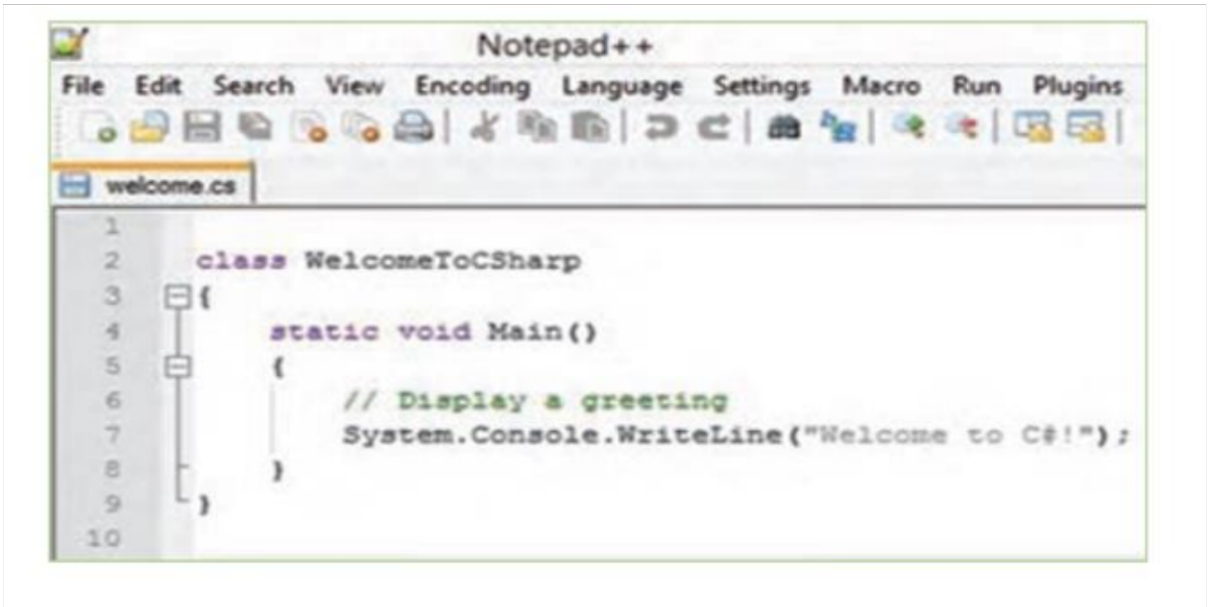
```
// Display a greeting message

/*

FileName: welcome.cs ... Date : ۰۵-۰۷-۲۰۲۰

Display a greeting message

*/
```

کار در کارگاه: برنامه نویسی به زبان#C

قدم اول: نوشتن و تایپ برنامه: برای نوشتن یک برنامه ساده مانند برنامه اول، که در این گام مورد بررسی قرار گرفت، نیاز به یک ویرایشگر متنی است. یک ویرایشگر متنی قادر است حروف، کلمات و آنچه را که تایپ می کنید بدون در نظر گرفتن اطلاعات نوع فونت، اندازه حروف و رنگ در یک فایل ذخیره کند. برنامه Notepad یک ویرایشگر ساده است که همراه با سیستم عامل ویندوز در روی کامپیوتر نصب می شود.

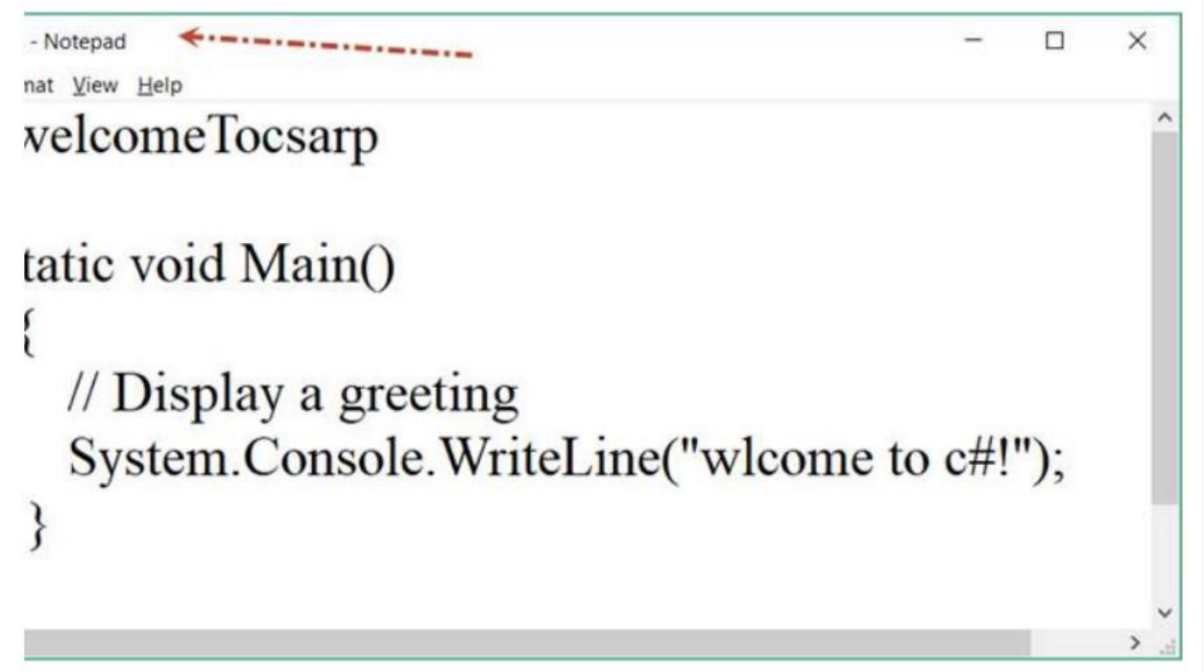
علاوه بر برنامهNotepad ، می توانیم از ویرایشگر دیگری مانند Notepad++ که به صورت رایگان از طریق سایت آن قابل دانلود است استفاده نماییم. این ویرایشگر به منظور برنامه نویسی به زبان های مختلف طراحی شده است به طوری که کلمات رزرو شده، رشته ها و توضیحات در این ویرایشگر با رنگ های مختلف نشان

داده می شود. البته برای استفاده از این ویژگی باید ابتدا از منوی Language ، زبان برنامه نویسی مورد نظر خود را C# انتخاب کنید.



توجه داشته باشید که از برنامه Word استفاده نکنید که در این صورت، کدهای اضافی مربوط به صفحه بندی، رنگ و فونت را نیز به فایل شما اضافه می کند که مترجم در هنگام ترجمه برنامه، انتظار آنها را ندارد و دچار مشکل می شود.

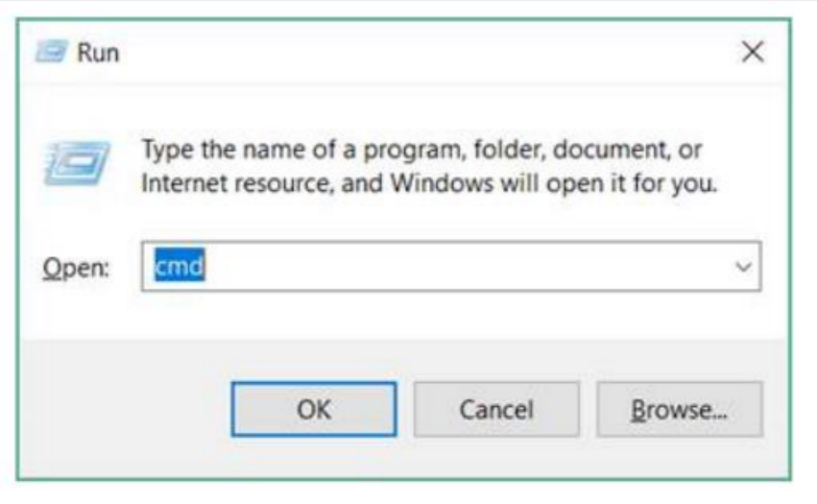
در هنگام تایپ دستورات، صرفنظر از اینکه از چه ویرایشگری (Notepad , Notepad++)استفاده می کنید، باید دقت داشته باشید که زبان C# نسبت به حروف کوچک و بزرگ حساس است؛ بنابراین برنامه را دقیقاً مانند جزوه تایپ کنید.



پس از آنکه برنامه اول را تایپ کردیم، باید آن را ذخیره کنیم. برای این منظور به منوی File بروید و گزینه Save As...را انتخاب کنید. در هنگام ذخیره فایل دقت کنید که فایل را در کجا و چه مسیری ذخیره می نمایید و نام فایل مناسبی را برای آن انتخاب کنید و پسوند آن را cs. قرار دهید (شاید بهتر باشد یک پوشه در یکی از درایوها به نام خودتان بسازید و فایل را در آن ذخیره کنید. در این تمرین نام فایل را Welcome.csقرار دهید.

قدم دوم: ترجمه و اجرای برنامه: پس از نوشتن و ذخیره کردن برنامه، لازم است ابتدا برنامه را ترجمه کنیم و اگر اشکالی در تایپ آن وجود دارد آن را برطرف کنیم. برای ترجمه برنامه مراحل زیر را دنبال کنید.

-۱از طریق گزینه Run فرمان cmd را اجرا کنید، تا وارد پنجره فرمان شویم.



-۲پنجره فرمان ظاهر می شود (نوشته ها در شکل، ممکن است با آنچه در پنجره Command Prompt کامپیوتر شما دیده می شود متفاوت باشد.

-۳در پنجره Command Prompt از فرمان Dir استفاده می کنیم و با توجه به این که پسوند فایل cs می باشد، با تایپ فرمان زیر از وجود فایل برنامه مطمئن می شویم. اگر فایل را پیدا نکردید باید وارد پوشه ای شوید که برنامه را در آنجا ذخیره کرده اید. به این ترتیب از دستور Cd برای وارد شدن به پوشه مورد نظر استفاده کنید. شاید دستور Cd.. نیز برای وقتی که می خواهید به یک پوشه بالاتر بروید مفید باشد.

-۴من برای اینکه فایل مورد نظر را به راحتی پیدا کنم در درایو C یک پوشه به نام ali ایجاد، و فایل مورد نظر که قبلاً ایجاد کرده بودم درون این پوشه قرار دادم بعد در پنجره Command Prompt به جستجوی فایل پرداختم. چون مرحله قبل را به جزوهتی انجام دادیم، شکل زیر مشاهده می شود:

-۵توجه داشته باشید اگر برنامه Net Framework. بر روی کامپیوتر شما قبلاً نصب شده باشد برنامه ای به نام csc.exe برای ترجمه برنامه های C# در اختیار دارید. پس از یافتن فایل خود، با استفاده از این مترجم، برنامه خود را ترجمه نمایید. در پنجره فرمان از دستور زیر استفاده کنید:

منظور از filename.cs نام فایل مورد نظر شما است.

csc filename.cs

گام دوم

آشنایی با ویژوال استودیو

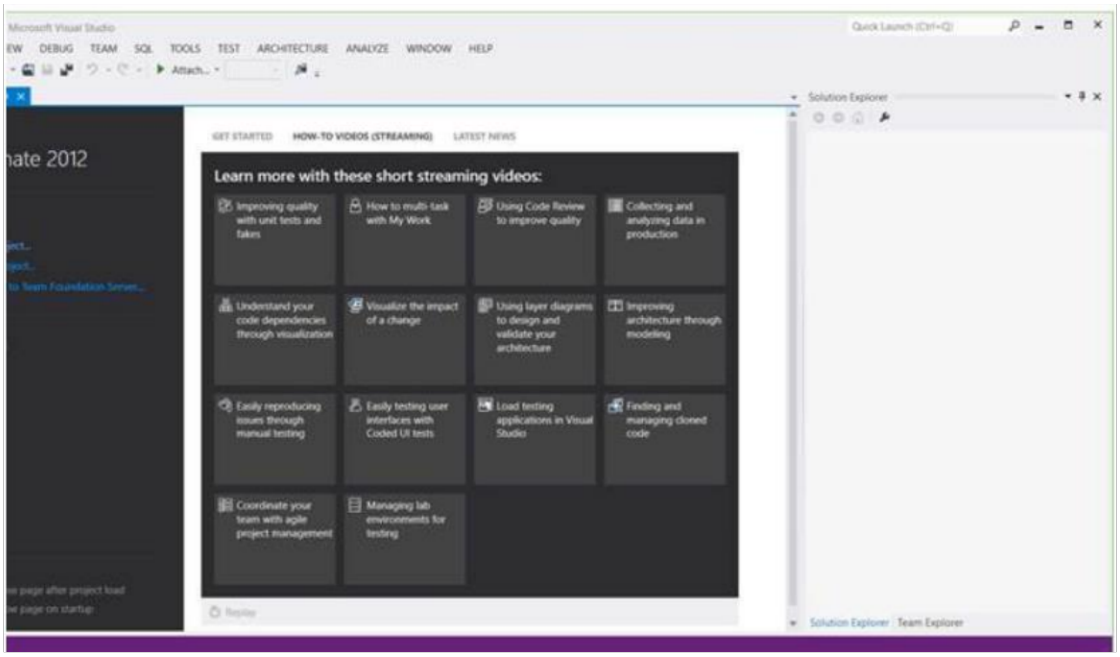
آشنایی با ویژوال استودیو

تایپ برنامه در یک ویرایشگر، ورود به پنجره فرمان، ترجمه کردن، عیب یابی و اشکال زدایی برنامه، همگی عملیاتی بودند که در گام قبلی انجام دادید وکمی وقت گیر و پر زحمت بود، چون ازیک محیط باید وارد محیط دیگری می شدید. برای این که راحت تر بتوانیم برنامه نویسی کنیم لازم است از محیطی استفاده کنیم که همه ابزارها و لوازم مورد نیاز برنامه نویسی در آن گردآوری و متمرکز شده باشد. به چنین محیط برنامه نویسی که در آن می توان تمام مراحل برنامه نویسی، ترجمه، اشکال یابی و سرانجام اجرا را انجام داد، IDE گفته می شود که به معنای محیط تولید برنامه متمرکز می باشد. یعنی همه ابزارها و امکانات لازم برای تولید برنامه در یک جا گردآوری شده است.

شرکت مایکروسافت یک IDE بسیار پیشرفته برای برنامه نویسی فراهم کرده است که با کمک آن می توانیم راحت تر برنامه بنویسیم و ترجمه و اجرا کنیم. نام این نرم افزار ویژوال استودیو است.

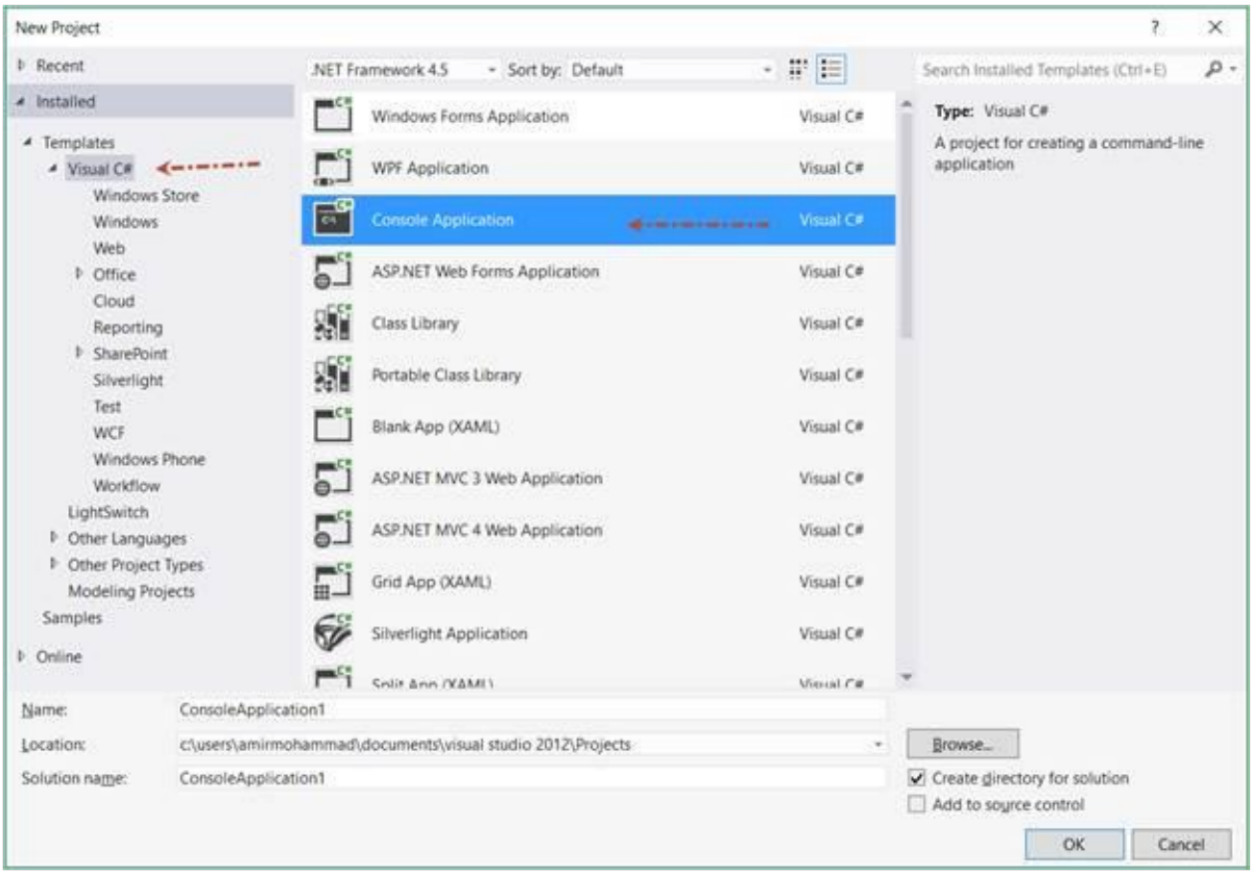
ویژوال استودیو یک محیط برنامه نویسی بسیار قوی برای تولید برنامه های کاربردی تحت ویندوز و بر پایه Net Framework.می باشد. ویژوال استودیو از چند زبان برنامه نویسی نظیر C++ ، C#و VB پشتیبانی می کند. در این محیط علاوه بر تایپ برنامه، می توان برنامه را ترجمه، عیب یابی و سرانجام اجرا کرد. لایه نرم افزاری Net Framework ٤,٥. به همراه ویژوال استودیو ۲۰۱۲ عرضه شده است. خوشبختانه شرکت مایکروسافت همراه با عرضه ویژوال استودیوی تجاری، یک نسخه رایگان از این نرم افزار را تحت عنوان ویژوال استودیو اکسپر نیز عرضه می کند که می توانید آن را از روی سایت شرکت مایکروسافت دانلود نمایید.

ما در این جزوه با Visual Studio Express ۲۰۱۲ کار می کنیم و از این به بعد در سرتاسر جزوه از مخففVS برای بیان کلمه ویژوال استودیو استفاده می کنیم. اگر نرم افزار VS ۲۰۱۲را در اختیار ندارید می توانید از نسخه های قدیمی تر نیز استفاده کنید فقط شکل ظاهری آن ممکن است کمی متفاوت باشد.



ایجاد یک پروژه جدید در ویژوال استودیو

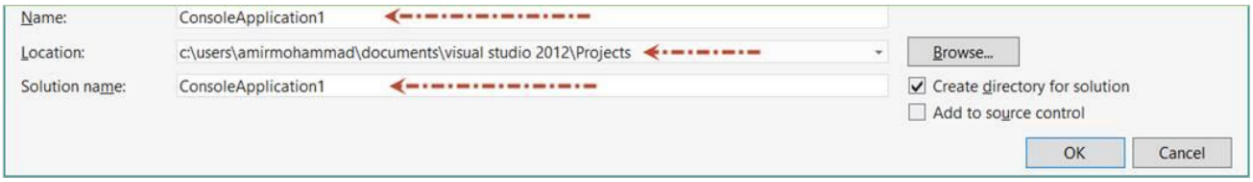
با اجرای برنامه VS ، صفحهٔ شروع (Start Page) مطابق با شکل زیر ظاهر می شود، از این صفحه می توانید یک پروژه یا برنامه جدید (New Project...) بسازید و یا برنامه های قبلی خود را باز (Open Project...) کنید.



برای ایجاد یک برنامهٔ جدید به زبان C# که در محیط کنسول کار می کند ابتدا روی گزینه (New Project...) ، در لیست سمت چپ کلیک کنید. پس از ظاهر شدن پنجره Visual C# را انتخاب کنید و در وسط صفحه بر روی گزینهٔ Console Application کلیک کنید.

در قسمت Name باید نام پروژه را تایپ کنید که به صورت پیش فرض ConsoleAppliaction۱ برای آن در نظر گرفته شده است. مناسب است نامی مطابق با هدف برنامه ای که می نویسید انتخاب کنید چون این نام به هیچ وجه گویا و روشن نیست.

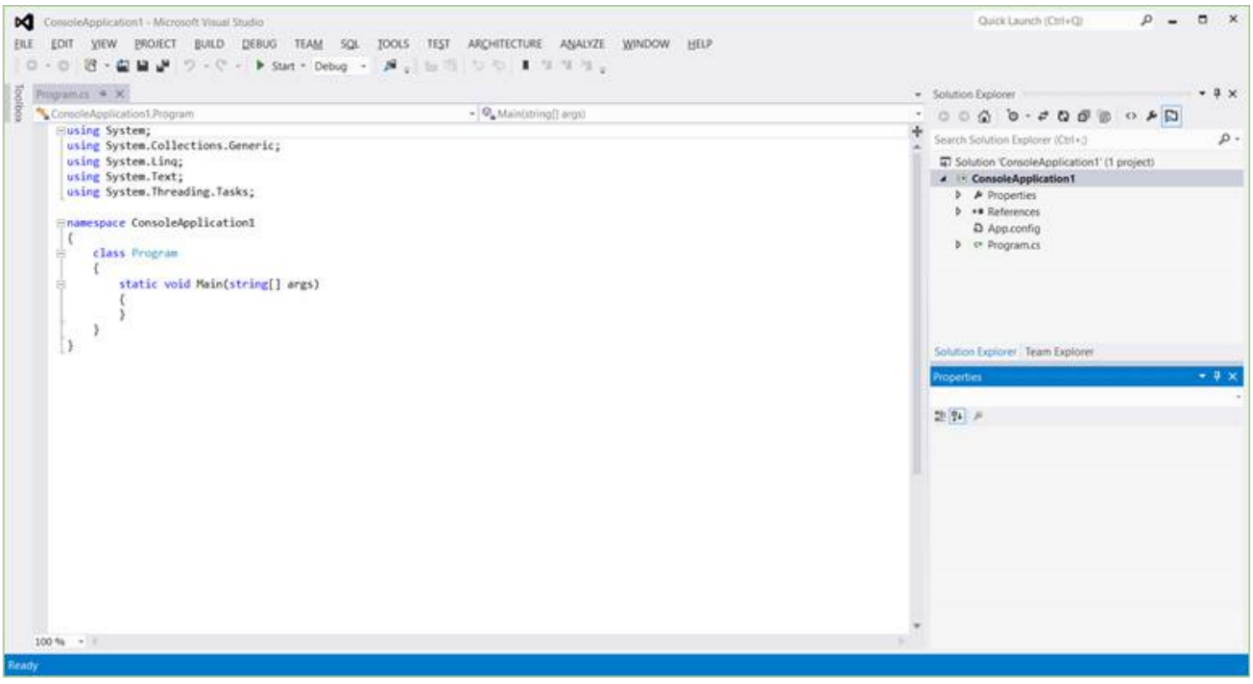
همان طور که در شکل زیر مشاهده می کنید، در قسمت Location مسیر ذخیره پروژه نشان داده شده است.هنگامی که VS را نصب می کنید در داخل My Documents یک پوشه به نام Visual Studio ساخته می شود که در داخل آن نیز یک پوشه فرعی به نام Projects ایجاد می شود. در داخل این پوشه، پروژه های شما به طور پیش فرض ذخیره می شود. اگر بخواهید می توانید پروژه خود را در مسیر دیگری ذخیره کنید. مسیر دلخواه خود را با کلیک بر روی دکمهٔ Browse مشخص کنید.



بعد از مشخص کردن محل پروژه، باید نامی برای Solution name در نظر بگیرید که نام پوشه ای را معین می کند که فایل های مربوط به یک یا چند پروژه در آن نگهداری می شود. معمولاً نام Solution با نام پروژه یکسان انتخاب می شود.

در حال حاضر چون هدف ما آشنایی با VS است نام Introduction را در قسمت Name می نویسیم و این نام برای Solution نیز انتخاب می شود. حال پس از تعیین نام ها، بر روی کلید Ok کلیک کنید. در این صورت یک پوشه با نام مذکور در مسیر پیش فرض ساخته می شود. درون این پوشه چندین فایل و یک پوشه فرعی با نامی که برای پروژه انتخاب کردید (در مثال ما Introduction) ساخته می شود.

پس از چند لحظه مشاهده خواهید کرد که در IDE پنجره های مختلفی نشان داده می شود و در یکی از پنجره ها صورت کلی یک برنامه C# به صورت آماده مانند شکل زیر نشان داده می شود.

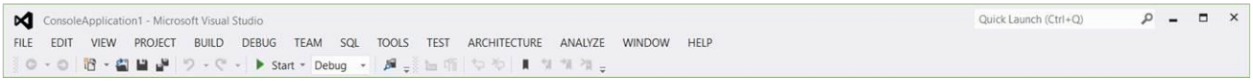


قبل از این که به برنامه نویسی بپردازیم ابتدا به صورت مختصر با بخش های مختلف محیط IDE آشنا می شویم.

معرفی بخش های اصلی ویژوال استودیو

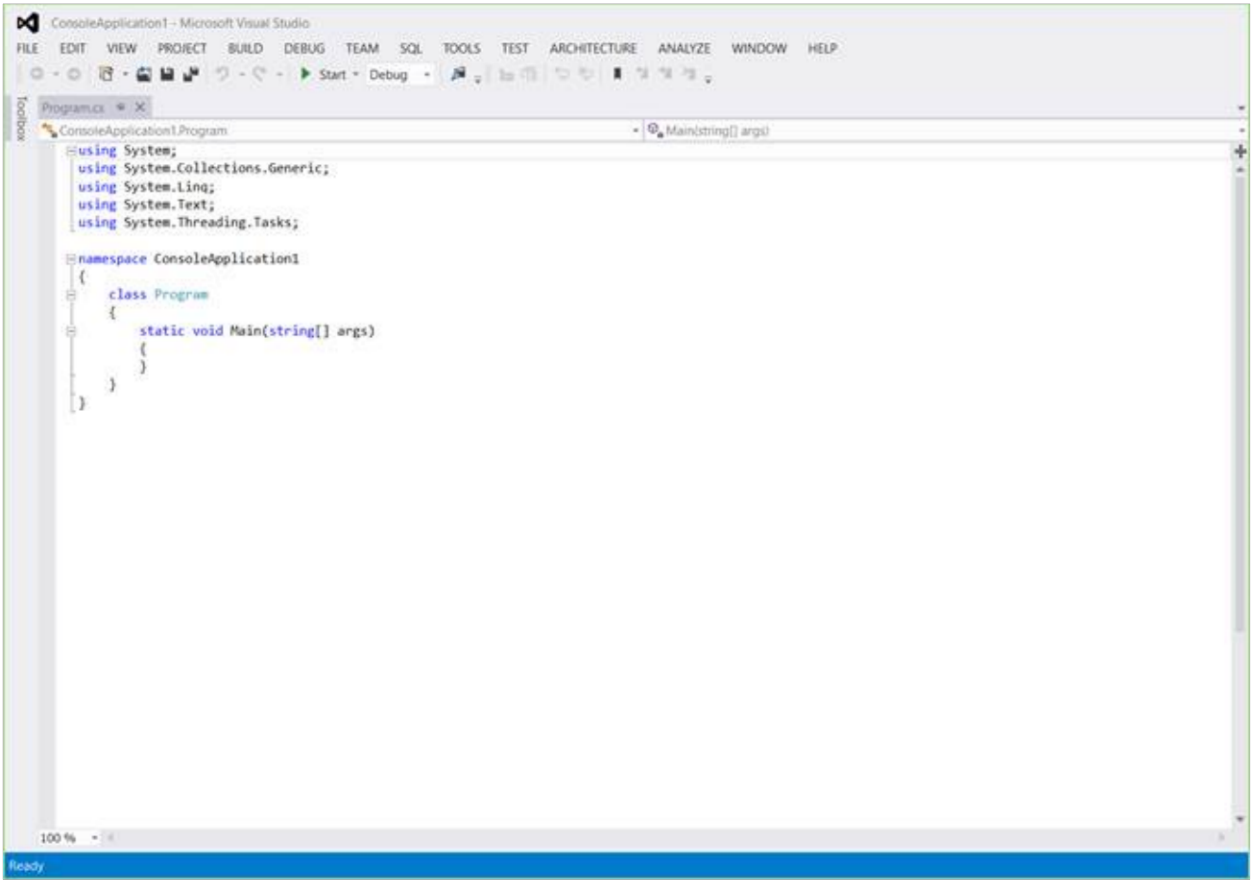
نوار منو و نوار ابزار

مانند بیشتر نرم افزارها، در قسمت بالای صفحه، منوهای مختلف VS و در زیر آن ابزارهای پرکاربرد مطابق با شکل زیر دیده می شود. به تدریج با این منوها وابزارها آشنا می شوید.



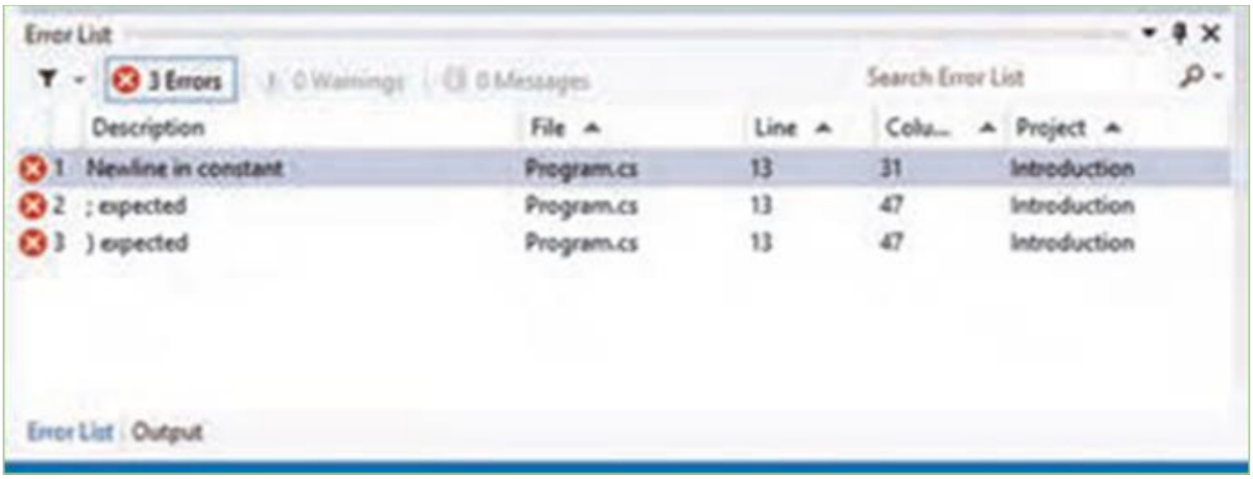
پنجره ویرایشگر برنامه

در شکل زیر پنجره ویرایشگر برنامه نشان داده شده است. در این پنجره متن برنامه نگهداری می شود و می توانید چندین برنامه را هم زمان به صورت باز داشته باشید. در برنامه نویسی، این پنجره کاربرد زیاد دارد.



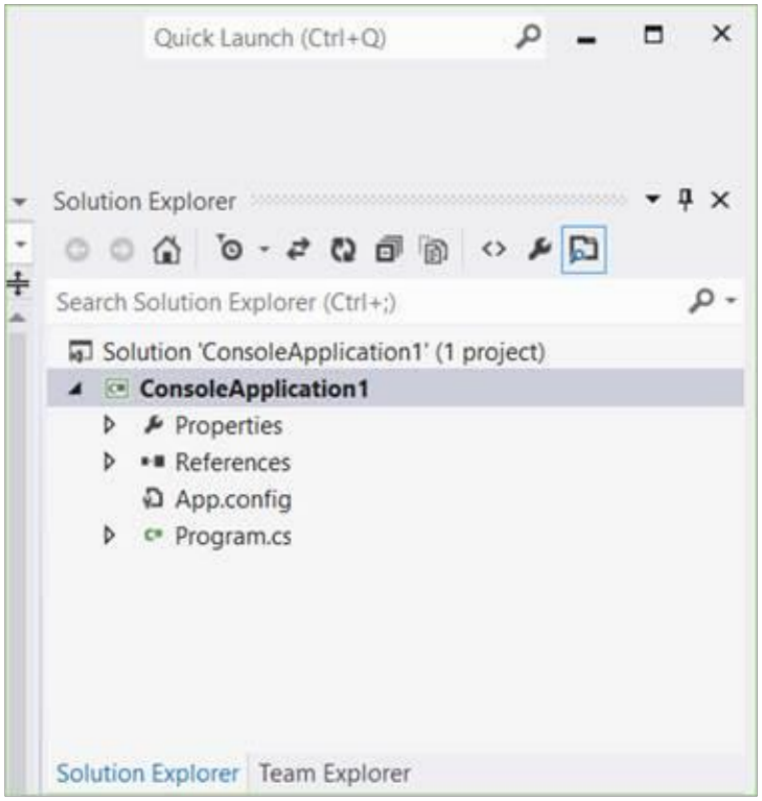
پنجره لیست خطاها (Error List)

در صورتی که برنامه اشکال تایپی یا ساختاری داشته باشد، خطاها و اشکالات برنامه در این پنجره مانند شکل زیر لیست می شوند. پس از ترجمهٔ برنامه باید به این پنجره نگاه کنیم تا خطاهای احتمالی برنامه را متوجه شویم.



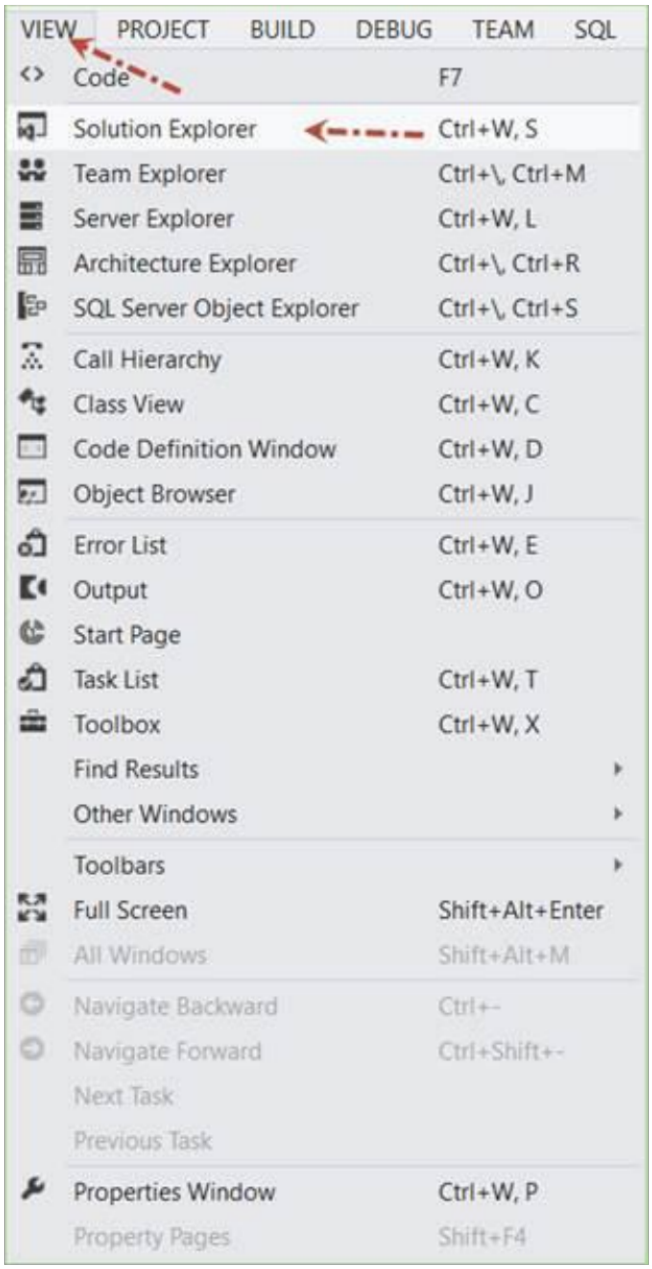
پنجره (Solution Explorer)

سمت راست صفحه، پنجره ای قرار دارد که ساختار پروژه و تمام فایل های موجود در آن را نشان می دهد. اگر پروژه ای باز نباشد، محتوای این پنجره خالی است. ما نام این پنجره را مرورگر پروژه می نامیم و به وسیلهٔ آن به تمام اجزای پروژه دسترسی داریم.

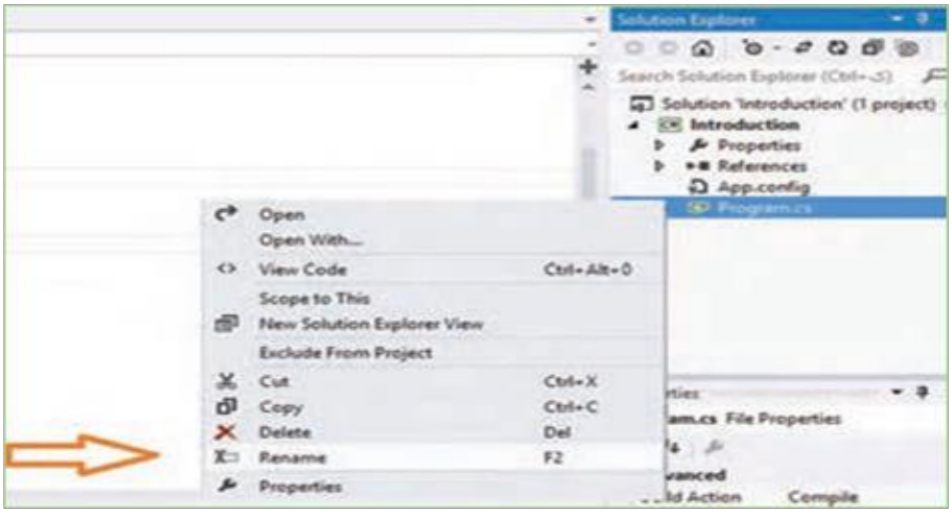


اگر پنجرهٔ مرورگر پروژه باز نیست، و آن را مشاهده نمی کنید از منوی View در بالای صفحه استفاده کنید.

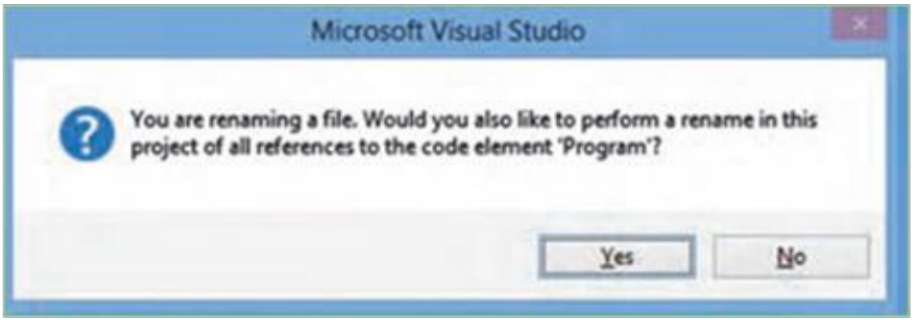
در این منو، نام تمام پنجره ها در شکل زیر مشاهده می شود، روی گزینه Solution Explorer کلیک کنید تا این پنجره دیده شود.



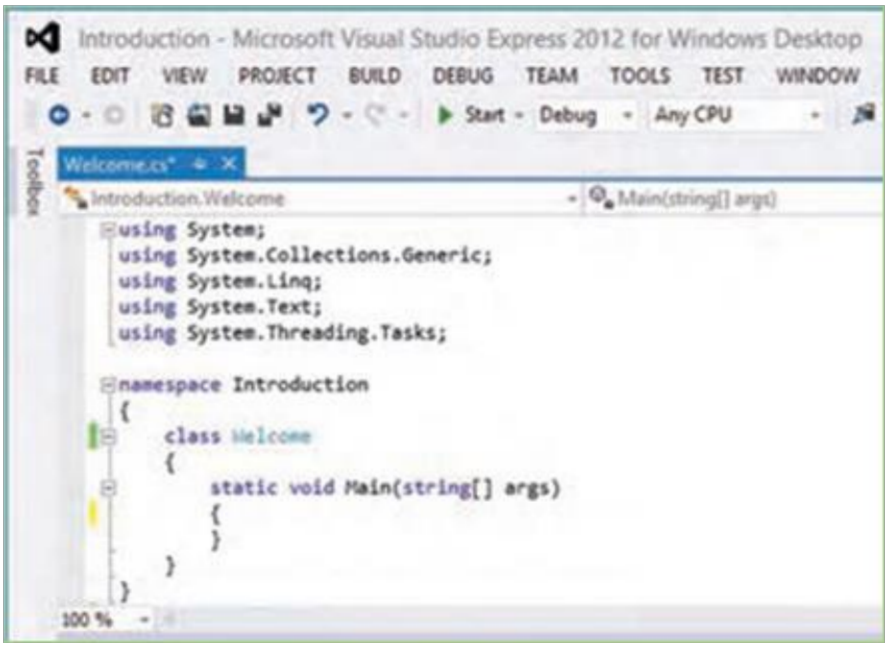
بعد از ایجاد یک پروژه جدید و وارد شدن به محیط برنامه نویسی (IDE)، اگر به پنجره کاوشگر Solution توجه کنید، داخل شاخه پروژه Introduction، یک فایل به نام program.cs وجود دارد. (پسوند .cs نشان دهنده برنامه به زبان C Sharp می باشد) این فایل، فایل متن برنامه است و به طور خودکار تولید شده است. محتوای این فایل در پنجره ویرایشگر برنامه نشان داده شده است. نام این فایل را می توانید مطابق با عملکرد برنامه تغییر دهید و یک نام مناسب برای آن انتخاب کنید که با دیدن نام فایل به عملکرد آن پی ببرید. در این مثال می خواهیم برنامه ای بنویسیم که یک پیام خوش آمدگویی مانند فصل قبل نمایش دهد بنابراین برای تغییر نام فایل Program.cs روی آن کلیک راست کرده و نام دلخواه خود مثلاً Welcome.cs را وارد می کنیم (شکل زیر).



پس از تغییر نام و زدن کلید Enter پنجره ای مانند شکل زیر باز می شود و سؤال می شود « آیا این تغییر نام در تمام مکان هایی که به این نام رجوع می شود نیز اعمال شود؟» اگر پاسخ مثبت یعنی Yes را انتخاب کنید مشاهده خواهید کرد که در متن برنامه، نام کلاس که قبلاً Program بود به Welcome تغییر نام می دهد.



حال به پنجره کد، باز می گردیم، همان طور که در شکل زیر ملاحظه می کنید ساختار کلی یک برنامه به وسیلهٔ VS برای شما آماده می شود که تعدادی دستور در آن موجود است.



در چند خط بالای برنامه، راهنمایی هایی برای مترجم یعنی دستورات using نوشته شده است و جلوی هر کدام، یک فضای نامی ذکر شده است. فضای نامی System برای شما آشنا است زیرا در گام قبل دستور using System را در بالای برنامه نوشتیم و این کار سبب شد که استفاده و نوشتن متدهای مربوط به Console در برنامه ساده تر شود. در حال حاضر می توانید دستورات دیگر using را پاک کنید چون فعلاً به کلاسی غیر از Console نیاز نداریم.

بعد از دستورات using، دستور namespace به همراه نام پروژه (Introduction)نوشته شده است و علامت های آکولاد باز و بسته کل برنامه را دربرگرفته است. با دستور namespace یک فضای نامی جدید تعریف می شود که برای سازماندهی و دسته بندی پروژه های بزرگ مورد استفاده دارد. فعلاً به آن کار نداریم ولی لازم نیست آن را پاک کنید، بنابراین تغییری روی آن انجام ندهید.

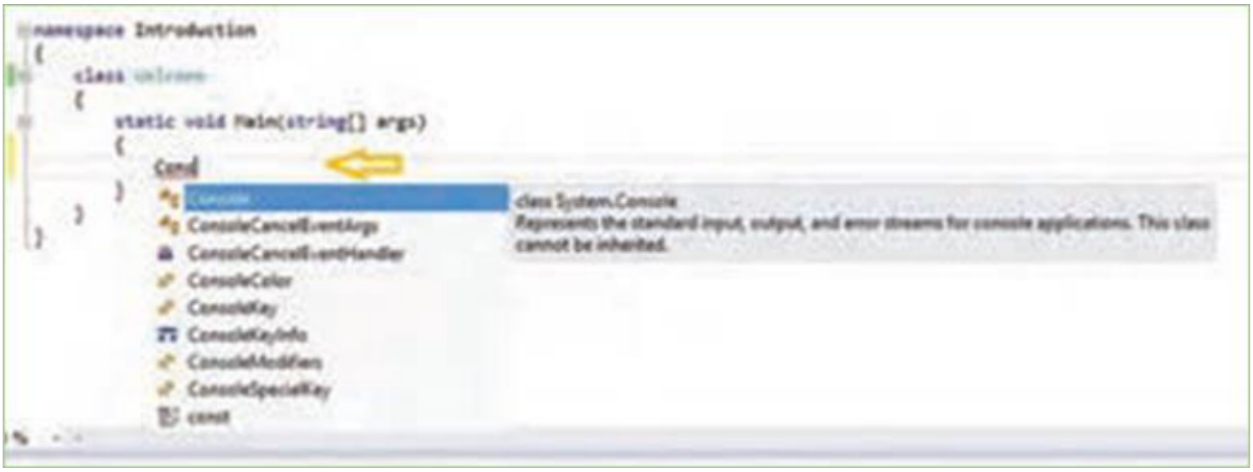
در داخل این فضای نامی، دستور class و سپس در داخل آن متد Main نوشته شده است. نام کلاس Welcome است مگر این که در مرحله قبل نام برنامه را تغییر نداده باشید که در این صورت نام کلاس Program است. متد Main() نیز در داخل کلاس دیده می شود ولی داخل پرانتزهای آن عبارت string[]

args نوشته شده است. می توانید این عبارت را از داخل پرانتزها پاک کنید تا برنامه ساده تر شود همان طور که در گام قبل داخل پرانتزها خالی بود.

داخل متد Main، بین آکولادها فضا ایجاد کنید (این کار را با کلیک کردن در بین دو علامت آکولاد و سپس با زدن کلید Enter انجام دهید) تا بتوانید دستورات مورد نظر خود را بنویسید. مثلاً دستورات زیر را بنویسید:

```
Console.WriteLine(''Hello World'');
Console.WriteLine(''Welcome to C#'');
```

هنگامی که مشغول تایپ برنامه هستید باید تفاوت قابل ملاحظه ای را با روش فصل قبل که برنامه را در محیط Notepad و یا در برنامه Notepad++ می نوشتید احساس کنید. اولاً کلمات با توجه به نوع آنها رنگی نوشته شده اند، مثلاً کلمات کلیدی با رنگ آبی نشان داده شده اند و ضمناً در هنگام تایپ برنامه به محض نوشتن یک حرف کلمه Console لیستی از کلمات مشابه نمایش داده می شود. با تایپ چند حرف دیگر، کلمه Console در لیست نشان داده می شود و در کنار آن یک توضیح مختصر دیده می شود.شکل زیر را مشاهده کنید.



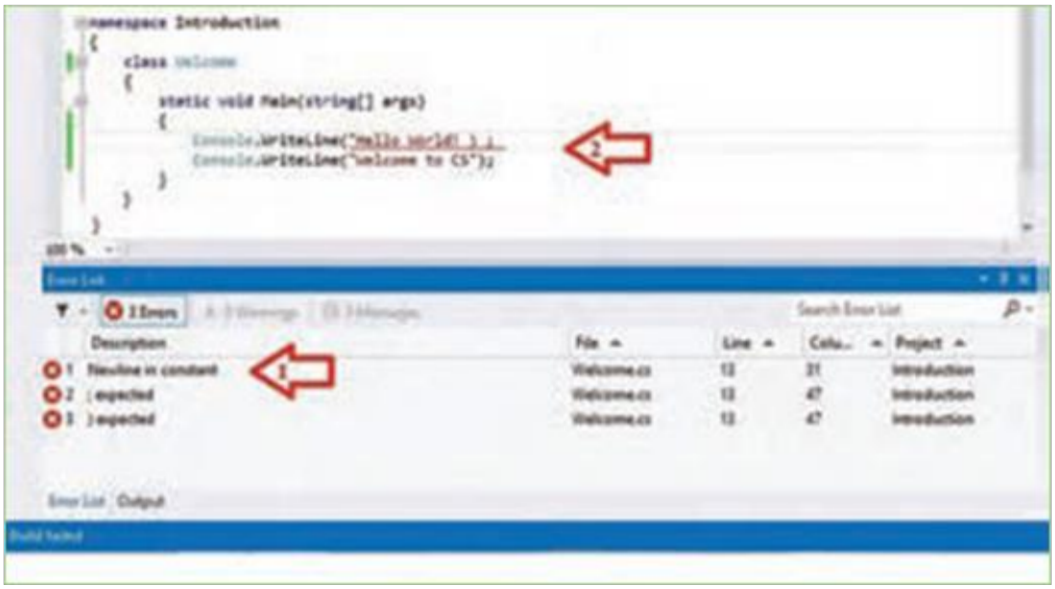
هرگاه کلمه مورد نظر شما در لیست دیده شد و نوار آبی رنگ روی آن قرار گرفت می توانید تایپ آن کلمه را رها کنید و کلید Enter را بزنید. در این صورت کلمه به طور کامل تایپ می شود و می توانید دنبالهٔ دستور را بنویسید. این کار باعث می شود سرعت شما در تایپ برنامه به طور چشمگیری افزایش یابد. اگر در تایپ یک دستور غلط املایی داشته باشید و یا قوانین برنامه نویسی زبان C# را رعایت نکنید در این صورت، یک

خط قرمز رنگ، زیر کلمه یا مکانی که در آن اشتباه وجود دارد کشیده می شود و به شما یادآوری می کند که در آن مکان یک خطا وجود دارد و شما باید آن را برطرف کنید.

مثلاً در شکل بالا مشاهده می کنید که یک خط قرمز در زیر عبارت Hello World! کشیده شده است. با کمی دقت متوجه می شوید که علامت نقل قول انتهای پیام فراموش شده است! چون پیام ها به عنوان یک رشته باید بین علامت های نقل قول قرار داشته باشند.

ترجمه برنامه

بعد از نوشتن دستورات بالا، برای ترجمهٔ برنامه، کلید F۷ یا F۶ را بزنید. اگر برنامه خطا داشته باشد، خطاها در پنجره Error List دیده می شود. برای مثال شکل زیر خطاهای برنامه را در حالتی نشان می دهد که فراموش کرده اید انتهای رشته را با علامت " مشخص نمایید.در این شکل، پنجرهٔ Error List چندخطا را نشان می دهد؟



شاید این مسئله تا حدودی تعجب آور باشد که شما در تایپ برنامه، یک علامت نقل قول را فراموش کرده اید، اما در پنجره Error List، سه خطا گزارش می شود. بنابراین در ترجمه یک برنامه، انتظار گزارش تعداد خطای زیاد را داشته باشید. در حالت بروز خطا، باید با خواندن توضیح خطا و شماره خط برنامه که گزارش می شود، اقدام به رفع اشکال برنامه کنید. همواره از خطای اول شروع کنید، بنابراین در پنجره Error List روی خطای اول دابل کلیک کنید تا به طور مستقیم به پنجره ویرایشگر برنامه و محل خطای مزبور هدایت شوید. در این صورت اقدام به رفع اشکال کنید و سپس برنامه را دوباره ترجمه کنید. اگر خطایی گزارش شد باز سراغ خطای اول بروید و برنامه را تصحیح کنید.

پس از رفع اشکالات برنامه اگر دوباره عمل ترجمه را انجام دهید، در پنجرهٔ خروجی (Output) پیام ترجمهٔ بدون خطا و موفق را مانند شکل زیر مشاهده خواهید کرد.

اجرای برنامه

بعد از ترجمه صحیح و بدون خطا، می توانید برنامه را اجرا کنید. برای اجرای برنامه از منوی DEBUG گزینه Start Without Debugging و یا از ترکیب کلیدی Ctrl + F۵ استفاده کنید. با اجرای برنامه صفحه کنسول با سه پیام مانند شکل زیر دیده می شود. دو پیام خوش آمدگویی، همان هایی هستند که شما نوشته اید. اما پیام سوم Press any key to continue... به طور خودکار به وسیلهٔ VS در صفحه کنسول اضافه می شود تا صفحه کنسول بسته نشود و شما فرصت دیدن پیام ها را داشته باشید. با زدن یک کلید دلخواه، پنجره کنسول بسته می شود زیرا دستور دیگری در برنامه وجود ندارد.

گام سوم

آشنایی با انواع داده ها و متغیرها

متغیر چیست؟

در هر کامپیوتر حافظه های مختلفی وجود دارد که هر یک برای انجام کار خاصی پیش بینی شده است. یک نوع از حافظه ی کامپیوتر که قادر است داده ها را نگهداری کند و به سرعت قابل دسترسی است، حافظه موقتیRAM است. از اطلاعات درون حافظه RAM، در هر لحظه می توان با اطلاع شد و یا در صورت لزوم محتویات آن را تغییر داد یا مقدار جدیدی را در آن ذخیره کرد.

با توجه به مطالب گفته شده، لازم است در یک برنامه، یک یا چند مکان (بایت) از حافظه RAM کامپیوتر برای نگهداری موقتی داده ها یا نتایج حاصل از پردازش مورد استفاده قرار گیرد. در زبان های برنامه نویسی به این مکان ها، متغیر گفته می شود زیرا می توان محتوای آنها را در طول اجرای برنامه تغییر داد.

٪ متغیر: مکانی از حافظه RAM کامپیوتر است که برای نگهداری موقتی داده ها یا اطلاعات استفاده می شود. متغیر را مانند یک ظرف در نظر بگیرید. در آشپزخانه ظروف متعددی با شکل ظاهری، اندازه و جنس مختلفی وجود دارد که هر یک برای نگهداری یک نوع غذا یا مایعات استفاده می شود که گنجایش آن را داشته باشد. در یک برنامه نیز برای نگهداری هر یک از داده ها با توجه به نوع و بزرگی داده، باید از متغیر مناسبی استفاده کنیم که بتواند داده را نگهداری کند.

روش اعلان (تعریف) و ایجاد متغیرها

قبل از اینکه بتوانید مقداری را در یک متغیر ذخیره کنید باید متغیری را ایجاد کنید که قادر باشد آن مقدار را به درستی ذخیره نماید. در هنگام ایجاد متغیر باید نوع متغیر را مشخص نمایید. در زبان C# برای ایجاد و مشخص کردن نوع متغیر، از الگوی زیر استفاده می شود.

نام متغیر نوع داده;

دستور زیر را در نظر بگیر:

```
int a;
```

در این دستور متغیر a از نوع عدد صحیح اعلان می شود. کلمه int نوع متغیر را مشخص می کند که قادر است اعداد صحیح را در خود نگهداری کند و a نام متغیر است. نام متغیر به وسیلهٔ برنامه نویس انتخاب می شود که بهتر است نام و نوعِ آن مطابق با داده ای باشد که مقداردهی می شود.

نوع داده (نوع متغیر)

نوع متغیر به طور کلی ۳ ویژگی را مشخص می کند:

۱- گنجایش یا ظرفیت متغیر: مثلاً نوع int چهار بایت است.

۲- نوع اطلاعاتی که در متغیر می توان ذخیره کرد: مثلاً در متغیر نوع int فقط اعداد صحیح و بدون ممیز قابل نگهداری است.

۳- چه عملیاتی را می توان بر روی آن انجام داد: مثلاً عملیات ریاضی معمول را می توان روی اعداد نوع int انجام داد.

در زبان C# علاوه بر نوع داده int، انواع دیگری از داده ها نیز دسته بندی و گروه بندی شده اند و نحوه نمایش یا نگهداری آنها در حافظه و عملیاتی که می توان بر روی آنها انجام داد از قبل مشخص و تعریف شده است و برای هر دسته یا گروه از داده ها، یک نام انتخاب شده است که به آن نوع داده اولیه یا درون ساخته می گویند.

جدول زیر انواع داده و مشخصات هر یک را نشان می دهد.

	نوع داده	کاربرد نوع داده	مقدار حافظه	کمترین مقدار	بیشترین مقدار
عددی	sbyte	اعداد صحیح	1	-128	127
	byte	اعداد صحیح مثبت	1	0	255
	short	اعداد صحیح	2	-32768	32767
	ushort	اعداد صحیح مثبت	2	0	65535
	int	اعداد صحیح	4	- 2147483648	2147483647
	uint	اعداد صحیح مثبت	4	0	4294967295
	long	اعداد صحیح	8	- 9223372036854778508	9223372036854778507
	ulong	اعداد صحیح مثبت	8	0	18446744073709551615
	float	اعداد صحیح	4	-3.402823×10^{38}	3.402823×10^{38}
	double	*اعداد اعشاری با دقت زیاد*	*8*	$-1.79769313486232 \times 10^{308}$	$1.79769313486232 \times 10^{308}$
	decimal	اعداد صحیح بزرگ اعداد اعشاری با دقت بسیار زیاد	16	$-79228162514264337593543950335 - 7.9 \times 10^{28}$	$79228162514264337593543950335 + 7.9 \times 10^{28}$
غیر عددی	bool	مقدار منطقی	1	false	true
	char	یک حرف یا علامت (کراکتر)	2	0 کد کراکتر Unicode مطابق با سیستم	65535 کد کراکتر Unicode مطابق با سیستم
	string	رشته		-	-
	object	آدرس یک داده		-	-

برای مثال در جدول بالا نوع داده sbyte را در نظر بگیرید. این نوع داده، اعداد صحیح و بدون ممیز در محدوده ۱۲۸- تا ۱۲۷ را شامل می شود که یک بایت حافظه را اشغال می کند و بر روی آنها می توان عملیات

ریاضی را انجام داد. اگر در یک برنامه، متغیری از نوع sbyte را استفاده کنیم، قادر خواهیم بود به عنوان مثال عدد ۷۸ را در آن ذخیره کنیم. اما نمی توان عدد ۲۰۶ و یا عدد ۵/۱ را در آن نگهداری کرد. همچنین نوع داده byte اعداد صحیحی فقط در محدوده ۰ تا ۲۵۵ را شامل می شود که در یک بایت قرار می گیرد. در این نوع داده فقط اعداد مثبت یا بدون علامت قابل نمایش می باشند.

دستور byte age ; متغیری به نام age ایجاد می کند که این متغیر بسیار کوچک و به ظرفیت یک بایت است و می تواند یکی از اعداد صفر تا ۲۵۵ را در خود ذخیره کند.

اگر بخواهید چند متغیر از یک نوع را تعریف کنید کافی است بعد از ذکر نوع داده، نام متغیرها را با علامت ویرگول از یکدیگر جدا کنید. مثلاً برای تعریف دو متغیر برای نگهداری حداقل و حداکثر درجه حرارت از دستور زیر استفاده می کنیم:

```
sbyte minTemp , maxTemp;
```

./اگر متغیر مقدار دهی نشود خطا می گیرد. هر نوع داده، مجموعه ای از مقادیر به همراه مجموعه ای از عملیات را مشخص می کند.

برای اعداد صحیح و بدون ممیز نوع داده های زیر استفاده می شود:

```
sbyte , byte , short , ushort , int , uint , long , ulong
```

و برای اعداد اعشاری می توانید از نوع داده های float وdouble استفاده کنید. نوع داده float برای اعداد اعشاری با دقت حداکثر ۷ رقم استفاده می شود. در صورتی که ارقام عدد بیش از آن باشد عدد گرد می شود. مثلاً عدد ۴۵۶۷۸۹/۱۲۳ به صورت عدد ۱۲۳/ ۴۵۶۸ قابل نگهداری است.

نوع داده double برای اعداد اعشاری بسیار بزرگ و یا بسیار کوچک مانند جرم و بار الکتریکی یک الکترون و با دقت زیاد ۱۵ رقم استفاده می شود.

./ در زبان برنامه نویسی C#، قبل از اینکه بتوانید داده ای را در یک متغیر ذخیره کنید باید متغیر را ایجاد (یا اعلان) کنید و در هنگام ایجاد کردن یک متغیر، باید نوع متغیر(نوع داده) را مشخص نمایید. مثال:

```
float mark;
```

مقداردهی متغیرها

پس از تعریف یا ایجاد متغیر، می توانید در آن، مقداری را با توجه به نوع متغیر ذخیره کنید. توجه داشته باشید که در یک متغیر همواره فقط یک مقدار نگهداری می شود و با ذخیره کردن داده جدید در یک متغیر، مقدار قبلی آن از بین می رود. مقداردهی متغیرها به چند روش صورت انجام می گیرد. با دستور زیر مستقیماً مقداری در متغیر قرار می گیرد به این دستور، دستور انتساب می گویند.

نام متغیر = مقدار;

دستورات زیر را در نظر بگیرید:

```
byte age;
age = ۱۶;
```

متغیر age از نوع عدد صحیح اعلان شده و با عدد ۱۶ مقداردهی شده است.

در هنگام تعریف یا ایجاد متغیر نیز می توانید آن را مستقیما مقداردهی کنید که به آن مقداردهی اولیه می گویند. الگوی آن چنین است:

نام متغیر نوع داده = مقدار;

بنابراین دو دستور قبل را با الگوی بالا جایگزین می کنیم:

```
byte age = ۱۶;
```

٪

۱- برای مشخص کردن اعداد مثبت نیازی به قراردادن علامت + در پشت عدد نیست.

۲- در بین ارقام عدد نباید ویرگول قرار دهید تا ارقام عدد، دسته بندی و جدا شوند.

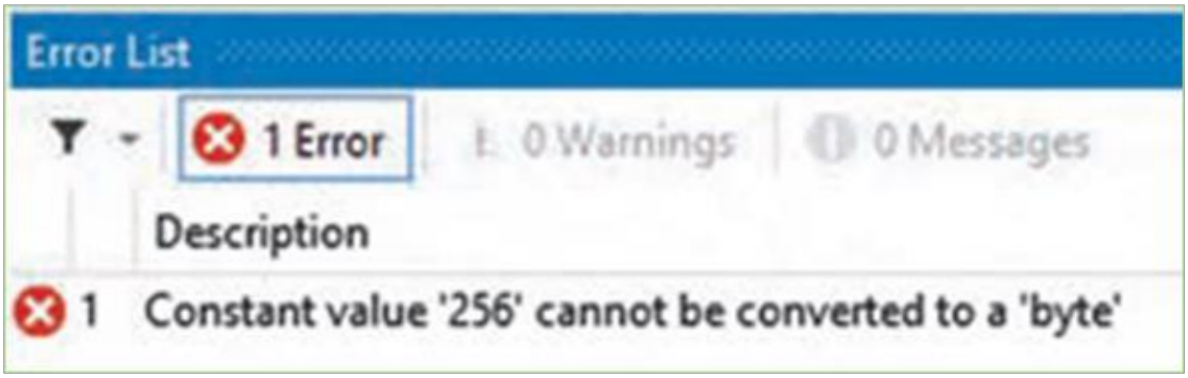
۳- اگر عددی را بخواهید در داخل یک متغیر ذخیره کنید که خارج از ظرفیت و گنجایش آن متغیر باشد، مترجم متوجه آن شده و اجازه نمی دهد.

مثلاً دستور انتساب زیر را در نظر بگیرید:

```
byte age = ۲۵۶;
```

با توجه به ظرفیت متغیر age که حداکثر عدد ۲۵۵ است، در هنگام ترجمه این دستور، خطای شکل زیر ظاهر می شود که شرح آن چنین است:

« مقدار ثابت ۲۵۶ را نمی تواند به یک byte تبدیل شود».



در یک برنامه به زبان C# می توانید اعداد صحیح را در مبنای ۱۶ نیز بنویسید. برای این منظور قبل از عدد مورد نظر از پیشوند ۰x یا ۰X استفاده کنید که نشانه اعداد مبنای ۱۶ می باشد. مثلاً:

```
byte portValue = ۰x۱B;
ushort portAddress = ۰X۰۰FF;
```

با اجرای این دستورات در متغیر portValue عدد ۲۷ و در متغیر portAddress عدد ۲۵۵ قرار می گیرد.

برای مشخص کردن انواع عددی دیگر از نشانه های جدول زیر استفاده می شود که در انتهای عدد ذکر می شود.

نوع عدد	نشانه	مثال
عدد صحیح مثبت unit	U یا u	125U
عدد صحیح بزرگ	L یا l	1700L
عدد صحیح بزرگ مثبت	UL	250000UL
عدد اعشاری با دقت معمولی	F یا f	2.5f
عدد اعشاری با دقت زیاد	D یا d	12.75d
عدد بسیار بزرگ	M یا m	12345678M

٪ اگر در برنامه، یک عدد اعشاری بدون نشانه بنویسید این عدد به عنوان عدداعشاری با دقت زیاد در نظر گرفته می شود.

برای ذخیره اعداد اعشاری باید ازمتغیرهای نوع float یا double استفاده کنید. مثلاً برای نگهداری نمرات درسی(معمولاً با دو رقم اعشار) یا اعداد گنگ مانند π باید از چنین متغیرهایی استفاده کرد.

دستور زیر را در نظر بگیرید:

```
double PI = ۳.۱۴۱۵۹۲۶۵۳۵۸۹۷۹۳۲۳۸;
```

در این دستور برای نگهداری عدد π متغیر PI با دقت زیاد اعلان و مقداردهی شده است.

٪برای ذخیره اغلب داده ها مانند نمره یک درس، متغیر float مناسب است. اگر چه می توانید از متغیر نوع double نیز استفاده کنید ولی حافظهٔ اشغالی این متغیر دو برابر متغیر نوع float است.

دستورات زیر را در نظر بگیرید:

```
float myPhysicMark;
myPhysicMark = ۱۷.۷۵f;
```

در دستورات بالا برای ذخیرهٔ نمرهٔ درس فیزیک متغیری اعلان و مقداردهی شده است.

٪ در زبان C# هر عدد اعشاری داخل برنامه، به وسیلهٔ مترجم به عنوان نوع double در نظر گرفته می شود. بنابراین اگر بخواهید یک عدد ممیزی را در یک متغیر نوع float ذخیره کنید مترجم خطا یا هشدار می دهد. برای جلوگیری از این مسئله باید از متغیرهای نوع double در هنگام کار با اعداد اعشاری استفاده کنید و یا اینکه در جلوی اعداد اعشاری حرف F یا f را بنویسید تا مترجم، این عدد را به عنوان یک عدد نوع float در نظر بگیرد.

نشان دادن محتوای متغیرها بر روی صفحه نمایش

معمولاً در برنامه ها لازم است محتوای متغیرها که شامل داده ها و یا نتایج پردازش با اطلاعات بر روی صفحه نشان داده شود تا کاربر از آنها آگاه شود. بدین منظور از Write() یا WriteLine() استفاده می کنیم که در گام های قبلی برای نمایش یک پیام یا حاصل یک عبارت به کار گرفته شد.

مثلاً برای نشان دادن محتوای متغیر age دستور زیر را می نویسیم:

```
byte age = ۱۶;
System.Console.WriteLine( age );
```

با توجه به اینکه در متغیر age عدد ۱۶ قرار دارد با اجرای دستور بالا، این عدد روی صفحه کنسول نشان داده می شود.

اگر شخص دیگری غیر از شما، این عدد را روی صفحه مشاهده کند، شاید متوجه نشود که این عدد چیست و شاید عدد ۱۶ را به عنوان نمره در نظر بگیرد. بنابراین بهتر است قبل از نمایش هرعدد، یک پیام (رشته) نیز نشان داده شود و به صورت کوتاه و مختصر منظور و مفهوم عددی را که قرار است روی صفحه نشان داده شود بیان کند. بنابراین دستور بالا را به صورت زیر می نویسیم:

```
System.Console.WriteLine("My age is " + age );
```

با اجرای این دستور، عبارت زیر روی صفحه نشان داده می شود:

```
My age is ۱۶
```

علامت + در دستور بالا، به معنای عمل جمع ریاضی نیست بلکه به منظور کنار هم قرار دادن این دو مقدار (رشته ها) استفاده شده است. همان طور که در دستور زیر نیز از علامت + استفاده شده است:

```
System.Console.WriteLine("I am " + age + "years old." );
```

با اجرای این دستور، عبارت زیر روی صفحه نشان داده می شود:

```
I am ۱۶ years old.
```

مثال- استفاده از چند متغیر صحیح و اعشاری در برنامه زیر ، نشان داده شده است:

```
class VariableDemo
{
static void Main()
{
// Declare some integer numbers variables
```

```
int a = ۱۰ , b = ۲۰ ,c;

c = a + b;

Console.WriteLine("a= " + a);

Console.WriteLine("b= " + b );

Console.WriteLine("a + b = " + c);

// Declare some real numbers variables

float lowPI = ۳.۱۴۱۵۹۲۶۵۳۵۸۹۷۹۳۲۳۸f;

double highPI = ۳.۱۴۱۵۹۲۶۵۳۵۸۹۷۹۳۲۳۸;

// Print the results on the console

Console.WriteLine("Float PI is: " + lowPI);

Console.WriteLine("Double PI is: " + highPI);

Console.ReadKey();

}

}
```

در برنامه بالا، سه متغیر a,b,c از نوع عدد صحیح تعریف شده اند و در متغیر c نتیجه حاصل جمع دو عدد a و b قرار می گیرد. در دو متغیر اعشاری lowPI و highPI عدد با دقت های مختلف نگهداری شده است.

نحوه نام گذاری متغیرها

همان طور که پدر و مادر برای انتخاب یک نام خوب و مناسب برای فرزند خود، وقت زیادی می گذارند و نکاتی از جمله زیبایی نام و با معنا بودن را رعایت می کنند و همچنین سعی می کنند که این نام قبلاً در خانواده و یا نزدیکان انتخاب نشده باشد، به همان صورت برنامه نویس نیز برای متغیرها باید یک نام صحیح، بامعنا و غیرتکراری در محدوده آن را انتخاب کند این کار باید با حوصله انجام شود و نام انتخابی نباید با نام های دیگر یکسان باشد.

در زبان C# در نام گذاری متغیرها، رعایت موارد زیر الزامی است:

۱- استفاده از حروف الفبا، اعداد و کاراکتر زیرخط، مجاز است.

۲- نام متغیر نمی تواند با عدد شروع شود.

۳- نام انتخابی نمی تواند با کلمات کلیدی و یا رزرو شده باشد.

۴- استفاده ازعلامت فاصله و خط تیره در نام متغیر مجاز نیست.

در انتخاب نام متغیرها، بهتر است نکات زیر رعایت شود:

۱- نام با معنی و با توجه به کاربرد متغیر در برنامه انتخاب شود. مانند woodLength

۲- از نام های مخفف استفاده نکنید چون خواندن آنها مشکل است. مانند crntStdnt

۳- اولین حرف نام متغیر را با حروف کوچک شروع کنید و اگر نام متغیر از چند کلمه تشکیل شده، برای خوانایی، حرف اول کلمات بعدی را با حروف بزرگ بنویسید. به این روشِ نوشتن نام، کوهان شتری می گویند.

چند نمونه نام متغیر دو کلمه ای به روش کوهان شتری را ملاحظه می کنید:

fileName , userName , notFound , localIP

روش دیگری برای نام گذاری متغیرها به نام روش مجارستانی به وسیلهٔ آقای چارلز سیمونیی ابداع شده که در ابتدای نام متغیر، مخفف نوع داده ذکر می شود که یک روش شناخته شده و معروف برای نام گذاری متغیرها است.

چند نمونه نام متغیر دو کلمه ای، به روش مجارستانی را ملاحظه می کنید:

IntNumber, LngSalary, BlnStatus

در این کتاب از روش کوهان شتری برای نام گذاری متغیرها استفاده شده است.

٪ با توجه به حساسیت زبان C# به حروف کوچک و بزرگ، در نام گذاری متغیرها دقت کنید که متغیر a و A مستقل هستند.

در جدول زیر تعدادی نام متغیر و علت مجاز یا غیرمجاز بودن این نام ها را می بینید.

مثال	توضیح
1a	غیر مجاز، نام متغیر نباید با عدد شروع شود
a1	مجاز
employee Salary	غیر مجاز، بین کلمات نباید فاصله وجود داشته باشد
First	مجاز
Hello!	غیر مجاز، علامت تعجب نباید در نام وجود داشته باشد
payRate	مجاز
one+two	غیر مجاز، علامت + در یک نام نباید قرار داشته باشد
Conversion	مجاز
counter_1	مجاز
2nd	غیر مجاز، نام نمی تواند با عدد شروع شود

در دستورات زیر، چند نمونه از اعلان و مقداردهی متغیرها را مشاهده می کنید:

تعریف متغیر برای نگهداری سرعت خودرو با مقداردهی اولیه ۷۰= intspeed; //

تعریف سه متغیر برای اضلاع مثلث // float a , b , c ;Triangle sides

تعریف یک متغیر برای نگهداری مساحت مثلث // float triangleArea;

متغیری برای نگهداری بار الکتریکی یک جسم // double electricalCharge;

کار با اعداد اعشاری

در فیزیک و شیمی و یا به طور کلی در علوم، با اعداد بسیار کوچک و بسیار بزرگ سروکار داریم. اگربخواهید عدد اعشاری بسیار کوچک و یا بسیار بزرگی را در یک متغیر ذخیره کنید، می توانید آن را به صورت کوتاه با روشی شبیه نماد علمی بنویسید. برای اینکه با این روش آشنا شوید ابتدا لازم است روش نماد علمی را یادآوری کنیم.

در روش نماد علمی، هر عدد از ۲ بخش تشکیل می شود که با علامت ضرب از یکدیگر جدا شده اند. بخش اول یک عدد اعشاری بین ۱ تا ۹ است (فقط یک رقم صحیح دارد) که به آن مانتیس می گویند و قسمت دوم که به صورت توانی از عدد ۱۰ است که به آن نما گفته می شود (جدول زیر).

فرم نماد علمی	فرم معمولی
4.38×10^{6}	4380000
2.65×10^{-5}	.0000265
4.79832×10^{1}	47.9832
1×10^{7}	10000000
-5.6×10^{3}	-5600

فرم نقطه شناور

در زبانC# از یک فرم نماد علمی برای نمایش اعداد اعشاری استفاده می شود که به آن فرم نقطه شناور گفته می شود. در این فرم مانند نماد علمی، عدد از دو بخش مانتیس و نما تشکیل شده است که با حرف E از یکدیگر جدا شده اند. در این فرم، توان ۱۰ بعد از حرف E نوشته می شود و خبری از علامت ضرب بین دو قسمت نیست(جدول زیر).

نمایش عدد در فرم نقطه شناور	عدد
7.5924E1	75.924
1.8E-1	0.18
4.53E - 5	0.0000453
-1.482E0	-1.482
7.8E3	7800.0

بار الکتریکی یک الکترون ۱۹-۱۰×۶۰۲/۱ کولن است که در متغیرهای زیر ذخیره شده است. می توانید این عدد بسیار کوچک را در یک متغیر نوع double یا float به صورت نقطه شناور ذخیره کنید.

```
double electricalCharge = ۱.۶۰۲E -۱۹;
float electricalCharges = ۱.۶۰۲E -۱۹F;
```

دقت اعداد قابل نمایش در فرم نقطه شناور

حداکثر تعداد ارقام غیر صفر و با معنی مانتیس عدد را، دقت عدد می نامند. دقت اعداد نوع float ۷ رقم و اعداد نوع double ۱۵ رقم است.

٪ به غیر از میزان حافظه مصرفی و محدوده اعداد قابل نمایش در نوع داده های float و double، میزان دقت این دو نوع داده نیز با یکدیگر متفاوت است.

نوع داده منطقی یا بولین (bool)

در انتهای جدول ابتدای این گام نوع داده منطقی یا بولین (bool) را مشاهده می کنید این نوع داده فقط شامل دو مقدار درست (true) و نادرست (false)است. متغیرهایی که از این نوع داده تعریف و ایجاد می شوند، قادرند یکی از دو مقدار true و false را بپذیرند که با حروف کوچک انگلیسی نوشته می شوند.

دستورات زیر متغیر response را اعلان و با false مقدار دهیِ اولیه می کند. سپس محتوای متغیر بر روی صفحه نمایش چاپ می شود.

```
bool response = false;
System.Console.WriteLine(response );
```

نوع داده حرفی یا کاراکتری char

کاراکترعبارت است از یک حرف الفباء یا یک علامت و یا نشانه هایی مانند آنچه که در روی دکمه های صفحه کلید مشاهده می کنید. در کامپیوتر برای هر دکمه صفحه کلید یک کدِ عددی در نظرگرفته می شود و در واقع هنگامی که یک کلید را فشار می دهید کدی متناظر با آن کلید تولید و این کد به صورت دنباله ای از صفر و یک در حافظه کامپیوتر ذخیره می شود. یک کاراکتر را می توانید با کد آن مشخص کنید و یا علامت آن را در بین علائم ' ' (تک کوتیشن) قرار دهید.

چند نمونه از کاراکترها را در زیر مشاهده می کنید.

'A' , 'a' , '&' , '$' , '+' , ' '

٪

۱- در داخل علامت ها فاصله (Space) نیز به عنوان یک کاراکتر در نظر گرفته می شود.

۲- در داده کاراکتری، فقط یک کاراکتر باید بین علائم ' ' وجود داشته باشد.

توجه داشته باشید که در زبان برنامه نویسی C#، نوع داده char به منظور کار با داده های کاراکتری پیش بینی شده است. اگر بخواهید یک کاراکتر را در یک متغیر ذخیره کنید باید متغیری از نوع داده char تعریف کنید.

گنجایش این متغیر، دو بایت است و کد کاراکتر را نگهداری می کند.

char نام متغیر;

در دستور زیر، متغیری به نام ch از نوع char تعریف و حرف A در آن ذخیره شده است.

```
char ch = 'A';
```

متغیر ch یک متغیر دو بایتی است که در آن کد کاراکتر نگهداری می شود. این کد دو بایتی طبق استاندارد یونیکد (Unicode) است. در استاندارد یونیکد، کد هر کاراکترعددی بین ۰ تا ۶۵۵۳۵ است و تمام نشانه ها، علائم و حروف الفباء زبان های مختلف کشورها به وسیلهٔ این استاندارد کدبندی شده است. این کدبندی مستقل از سیستم عامل، زبان برنامه نویسی و سخت افزار است.

در برنامه می توانید به جای قرار دادن کاراکتر در علائم ' ' از کد آنها استفاده کنید، چون کاراکتر ها فقط محدود به آنچه که بر روی صفحه کلید قرار دارد نیستند. بنابراین با دانستن کد هر کاراکتر می توانید آن را

در برنامه استفاده کنید. معمولاً برای سادگی، این کد را در مبنای ۱۶ ذکر می کنند. با توجه به اینکه در کدبندی یونیکد، از دو بایت استفاده می شود و هر ۴ بیت یک رقم مبنای ۱۶ است، برای نمایش این کد در مبنای ۱۶ از یک عدد ۴ رقمی استفاده می شود. مثلاً کد کاراکتر A عدد ۶۵ در مبنای ۱۰ است.

معادل این کد در مبنای ۱۶ عدد ۴۱ است. این عدد را در داخل علائم ' ' قرار می دهیم و برای مشخص کردن این عدد به عنوان کد کاراکتر، قبل از آن ، علامت \u یا \x را می نویسیم مانند الگوی زیر :

'\u کد ۴رقمی '

در دستور زیر، متغیر ch اعلان و حرف A در آن ذخیره می شود. از صفرهای اضافی قبل از عدد، برای تکمیل کد به صورت ۴ رقمی استفاده شده است.

```
char ch = '\u۰۰۴۱'; // Same as Char ch ='A'
```

نوع داده رشته ای (String)

نوع داده char، تنها برای نگهداری یک کاراکتر مناسب است. برای هنگامی که داده ها، مانند نام یک شخص، بیش از یک کاراکتر است باید از نوع داده رشته ای (String) استفاده کنیم.

یک رشته شامل تعدادی حروف و کاراکتر است که در بین جفت کوتیشن " " قرار گرفته است. مثلاً "Mohammad" یک دادهٔ رشته ای شامل ۸ کاراکتر است.

متغیر رشته ای

برای نگهداری داده های رشته ای در برنامه، باید متغیر رشته ای تعریف کنید. متغیرهای رشته ای قادرند آدرس محلی که یک داده رشته ای وجود دارد را نگهداری کنند یا به عبارت ساده، قادرند داده های رشته ای را ذخیره کنند. بنابراین با متغیری از نوع رشته، قادر خواهیم بود به داده های رشته ای دسترسی داشته باشیم.

دستور زیر یک متغیر رشته ای به نام name را اعلان می کند.

```
string name;
```

و با دستور انتساب زیر، می توانید رشته "Mohammad" را در متغیر name ذخیره کنید و در طول برنامه به آن دسترسی داشته باشید:

```
name = "Mohammad";
```

عملیات بر روی داده ها یا متغیرهای رشته ای

عملیات مختلفی بر روی رشته ها می توان انجام داد، یکی از عملیات معمول و کاربردی، الحاق یا کنارهم قرار دادن رشته ها است. برای الحاق دو رشته از علامت + استفاده می شود.

قطعه کد زیر را در نظر بگیرید. در این کدها، محتوای متغیر رشته ای name با رشته 'Welcome" الحاق شده و حاصل در متغیر message قرار می گیرد.

```
string name ="Mohammad";
string message = "Welcome" + name;
System.Console.WriteLine(message);
```

نتیجه خروجی چنین خواهد بود:

```
WelcomeMohammad
```

./ته با توجه به این که در زبان C#، علامت + هم برای عمل جمع ریاضی و هم برای الحاق رشته ها استفاده می شود، در به کارگیری این علامت در برنامه باید دقت کافی داشته باشید.

برای یادگیری به بررسی چند مثال می پردازیم:

مثال- برنامه زیر برای محاسبه مجموع دو عدد a و b نوشته شده است.

```
class VariableDemo
{
static void Main()
{
// Declare two integer variables
int a , b;
```

```
a = ۱۰;

b = ۱۵;

Console.WriteLine("a= " + a);

Console.WriteLine("b= " + b );

// What is displayed?

Console.WriteLine("a + b = " + a + b);

}

}
```

در خط آخر برنامه بالا، علامت + دو بار استفاده شده است که هر دو علامت، عمل الحاق رشته را انجام می دهند . خروجی این برنامه مطابق شکل زیر است:

برای رفع اشکال برنامه بالا، خط آخر را به صورت زیر بازنویسی می کنیم:

```
Console.WriteLine("a + b = " + (a + b) );
```

با تصحیح خط آخر، نتیجه اجرای برنامه شکل زیر خواهد شد:

دریافت رشته

تاکنون داده های مشخص و ثابتی را در داخل برنامه استفاده کردیم. این داده ها به وسیلهٔ برنامه نویس درون برنامه تعیین شده بود. حال می خواهیم برنامه های خود را کاربردی کنیم و داده ای را از کاربر دریافت کنیم. برای این منظور از متد ReadLine() استفاده می کنیم که به کاربر اجازه می دهد تا داده مورد نظر خود را از طریق صفحه کلید وارد کند.

متد ReadLine() مانند متد هایی که تاکنون خوانده ایم در کلاس Console تعریف شده است و در فضای نامی System قرار دارد .بنابراین به صورت زیر استفاده می شود:

```
System.Console.ReadLine();
```

کامپیوتر با اجرای این متد متوقف شده و منتظر دریافت داده می شود. کاربر می تواند داده مورد نظر خود را تایپ کند و در پایان دکمه Enter را بزند که در این صورت، داده به صورت یک رشته در حافظه ذخیره می شود. اگر رشته دریافتی را با دستور انتساب در یک متغیر رشته ای ذخیره کنیم، داده وارد شده، در برنامه قابل دسترسی خواهد بود.

برای مثال می خواهیم نام و نام خانوادگی یک شخص را از کاربر سؤال کرده و در برنامه استفاده کنیم. برای این منظور ابتدا دو متغیر رشته ای به نام name و family از نوع رشته ای اعلان می کنیم و سپس از متد ReadLine() برای دریافت نام و نام خانوادگی به صورت زیر استفاده می نماییم:

```
string name, family;
name = System.Console.ReadLine();
family = System.Console.ReadLine();
```

٪ متد ReadLine() شبیه متد ReadKey() است با این تفاوت که متد ReadKey() فقط منتظر دریافت یک کلید می شود اما در متد ReadLine() تا هنگامی که کلید Enter زده نشده است کامپیوتر منتظر می ماند.

توجه داشته باشید وقتی کامپیوتر منتظر دریافت داده است کاربر باید بداند که چه داده ای را لازم است وارد کند (نام، نمره، سن) بنابراین لازم است قبل از استفاده از متد ReadLine() یک دستور برای نمایش یک پیام و توضیحی کوتاه در مورد اینکه کامپیوتر منتظر دریافت چه داده ای است در برنامه نوشته شود. از متد Write() بدین منظور استفاده می کنیم.

مثلاً برای دریافت نام کاربر دستورات زیر را می نویسیم:

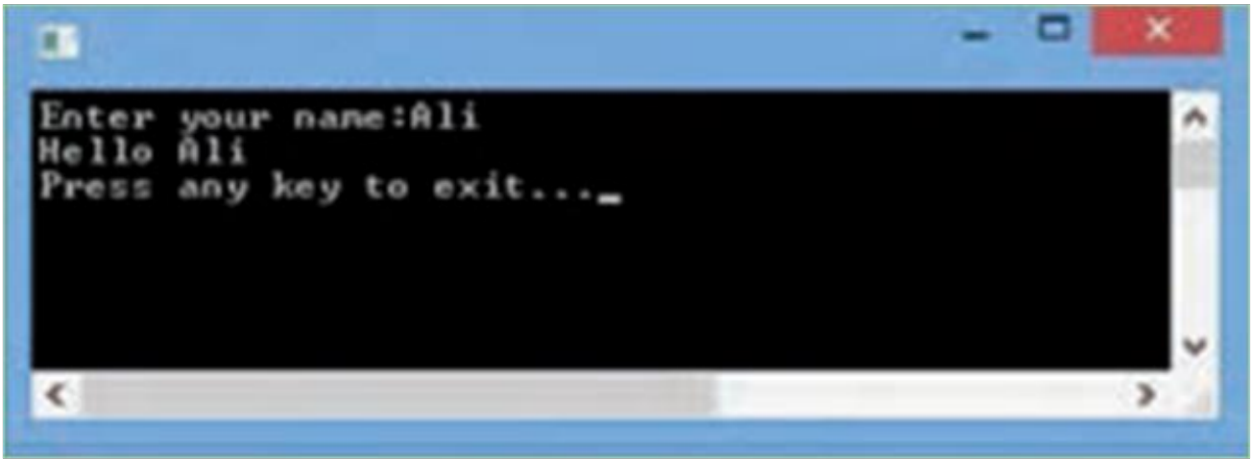
```
string name;
System.Console.Write("Enter your name:");
name = System.Console.ReadLine();
```

مثال- نام کاربر از ورودی دریافت شده و خطاب به او پیام خوشامدگویی اعلام شود.

```
using System;
class HelloYourName
{
static void Main()
{
string name;
Console.Write("Enter your name:" );
name = Console.ReadLine();
```

```
Console.WriteLine("Hello "+name);
Console.Write("Press any key to exit...");
Console.ReadKey();
}
}
```

اگر فرض کنید که کاربر، نام Ali را وارد کند، خروجی برنامه به صورت شکل زیر خواهد بود:



مثال- می خواهیم به برنامه بالا، دستوراتی اضافه کنیم که علاوه بر دریافت نام کاربر، نام خانوادگی وی نیز سؤال شود و سپس نام ونام خانوادگی را در یک خط نمایش دهد.

در برنامه بالا، کافی است یک متغیر رشته ای به نام family تعریف کرده و از متد ReadLine() برای دریافت نام خانوادگی استفاده کنیم. برای نمایش نام و نام خانوادگی در یک خط نیز، از علامت + برای الحاق رشته ها استفاده می کنیم (کدهای برجسته، تغییرات جدید هستند).

```
using System;
class HelloYourName
{
static void Main()
{
string name, family;
```

```
Console.Write("Enter your name:");
name = Console.ReadLine();
Console.Write("Enter your family:" );
family = Console.ReadLine() ;
Console.WriteLine("Hello "+ name +" "+ family);
Console.Write("Press any key to exit...");
Console.ReadKey();
}
}
```

مثال- می خواهیم برنامه ای بنویسیم که دو عدد دلخواه از کاربر دریافت کند و مجموع آن ها را حساب کرده و روی صفحه نمایش، نشان دهد.

برای دریافت داده ها از کاربر، از متد ReadLine() مانند مثال های قبلی استفاده می کنیم.

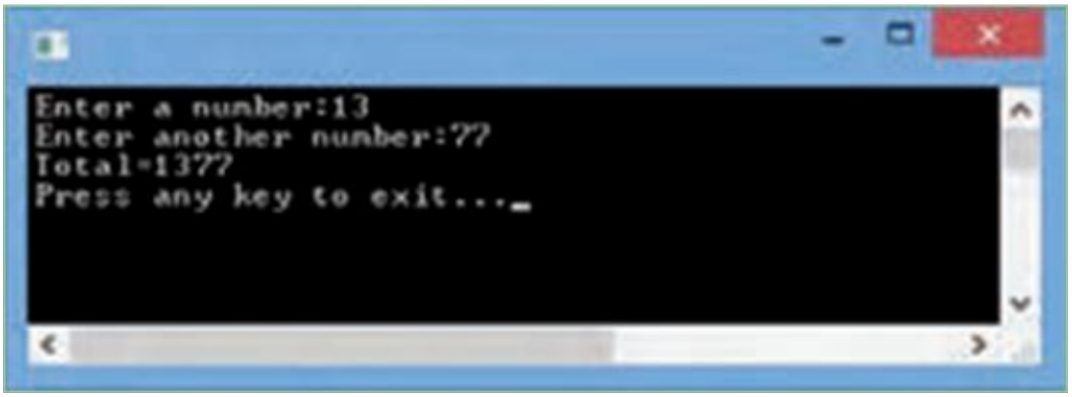
داده های دریافتی به وسیلهٔ این متد، در قالب رشته در حافظه ذخیره می شوند، بنابراین برای دسترسی به آن ها باید از متغیرهای رشته ای استفاده کنیم.

```
using System;
class GetNumbers
{
static void Main()
{
string firstNumber, secondNumber;
Console.Write("Enter a number:" );
firstNumber = Console.ReadLine();
Console.Write("Enter another number:" );
```

```
secondNumber = Console.ReadLine();
Console.WriteLine("Total="+ (firstNumber + secondNumber) );
Console.Write("Press any key to exit...");
Console.ReadKey();
}
}
```

با اجرای این برنامه، پنجره ای ظاهر می شود که از کاربر خواسته می شود که یک عدد وارد کند.

پس از وارد کردن یک عدد و زدن دکمه Enter، عدد دیگری خواسته می شود. فرض کنید اعداد ۱۳ و ۷۷ توسط کاربر وارد شود.



همان طور که بیان شد متد ReadLine() داده دریافتی را به صورت یک رشته در حافظه ذخیره می کند و در برنامه بالا از متغیرهای رشته ای firstNumber و secondNumber برای دسترسی به داده های ورودی استفاده کردیم. بنابراین علامت + در دستور زیرعمل الحاق دو رشته مثلاً "۱۳" و "۷۷" را انجام می دهد و طبیعی است که نباید انتظار عمل جمع ریاضی داشته باشیم.

دریافت اعداد

با توجه به این که داده های دریافتی به وسیله متد ReadLine() همواره به صورت رشته تحویل داده می شود باید به وسیله دستوری، رشته دریافتی را به عدد تبدیل کنیم. بنابراین به متدی نیاز داریم که بتواند یک رشته شامل ارقام را به ارزش عددی تبدیل کند تا بتوانیم روی آنها محاسبات ریاضی انجام دهیم.

خوشبختانه برای انواع داده های عددی، متدی به نام Parse() از قبل تعریف شده است که می تواند از یک رشته شامل ارقام، معادل عددی آن را بدست آورد. مثلاً برای تجزیه رشته " ۲۵۹ " به ارزش عددی، با توجه به این که درون رشته، یک عدد صحیح قرار دارد، از متد Parse() مربوط به نوع داده int استفاده می کنیم:

```
int.Parse("۲۵۹");
```

٪ به عمل بررسی کاراکتر به کاراکتر یک رشته، برای جدا کردن و بدست آوردن یک مقدار با معنی، تجزیه کردن می گویند.

حاصل اجرای این متد، عدد ۲۵۹ است که باید در یک متغیر نوع صحیح ذخیره شود. بنابراین استفاده مفید از این متد به صورت زیر خواهد بود:

```
int a;
a = int.Parse(" ۲۵۹ ");
```

اگر رشته ای حاوی عدد اعشاری باشد باید از متد Parse() مربوط به نوع داده اعشاری مثلاً float یا double استفاده کنید. مثلاً برای تبدیل رشته " ۲.۵۰ " به عدد ۲.۵ از دستورات زیر استفاده می کنیم:

```
float b;
b = float.Parse("۲.۵۰");
```

با استفاده از متد Parse() می توانیم رشته دریافتی که به وسیلهٔ متد ReadLine() از کاربر گرفته شده است را به عدد تبدیل کنیم به شرط اینکه حاوی اعداد باشد.

```
string input;
float number;
input = Console.ReadLine();
```

```
number = float.Parse(input);
```

همچنین می توانید متد ReadLine() را مستقیماً در متد Parse() استفاده کنید که در این صورت نیازی به متغیر رشته ای نیست:

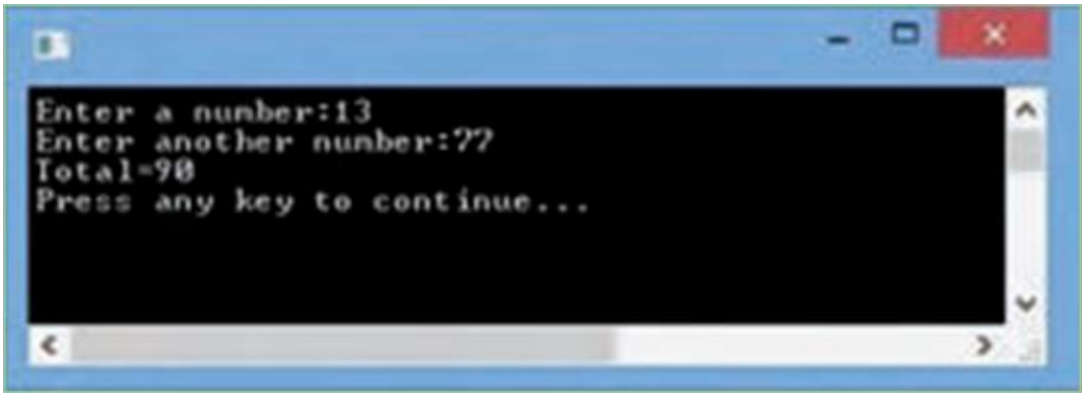
```
float number;
number = float.Parse(Console.ReadLine());
```

مثال- با تکمیل برنامه بالا مجموع دو عدد دریافتی را چاپ نمایید.

```
using System;
class GetNumbers
{
static void Main()
{
string input;
float firstNumber, secondNumber;
Console.Write("Enter a number:");
input = Console.ReadLine();
firstNumber = float.Parse(input);
Console.Write("Enter another number:");
input = Console.ReadLine();
secondNumber = float.Parse(input);
Console.WriteLine("Total=" + (firstNumber + secondNumber));
Console.WriteLine("Press any key to continue...");
Console.ReadKey();
}
```

}

نتیجه اجرای برنامه در شکل زیر نشان داده شده است:



گام چهارم

عبارت های محاسباتی

عبارت چیست؟

در درس ریاضی با عبارت های محاسباتی مختلفی مانند عبارت های زیر آشنا شدید.

۸ + ۳ * ۵

۹ – ۷.۲۵

۵.۶* ۳ + ۵.۶

در این عبارت ها، علامت + نشانهٔ عمل جمع و علامت * نشانهٔ عمل ضرب است. به این علامت ها که بیانگر انجام یک عمل بر روی اعداد و داده ها هستند، عملگر گفته می شود. مثلاً عملگر * در عبارت ۸ + ۳ * ۵ بر روی اعداد ۳ و ۵ عمل ضرب را انجام می دهد و همچنین عملگر + در عبارت بالا بر روی عدد ۸ و نتیجه حاصلضرب یعنی عدد ۱۵، عمل جمع را انجام می دهد. به اعدادی که یک عملگر بر روی آنها عملی را انجام

می دهد عملوند می گویند. اعداد ۳ و۵ عملوندهای عملگر ضرب و عدد ۸ و ۱۵ عملوندهای عملگر جمع هستند.

هر یک از عملگرهای ضرب و جمع بر روی دو عدد عمل می کنند و به عبارتی دارای دو عملوند هستند به این عملگرها، عملگرهای دوتایی گفته می شود. عملگر - در عبارت ۹-۷.۲۵ عملگر تفریق است که آن نیز یک عملگر دوتایی است و حاصل تفریق ۷.۲۵ از ۹ را محاسبه می کند. اما عملگر قرینه – در عبارت –x، فقط دارای یک عملوند x است و آن را قرینه می کند. این عملگر یک عملگر یکتایی است.

٪ یک عبارت از تعدادی عملگر و عملوند تشکیل شده است و دارای یک حاصل یا نتیجه می باشد. نتیجه یا حاصل یک عبارت ممکن است عددی یا غیرعددی باشد.

عملگرهای ریاضی یا حسابی

اگر در عبارتی بیش از یک عملگر وجود داشته باشد ابتدا عملگری عمل خود را انجام می دهد که اولویت بالاتری نسبت به دیگری داشته باشد. مثلاً اولویت عملگر ضرب بیش از اولویت عملگرجمع است. چنانچه دو یا چند عملگر دوتایی، با اولویت یکسان در یک عبارت وجود داشته باشد ابتدا عملگر سمت چپ انجام می شود. به عبارت دیگر از سمت چپ به راست، عملگرها به ترتیب انجام می شوند که به آن « شرکت پذیری » می گویند.

در جدول زیر لیست عملگرهای ریاضی را به ترتیب اولویت مشاهده می کنید. عملگر قرینه اولویت بالاتری نسبت به بقیه عملگرهای ریاضی دارد و عملگرهای جمع و تفریق دارای اولویت یکسان ولی کمترین اولویت را در بین عملگرهای ریاضی دارند.

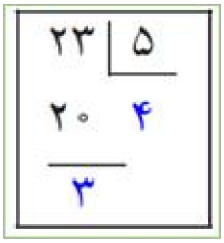
اولویت	نام عملگر	نشانه	مثال	نوع عملگر
۱	قرینه	-	-5	یکتایی
۲	ضرب تقسیم باقیمانده تقسیم	* / ٪	12 * 36 25 / 4 23%5	دوتایی
۳	جمع تفریق	+ -	75 + 14 29 - 36	دو تایی

عملکرد عملگرهای جمع و تفریق و ضرب مانندعملکرد آنها در ریاضیات است اما عملگر تقسیم با توجه به نوع عملوندهایش می تواند تقسیم صحیح و بدون ممیز و یا تقسیم اعشاری و ممیزی انجام دهد. مثلاً در عبارت ۹/۲ چون عملوندها اعداد صحیح هستند بنابراین تقسیم بدون ممیز و صحیح انجام خواهد شد که نتیجه آن عدد ۴ است. اما در عبارت۹.۰/۲ یا در عبارت۹/۲.۰ ، چون حداقل یکی از عملوندها، اعشاری است بنابراین تقسیم به صورت اعشاری انجام می شود که حاصل عبارت عدد ۵/۴ است.

در جدول بالا، عملگر جدیدی نیز به نام باقیمانده تقسیم که با نشانهٔ ٪ مشخص می شود، مشاهده می کنید. به وسیلهٔ این عملگر می توانیم باقیماندهٔ تقسیم یک عدد بر عدد دیگر را با توجه به خارج قسمت صحیح و بدون اعشار به دست آوریم. مثلاً در تقسیم عدد ۲۳ بر عدد ۵، خارج قسمت بدون اعشار عدد ۴ است بنابراین باقیمانده عدد ۳ است.

۲۳ / ۵ = ۴

۲۳ ٪ ۵ = ۳



برای تغییر دادن اولویت عملگرها، از علامت های پرانتز استفاده می شود. مثلاً در عبارت زیر ابتدا عمل جمع و سپس عمل ضرب انجام می شود.

۵.۶ *(۳ + ۶.۵)

اگر چند پرانتز تو در تو نیز وجود داشته باشد، ابتدا داخلی ترین پرانتز انجام می شود.

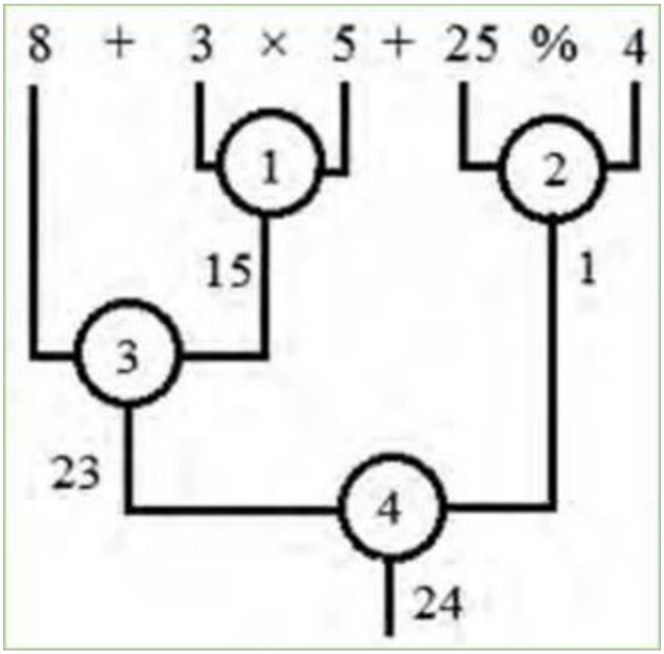
در جدول زیر، چند نمونه از عبارت های ریاضی نشان داده شده است.

عبارت	حاصل عبارت	نوع عبارت
175 / 31	5	صحیح
175 % 31	20	صحیح
7.5 / 2	3.75	اعشاری double
7.5 % 2	1.5	اعشاری double
36 / 2.0	18.0	اعشاری double
36 % 2	0	صحیح
254138 / 10	254138	صحیح
2451389 % 10	9	صحیح

عبارت محاسباتی زیر را در نظر می گیریم و سپس نوع آن را تعیین می کنیم.

۸ + ۳ * ۵ + ۲۵ ٪ ۴

در این عبارت بیش از یک عملگر وجود دارد، بنابراین ابتدا عملگری که دارای اولویت بالاتر است، انجام می شود. چون اولویت عملگرهای * و ٪ بالاتر از عملگر + است بنابراین ابتدا این دو عملگر انجام می شود و از طرفی چون این دو عملگر دارای اولویت یکسان هستند، برطبق شرکت پذیریِ چپ، ابتدا عملگر سمت چپ یعنی * و سپس عملگر ٪ انجام می شود. بر همین اساس در مورد عملگرهای جمع نیز ابتدا عملگرسمت چپ و سپس عملگر + بعدی انجام می شود.



در برنامه ها، معمولاً حاصل یا نتیجهٔ یک عبارت را در یک متغیر نگهداری می کنند. البته نوع متغیری که حاصل یک عبارت، در آن قرار می گیرد باید با نوع عبارت، سازگار باشد. مانند ظرفی در آشپزخانه که بخواهیم در آن غذا یا نوشیدنی بریزیم باید گنجایش مناسب آن غذا را داشته باشد.

قوانین زیر، باید به وسیلهٔ برنامه نویس، در هنگام انتساب یک عبارت به یک متغیر، رعایت شود، در غیر این صورت با پیام خطای مترجم مواجه می شویم. مترجم زبان C# روی این قوانین سخت گیر است زیرا می خواهد از اشتباهات برنامه نویسان جلوگیری نماید، این یکی از ویژگی های زبان C# است.

۱- اگر حاصل یک عبارت عدد صحیح باشد بسته به اندازه و بزرگی عدد، می تواند در یک متغیرِ نوعِ صحیح که گنجایش آن مساوی یا بزرگ تر از حاصل عبارت باشد جای گیرد.

مثلاً حاصل عبارت ۱۷۵ / ۳۱ عدد ۵ است این عدد کوچک می تواند در تمام متغیرهای نوع صحیح زیر قرار گیرد.

sbyte , byte , short , ushort , int , uint , long , ulong

ولی در عبارت ۱۰/۲۵۴۱۳۸۹ چون حاصل عبارت عدد ۲۵۴۱۳۸ است که عدد صحیح بزرگی است فقط در متغیرهای نوع int , uint , long , ulong قابل نگهداری است.

۲- اگر حاصل یک عبارت از نوع صحیح باشد می تواند در یک متغیر نوع اعشاری نیز ذخیره شود، اما با این تفاوت که اعداد بزرگ (Long) فقط با ۷ رقم دقت (در نوع float) و یا با ۱۵ رقم دقت (در نوع double) ذخیره می شود و بقیه ارقام عدد، گرد می شود.

مثلاً در دستور زیر حاصل عبارت ۱۰/۱۲۳۴۵۶۷۸۹۰ در متغیر اعشاری ذخیره می شود ولی به دلیل اینکه عدد بزرگی است به صورت گرد شده در متغیر ذخیره می شود.

```
float number = ۱۲۳۴۵۶۷۸۹۰ / ۱۰;
Console.WriteLine("Number =" + number);
```

همان طور که در شکل زیر مشاهده می کنید مقداری که در متغیر number وجود دارد به صورت نماد علمی نشان داده می شود که اگر آن را به صورت معمولی تبدیل کنیم خواهیم داشت:

$$1.234568E+08 \longrightarrow 1.234568 \times 10^{8} = 123456800$$

اگر این مقدار را با حاصل عبارت اولیه مقایسه کنیم خواهیم دید که عدد با ۷ رقم گرد شده است:

۱۲۳۴۵۶۷۸۹ حاصل عبارت

۱۲۳۴۵۶۸۰۰ مقدار ذخیره شده در متغیر

۳- اگر حاصل یک عبارت از نوع اعشاری باشد نمی تواند به طور ضمنی، در یک متغیر نوع صحیح جای داده شود، فقط می تواند در یک متغیر اعشاری (نوع float و یا double) جای گیرد.

۴- اگر حاصل یک عبارت از نوع اعشاری double باشد، فقط در متغیر نوع double می تواند جای می گیرد. چرا؟

حاصل تقسیم یک عدد اعشاری بر یک عدد صحیح، عددی اعشاری است و مترجم آن را از نوع double در نظر می گیرد. تقسیم زیر را در نظر بگیرید:

۲۱۹.۵ / ۱۴

با توجه به اینکه عدد ۲۱۹.۵ اعشاری است، بنابراین تقسیم نوع اعشاری انجام می شود و نتیجه عبارت از نوع double خواهد بود.

برای نگهداری نتیجهٔ این عبارت، باید ابتدا متغیر مناسبی را تعریف کنیم و سپس با دستور انتساب حاصل عبارت را در آن ذخیره کنیم. با توجه به اینکه حاصل عبارت از نوع double است بنابراین باید متغیر نیز اعشاری و از نوع double باشد.

دستور زیر متغیری برای ذخیرهٔ معدل دانش آموز تعریف می کند تا نتیجهٔ تقسیم را در آن ذخیره نماید.

```
double meanScore;
```

اکنون می توانید مقدار عبارت را در متغیر مزبور با استفاده از دستور انتساب مقداردهی کنید.

```
double meanScore = ۲۱۹.۵ / ۱۴;
```

جایگزینی دو دستور بالا با یک دستور به صورت زیر خواهد بود:

```
double meanScore = ۲۱۹.۵ / ۱۴;
```

با اجرای دستور قبل عمل تقسیم انجام شده و حاصل تقسیم یعنی عدد ۱۵.۶۷۸۶۷۱۴۲۸۵۷۱۴ در متغیر meanScore ذخیره می شود.

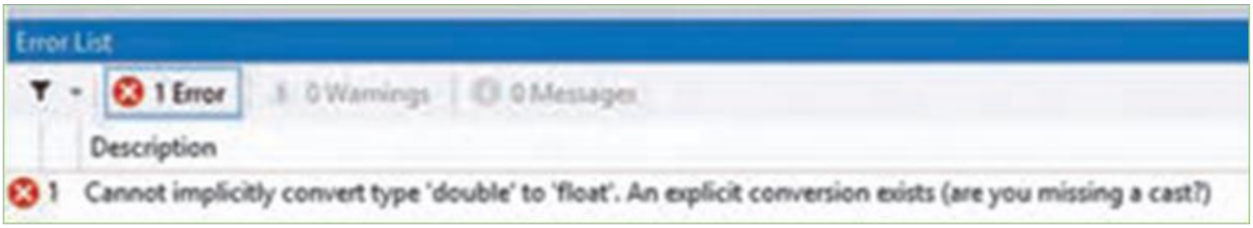
با توجه به این که معمولاً معدل نمرات با دو رقم اعشار بیان می شود، بنابراین در این موارد بهتر است از متغیر نوع float استفاده کنیم. در این صورت لازم است نوع عبارت محاسبه معدل نیز float باشد. برای این که نوع عبارت float شود می توانید از مترجم بخواهید عدد ۲۱۹.۵ را یک عدد نوع float در نظر بگیرد بدین منظورحرف f یا F را بعد از عدد می نویسیم ۲۱۹.۵F

بنابراین دستور زیر را می نویسیم:

```
float meanScore = ۲۱۹. ۵f / ۱۴;
```

با اجرای دستور بالا، عدد ۱۵/۶۷۸۵۷ در متغیر meanScore ذخیره می شود.

اگر فراموش کنید که در دستور بالا، حرف f را بنویسید مترجم خطا می دهد. چون در سمت راست علامت انتساب، یک عبارت از نوع داده double است ولی در سمت چپ یک متغیر از نوع float است که ظرفیت کمتری نسبت به double دارد. در چنین حالتی در محیط VS یک خط قرمز زیر عبارت کشیده می شود و مترجم خطایی را صادر می کند (شکل زیر).



شکل بالا بیان می کند که:

« مترجم نمی تواند نوع دادهdouble را به طور ضمنی و خودکار به نوع float تبدیل کند.» باید به طور صریح و واضح از مترجم بخواهید عمل تبدیل نوع را انجام دهد.

٪ اگر در برنامه ای با عبارت ها و اعداد اعشاری با دقت حداکثر ۷ رقم سر و کار دارید و می خواهید از متغیرهای نوع float استفاده کنید، باید پس از هر عدد اعشاری، یک حرف f یا F قرار دهید تا مترجم آن عدد را به عنوان نوع float در نظر بگیرد. اما اگر از متغیرهای نوع double استفاده می کنید دیگر نیازی به نوشتن حرف f نیست.

float meanMark = ۲۱۹ .۵f /۱۴;

مثال- مجموع و معدل نمرات درسی را طبق جدول زیر محاسبه کنید.

نام درس	نمره درس	تعداد واحد
فیزیک	۱۷/۵	۳
شیمی	۱۹	۲
ریاضی	۱٤/۵	٤
ادبیات	۱۸	۲
ورزش	۱۹	۱

به دلیل این که می خواهیم روی داده ها پردازش انجام دهیم بهتر است ابتدا نمرات را داخل متغیرهای مناسب ذخیره کنیم:

```
float physicMark = ۱۷.۵F, chemistryMark=۱۹, mathMark=۱۴.۵F;
float litracyMark=۱۸, PEMark=۱۹;
```

به همین صورت تعداد واحدها را نیز داخل متغیرهای مناسب قرار می دهیم:

```
int physicCredit=۳, chemistryCredit=۲, mathCredit=۴;
int literacyCredit=۲, PECredit=۱;
```

حال می خواهیم مجموع نمرات را به دست آوریم. با توجه به اینکه تعداد واحد هر درس مختلف است نمی توانیم به سادگی نمرات را با یکدیگر جمع کنیم بلکه باید هر نمره را در تعداد واحد درسی مربوطه ضرب کنیم و سپس حاصل ضرب های به دست آمده را با یکدیگر جمع کنیم و نتیجه را در یک متغیر قرار دهیم. بنابراین متغیری به نام totalMark برای این منظور تعریف و حاصل عبارت محاسباتی مربوطه را در آن قرار می دهیم:

```
; float totalMark;
 physicCredit)*+ totalMark= (physicMark
 chemistryCredit)*+( chemistryMark
 mathCredit)*+ (mathMark
 literacyCredit)*+ (literacyMark
 PECredit);*(PEMark
```

پس از به دست آوردن مجموع نمرات که در متغیر totalMark، قرار گرفته است، مجموع واحدها را نیز حساب می کنیم و در یک متغیر به نام totalCredit قرار می دهیم.

```
int totalCredit = physicCredit + chemistryCredit + mathCredit + literacyCredit +
PECredit;
```

در این لحظه می توانیم با تقسیم مجموع نمرات بر مجموع واحدها، معدل را به دست آوریم و در یک متغیر ذخیره کنیم.

```
float average = totalMark / totalCredit;
```

برای نمایش مقادیر متغیرها از دستورات زیر استفاده می کنیم:

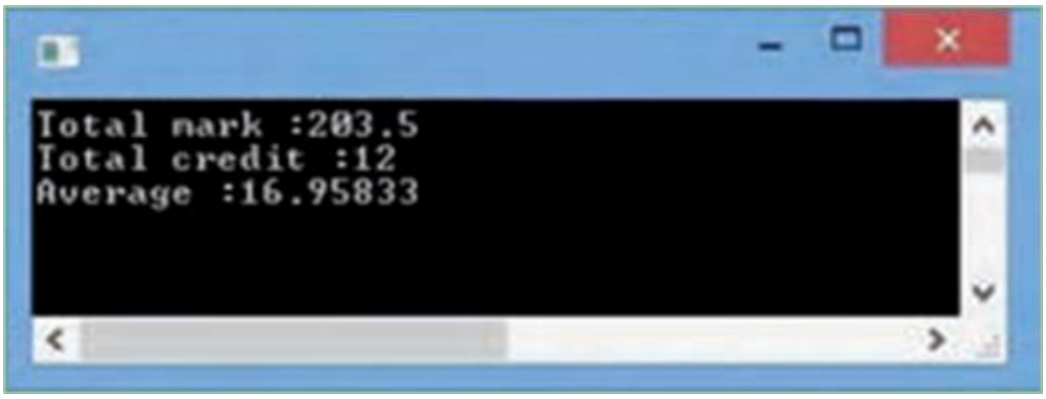
```
Console.WriteLine("Total mark:" + totalMark);
Console.WriteLine("Total credit:" + totalCredit);
Console.WriteLine("Average:" + average);
```

با توجه به توضیحات بالا برنامه مورد نظر چنین خواهد بود:

```
using System;
class Average
{
static void Main()
{
float physicMark = ۱۷.۵F, chemistryMark = ۱۹, mathMark = ۱۴.۵F;
float litracyMark = ۱۸, PEMark = ۱۹;
int physicCredit = ۳, chemistryCredit = ۲, mathCredit = ۴;
int literacyCredit = ۲, PECredit = ۱;
float totalMark;
 physicCredit)*+ totalMark = (physicMark
 chemistryCredit)*+ (chemistryMark
 mathCredit)*+ (mathMark
 literacyCredit)*+ (litracyMark
 PECredit);*(PEMark
```

```
int totalCredit = physicCredit + chemistryCredit + mathCredit + literacyCredit
+ PECredit;
float average = totalMark / totalCredit;
Console.WriteLine("Total mark:" + totalMark);
Console.WriteLine("Total credit:" + totalCredit);
Console.WriteLine("Average:" + average);
Console.ReadKey();
}
}
```

خروجی برنامه چنین است:



عملگرهای افزایشی و کاهشی

در زبان برنامه نویسی C#، عملگرهای یکتایی ++ و-- به ترتیب برای افزایش و کاهش مقدار یک متغیر به اندازهٔ یک واحد در نظر گرفته شده است. البته این عملگرها از زبان C وارد این زبان شده اند و برای کوتاه شدن و تایپ کمتر دستورات، ابداع شده اند که در زبان های دیگر امروزی مانند جاوا نیز وجود دارند (جدول زیر).

نام عملگر	نوع عملگر	نشانهٔ	مثال
افزایش	یکتایی	++	++ value یا value ++
کاهش	یکتایی	--	-- value یا value --

اکنون می توانیم اضافه کردن یک نمره به درس ریاضی را با استفاده از عملگر افزایشی انجام دهیم:

```
// ++mathMark ; عملگر افزایشی قبل از نام متغیر قرار گرفته است
```

یا

```
// mathMark++ ; عملگر افزایشی بعد از نام متغیر قرار گرفته است
```

عملگر افزایشی یا کاهشی را می توانید قبل از نام متغیر و یا بعد از نام متغیر ذکر کنید که درهر دو حالت سبب می شود مقدار متغیر به اندازهٔ یک واحد تغییر کند.

عملگرهای انتساب

در جدول زیر عملگرهای انتساب دیده می شود.

نوع عملگر	نشانهٔ
انتساب	= , += , -= , *= , /= , %= &= , |= , ^= , <<= , >>=

با عملگر = و کاربرد آن در گام قبل آشنا شدید که برای ذخیره کردن یک مقدار در یک متغیر استفاده می شود. دستورات زیر را در نظر بگیرید.

```
int x , y;
x = ۶;
```

string helloString = "Hello World";

y = x;

در دستور آخر، مقداری که در متغیر x قرار دارد (عدد ۶) به متغیرy منتسب می شود و y نیزبرابر ۶ می شود.

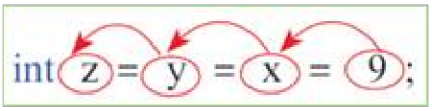
٪ حاصل عبارتی که دارای عملگر انتساب است، مقدار داده یا متغیری است که در سمت راست عملگر واقع شده است.

مثلاً در دستور زیر

int x , y;

int z= y = x = ۹;

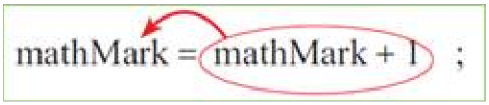
چند عملگر انتساب وجود دارد و از سمت راست به ترتیب انجام می شوند. ابتدا عدد ۹ در متغیر x قرار می گیرد و سپس حاصل عبارت که عدد ۹ است در متغیر y کپی شده و سپس این مقدار در متغیر z کپی می شود. یعنی در عبارتی که عملگرهای انتساب وجود دارد این عملگرها از سمت راست به چپ انجام می شوند (شرکت پذیر راست) بر خلاف عملگرهای ریاضی مشابه که از چپ به راست انجام می شوند.



اکنون اگر بخواهیم یک نمره، به نمره درس ریاضی اضافه کنیم، با توجه به اینکه نمرهٔ هر درس را در یک متغیر ذخیره کردیم، دستور لازم برای افزایش یک واحد به متغیر مربوطه را بنویسید.

mathMark+=۱

متغیر mathMark حاوی نمره درس ریاضی است. برای افزایش یک نمره از دستور انتساب زیر استفاده می کنیم:



فرض کنید بخواهید محتوای متغیر x را سه برابر کنید در این صورت از دستور زیر استفاده می کنید:

x *= ۳;

مثال- با اجرای دستورات زیر چه عددی بر روی خروجی نشان داده می شود؟

```
int x = ۳;
int y = ۴;
x *= y;
Console.WriteLine(x);
```

در دستور x*=y; حاصل ضرب متغیر x در متغیر y محاسبه می شود (برابر ۱۲) و در متغیر x قرار می گیرد. بنابراین عدد ۱۲ در روی صفحه نمایش داده می شود.

گام پنجم

دستورهای شرطی

عبارت منطقی یا بولین

در گام سوم در قسمت معرفی انواع داده ها با نوع داده منطقی، آشنا شدید. این نوع داده، فقط دارای دو مقدار true (درست) یا false (نادرست) است. عبارت منطقی نیز عبارتی است که حاصل آن فقط یکی از دو مقدار true یا false است.

مثلاً عبارت ۱۰ > ۱۲ یک عبارت منطقی است، که نتیجهٔ آن false است چون ۱۲ کوچک تر از ۱۰ نیست. ولی در عبارت X > ۰ به شرط اینکه X عدد مثبتی باشد نتیجه true است.

در دو مثال بالا از عملگرهای مقایسه ای < یا > استفاده شده است، عملگرهای مقایسه ای دیگری نیز وجود دارند که در جدول زیر، مشاهده می شود. این عملگرها شبیه عملگرهایی است که در ریاضیات استفاده می شود. معمولاً درعبارت های منطقی از عملگرهای مقایسه ای استفاده می شود.

نام عملگر	عملگر در زبان C#	نشانه عملگر در ریاضی	مثال
برابری	==	=	delta == 0
نامساوی	!=	≠	name != "AMIN"
کوچک تر	<	<	max < number
کوچک تر یا مساوی	<=	≤	x <= a
بزرگ تر	>	>	temperature > 15
بزرگ تر یا مساوی	>=	≥	(a+b) >= c

عملگرهای مقایسه ای

همان طور که از نام این عملگرها مشخص است، برای مقایسهٔ داده ها استفاده می شوند و نتیجهٔ آنها یکی از دو مقدار true یا false است. عملگرهای مقایسه ای می توانند دو عدد صحیح یا اعشاری و یا دو داده کاراکتری و یا رشته ای را با یکدیگر مقایسه کنند.

علامت بعضی از عملگرهای مقایسه ای در زبان C#، با علامت ریاضی آنها کمی متفاوت است مثلاً برای بررسی مساوی یا برابر بودن دو مقدار، از علامت = = استفاده می شود (دو بار علامت =) و یا برای بررسی مخالف یا نابرابر بودن دو مقدار، از علامت=! استفاده می شود.

در جدول بالا هر یک از عملگرهای مقایسه ای به همراه علامت ریاضی آنها نشان داده شده است.

مثال- در برنامه زیر حاصل چند عبارت منطقی بر روی صفحه نشان داده می شود:

```
using System;
class Expression
{
static void Main()
{
int weight = ۷۰۰;
Console.WriteLine(weight >= ۵۰۰); // True
```

```
char gender = 'm';
Console.WriteLine(gender == 'f' ); // False
double colorWaveLength = ۱.۶۳۰;
Console.WriteLine(colorWaveLength> ۱.۶۲۱); // True
Console.WriteLine('B' == 'A' + ۱); // True
Console.ReadKey();
}
}
```

٪ حاصل عبارت های منطقی را می توانید بر روی صفحه خروجی نمایش دهید و یا در داخل متغیرهایی از نوع bool ذخیره کنید.

دستور شرطی if

یکی از کاربردی ترین دستورات زبان برنامه نویسی، دستور شر طی if است. در مقابل دستور if یک عبارت شرطی قرار داده می شود، که در صورت درست بودن شرط، دستور یا دستورات if اجرا می شوند. در زبان برنامه نویسی C# از دستور if (با حروف کوچک نوشته می شود) برای کنترل اجرای دستورها و بررسی شرط، استفاده می شود.

ساختار دستور if به صورت زیر است:

```
if (عبارت شرطی)
دستور;
```

دستور شرطی if از سه بخش تشکیل شده است:کلمه رزرو شدهٔ if، عبارت منطقی داخل پرانتز و دستوری که در صورت درست (true) بودنِ نتیجه عبارت، اجرا خواهد شد.

توجه داشته باشید که پس از کلمهٔ if، یک جفت پرانتز وجود دارد و عبارتی از نوع منطقی که برای بررسی و مقایسهٔ داده است، داخل پرانتز نوشته می شود. در خط بعدی، دستوری که می خواهیم در صورتِ درست بودنِ عبارت منطقی اجرا شود، با تورفتگی می نویسیم و در انتهای آن علامت ; را به منزله پایان دستور if قرار می دهیم.

در دستورات زیر نمونه ای از به کار گیری دستور if را می بینید.

```
if (mark < ۱۰)
Console.WriteLine("Failed");
```

٪ به علامت نقطه ویرگول در دستور if توجه کنید. بعد از علامت پرانتز علامت نقطه ویرگول نگذارید، زیرا دستور if هنوز تمام نشده است. بلکه علامت نقطه ویرگول باید در انتهای دستور نوشته شود.

وقتی که کامپیوتر، در حال اجرای برنامه است با رسیدن به دستورif، ابتدا مقدار عبارت را محاسبه می کند. در صورتی که ارزش عبارت true باشد، دستوری که بعد از if قرار دارد، اجرا می شود و در صورتی که مقدار عبارت false باشد، دستور مربوطه اجرا نمی شود.

مثال- می خواهیم برنامه ای بنویسیم که رمز ورود را از کاربر دریافت کند و در صورت صحیح بودن پیام مناسب چاپ نماید(رمز صحیح admin۱۲۳).

```
class Login
{
static void Main()
{
string password;
Console.Write("Enter password:");
password = Console.ReadLine();
if (password == "admin۱۲۳")
Console.WriteLine("Welcome");
```

```
Console.WriteLine("Press any key to continue...");
Console.ReadKey();
}
}
```

مثال- می خواهیم برنامه ای بنویسیم که عدد بزرگ تر را از بین دو عدد دریافتیِ کاربر، پیدا کند.

```
class FindMaximum
{
static void Main()
{
string input;
int firstNumber , secondNumber;
Console.Write("Enter a number:");
input = Console.ReadLine();
firstNumber = int.Parse(input);
Console.Write("Enter another number:");
input = Console.ReadLine();
secondNumber = int.Parse(input);
int biggerNumber = firstNumber;
if (secondNumber>biggerNumber)
biggerNumber = secondNumber;
Console.WriteLine("The maximum number is:" + biggerNumber);
Console.WriteLine("Press any key to continue...");
Console.ReadKey();
```

```
}
}
```

توجه داشته باشید که در نوشتن دستورهای بالا، همه دستورها به جز خط بعد از دستور if، زیر هم نوشته می شوند.

مثال- عددی صحیح، داخل متغیری به نام number قرار دارد و می خواهیم زوج بودن آن را پس از تشخیص، با پیامی مناسب اعلام کنیم.

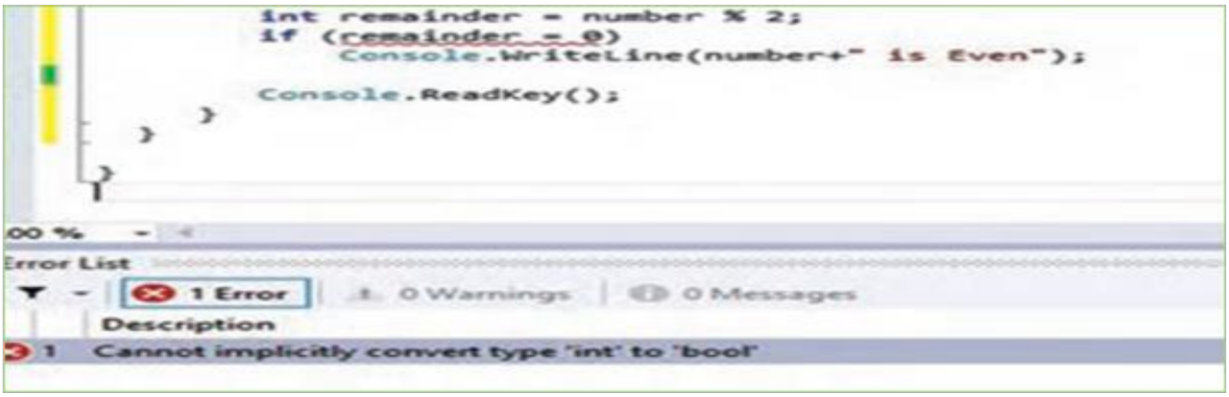
در این مثال، از روش بخش پذیری بر ۲ استفاده می کنیم. ابتدا باقیماندهٔ تقسیم عدد بر ۲ را حساب می کنیم، اگر باقیمانده برابر صفر بود پیامی را روی صفحه نمایش می دهیم تا نشان دهد که عدد زوج است.

```
int remainder = number % ۲;
if (remainder == ۰)
Console.WriteLine(number+" is Even.");
```

اگر در متغیر number، عددی زوج مانند ۱۸ باشد، حاصل باقیماندهٔ تقسیم آن عدد بر۲، صفر خواهد شد و در این صورت نتیجهٔ عبارت منطقی در دستور if، مقدار true خواهد بود. بنابراین دستورِ نمایش پیام اجرا می شود و اعلام می کند که عدد زوج است.

اما اگر در دستورهای بالا، عددی که در متغیر number قرار دارد، عددی فرد مانند ۱۵ باشد، حاصل باقیماندهٔ تقسیم آن عدد بر ۲ عدد یک خواهد شد و در این صورت نتیجه عبارت منطقی در دستورif، مقدار false خواهد بود. بنابراین دستورِ نمایش پیام، اجرا نمی شود و چیزی روی صفحه نشان داده نمی شود.

% یکی از اشتباهات رایج برنامه نویسان در نوشتن عبارت منطقی دستور if، استفاده از علامت = به جای== است. در این صورت، مترجم متوجه بروز چنین اشتباهی می شود و علاوه بر کشیدن خط قرمز در زیر عبارت مورد نظر، خطایی را صادر می کند.



مثال- اکنون می خواهیم دستورهای مربوط به مثال بالا را طوری توسعه دهیم که اگر در متغیر number، عددی فرد وجود داشت، آن را شناسایی کرده و پیام مناسب اعلام کند.

```
int remainder = number % ۲;
if (remainder == ۰)
Console.WriteLine(number+"is Even");
if (remainder != ۰)
Console.WriteLine(number + "is Odd");
```

دستور شرطی if else

در مثال بالا، برای اینکه عدد زوج یا فرد را تشخیص دهیم از باقیمانده تقسیم عدد بر ۲ استفاده کردیم. برای مثال هایی که شبیه مثال بالا می باشد از ساختار if-else استفاده می کنیم. شکل کلی این دستور چنین است:

```
if (عبارت منطقی)
دستور شماره۱;
else
دستور شماره۲;
```

اگر نتیجه عبارت منطقیtrue باشد، دستور شماره۱ که مربوط به قسمت if است اجرا می شود و در غیر این صورت، یعنی اگر نتیجه عبارت منطقی false باشد، دستور شماره ۲ که مربوط به قسمت else است،اجرا می شود.

٪ دستور if-else را خوانا بنویسید یعنی:

دستورهای شماره ۱ و ۲ که مربوط به قسمت if یا قسمت else است را با کمی تورفتگی بنویسید تا معلوم شود که هرکدام متعلق به چه قسمتی است. مراقب باشید که بعد از کلمه رزرو شده else، علامت نقطه ویرگول قرار ندهید.

مثال- در این مثال، دستورهای تشخیص زوج یا فرد بودن عدد را با استفاده از دستور if else، می نویسیم.

```
int remainder = number ٪ ۲;
if (remainder == ۰)
Console.WriteLine(number+"is Even");
else
Console.WriteLine(number + " is Odd");
```

مثال- می خواهیم برنامه ای بنویسیم که حقوق دریافتی (خالص) یک کارمند را محاسبه نماید. هر کارمند دارای یک حقوق ثابت است که با توجه به میزان تحصیلات و تجربه کاری معین می شود. از حقوق تمام کارمندان، ۷ درصد به عنوان حق بیمه کسر می گردد. همچنین کارکنانی که حقوق آنها بیش از ده میلیون ریال باشد، مالیات به اندازه ۵ درصد کسر می گردد (مالیات به مازاد بر ده میلیون تعلق می گیرد).

```
class Salary
{
static void Main()
{
long income;
double tax, insurance, net;
```

```
string input;
Console.Write (''Enter income:'');
input = Console.ReadLine();
income = long.Parse(input);
 ۰.۰۷;*insurance=income
if (income > ۱۰۰۰۰۰۰۰)
{
 ۰.۰۵;*tax = (income - ۱۰۰۰۰۰۰۰)
Console.WriteLine(''Tax=''+ tax);
}
else
tax = ۰;
Console.WriteLine(''Insurance=''+ insurance);
net = income - insurance - tax;
Console.WriteLine(''Net=''+ net);
Console.WriteLine(''Press any key to continue...'');
Console.ReadKey();
}
}
```

٪ بلاک چیست؟

به تعدادی دستور که داخل علامت های آکولاد باز و بسته قرار داشته باشند بلاک گفته می شود.

```
{
دستور;
```

دستور;

دستور;

}

بلاک می تواند خالی باشد یعنی بین علامت های آکولاد، هیچ دستوری وجود نداشته باشد.

بلاک می تواند فقط شامل یک دستور باشد.

برای خوانا و واضح شدن یک بلاک، دستورهای داخل بلاک را با تورفتگی می نویسیم تا به راحتی در برنامه مشخص شوند.

عملگرهای منطقی

عملگر منطقی && یک عملگر دوتایی و دارای دو عملوند است. در هنگام اجرا، ابتدا نتیجه عملوند سمت چپ، به وسیلهٔ کامپیوتر محاسبه و درستی یا نادرستی آن مشخص می شود. اگر ارزش عملوند سمت چپ falseباشد، نتیجه عملگر && نیز false خواهد بود. اما اگر نتیجه عملوند سمت چپ true باشد، آنگاه عملوند سمت راست محاسبه می شود و ارزش عبارت ترکیبی به دست می آید. علاوه بر عملگر &&، عملگرهای منطقی دیگری نیز در زبان C#، برای ایجاد عبارت های مرکب وجود دارد، که علامت و عملکرد آنها به ترتیب اولویت، در جدول زیر نشان داده شده است.

اولویت	نام عملگر	علامت	عملکرد
۱	نقیض	!	ارزش عملوند را معکوس می کند.
۲	و	&&	تنها در صورتی که هر دو عملوند true، باشند، نتیجه این عملگر نیز true خواهد بود .در غیر این صورت false است.
۳	یا	||	اگر حداقل یکی از عملوندها true باشند، نتیجه این عملگر نیز true خواهد بود.
۴	یا انحصاری	^	اگر ارزش عملوندها مخالف یکدیگر باشد، نتیجه این عملگر true خواهد بود.

حالت های مختلف در عملگرهای منطقی نیز در جدول زیر آمده است:

ارزش عبارت سمت چپ	ارزش عبارت سمت راست	نتیجه عملگر &&	نتیجه عملگر ||	نتیجه عملگر ^
false	false	false	false	false
false	true	false	true	true
true	false	false	true	true
true	true	true	true	false

مثال- می خواهیم برنامه ای بنویسیم که عددی را دریافت کند و تشخیص دهد که مضرب پنج است یا خیر.

```
class CheckNumbers
{
static void Main()
{
string input;
int number;
Console.Write("Enter a number:");
input = Console.ReadLine();
number = int.Parse(input);
int firstDigit = number % ۱۰;
if ((firstDigit == ۰) || (firstDigit == ۵))
Console.WriteLine("The number is a multiple of ۵.");
Console.WriteLine("Press any key to continue...");
Console.ReadKey();
}
}
```

مثال- می خواهیم برنامه ای بنویسیم که نام کاربری و رمز ورود را از کاربر دریافت کند و در صورت صحیح بودن پیام مناسب چاپ نماید.(فرض شده است که کاربِر مجاز، دارای نام کاربری « admin » و کلمهٔ عبور «admin۱۲۳ » است).

```
class Login
{
static void Main()
{
string userName, password;
Console.Write("Enter username:");
userName = Console.ReadLine();
Console.Write("Enter password:");
password = Console.ReadLine();
if ((userName == "admin") && (password == "admin۱۲۳"))
Console.WriteLine("Welcome Admin.");
else
Console.WriteLine("Invalid username or password!.");
Console.WriteLine("Press any key to continue...");
Console.ReadKey();
}
}
```

مثال- می خواهیم برنامه ای بنویسیم که با توجه به عدد دریافتی از کاربر، نام یکی از فصل های سال را نمایش دهد)۱: بهار، ۲: تابستان، ۳: پائیز، ۴: زمستان).

برای نوشتن شرط های این برنامه، نیاز به مقایسه هایی داریم که شرط برنامه را پیچیده و به هم مرتبط می کنند.

ساختار if_else پیچیده

در این ساختار هرگاه یکی از عبارت های منطقی if درست باشد، دستور مربوط به همان if اجرا شده و دیگر سایر شرط ها بررسی نمی شود و در نتیجه ساختار سریع تر اجرا می شود. زیرا نیازی ندارد به بررسی شرط های اضافی بپردازد که قطعاً نادرست هستند. در ساختار زیر در صورتی که عبارت منطقی۱ درست (true) نباشد، عبارت منطقی ۲ بررسی می شود و در صورتی که هیچ یک از عبارات منطقی ۱، ۲ و ۳ درست (true) نباشد، دستورات شماره ۴ انجام می شود.

```
if (عبارت منطقی(۱
{
 دستورات شماره;۱
}
elseif(عبارت منطقی(۲
{
دستورات شماره;۲
}
elseif(عبارت منطقی(۳
{
 دستورات شماره;۳
}
else
```

{

دستورات شماره۴;

}

در مثال بالا چنین ساختاری مورد استفاده قرار می گیرد.

```
class CheckSeason
{
static void Main
{
string input;
int number;
Console.Write(''Enter a number:'');
input = Console.ReadLine();
number = int.Parse(input);
if (number == ۱)
Console.WriteLine(''Spring'');
else if (number == ۲)
Console.WriteLine(''Summer'');
else if (number == ۳)
Console.WriteLine(''Fall'');
else if (number == ۴)
Console.WriteLine(''Winter'');
else
Console.WriteLine(''Invalid Number!.'');
```

```
Console.WriteLine("Press any key to continue...");
Console.ReadKey();
}
}
```

دستور switch

در مواردی که بخواهیم حالت های مختلف یک عبارت را بررسی و بر اساس آن دستورهایی را اجرا کنیم، از دستور switch استفاده می کنیم. ساختار کلی این دستور را می بینید.

```
switch (عبارت)
{
Case مقدار۱:
دستور۱;
break;
Case مقدار۲:
دستور۲;
break;
.
.
.
default:
دستورهای دیگر;
break;
```

}

در جلوی کلمه رزرو شده switch، عبارتی در داخل پرانتز نوشته می شود که بر اساس حاصل آن، تصمیم گیری و اجرای دستورها کنترل می شود. مقادیری که حاصل عبارت با آنها مقایسه می شود، هر یک در جلوی کلمه رزرو شده case نوشته می شود. اگر حاصل عبارت با یک مقدار case برابر باشد، آن گاه دستور یا دستورهای جلوی case تا رسیدن به کلمه رزرو شده break اجرا می شود. اگر حاصل عبارت با هیچ کدام از مقادیر case برابر نشد، آن گاه دستورهای قسمت default اجرا می شوند.

آکولاد های باز و بسته، محدوده عملیات شروع و پایان دستور switch را معین می کند. نوع عبارتی که داخل پرانتز دستور switch نوشته می شود، نمی تواند اعشاری باشد. اما عبارت های حرفی، رشته ای و انواع داده صحیح می تواند استفاده شود.

مثال - اکنون می خواهیم مثال بالا را با استفاده از دستور switch بازنویسی نماییم.

```
class CheckSeason
{
static void Main()
{
string input;
Console.Write("Enter a number:");
input = Console.ReadLine();
switch (input)
{
case "۱":
Console.WriteLine("Spring");
break;
case "۲":
```

```
Console.WriteLine("Summer");
break;
case "۳":
Console.WriteLine("Fall");
break;
case "۴":
Console.WriteLine("Winter");
break;
default:
Console.WriteLine("Invalid Number!");
break;
}
Console.WriteLine("Press any key to continue...");
Console.ReadKey();
}
}
```

مثال - می خواهیم برنامه ای بنویسیم که تشخیص سه کاربر مختلف، با دریافت نام کاربری را انجام دهد. در این برنامه، نام کاربری با دستور switch و کلمه عبور با دستور if بررسی شده است.

```
class SwitchDemo
{
static void Main(string[] args)
{
string userName, password;
```

```
Console.Write("Enter username:");
userName = Console.ReadLine();
Console.Write("Enter password:");
password = Console.ReadLine();
switch (userName)
{
case "admin":
if (password == "admin۱۲۳")
Console.WriteLine("Welcome Manager.");
else
Console.WriteLine("Wrong password!");
break;
case "accountant":
if (password == "acc۱۲۳")
Console.WriteLine("Welcome accountant.");
else
Console.WriteLine("Wrong password!");
break;
case "sales":
if (password == "sales۱۲۳")
Console.WriteLine("Welcome");
else
Console.WriteLine("Wrong password!");
break;
```

```
default:
Console.WriteLine("Invalid username!");
break;
}
Console.WriteLine("Press any key to continue...");
Console.ReadKey();
}
}
```

اگر به ساختار دستور switch توجه کنید، پس از هر دستور case، یک دستور break نوشته شده است. از کلمه رزرو شده break برای خاتمه دادن به یک case استفاده می شود. اگر دستور break نوشته نشود، مترجم برای جلوگیری از اشتباهِ برنامه نویس، خطا می دهد. برای هر case، می توان بیش از یک دستور نوشت و نیازی به بلاک ندارد.

مثال- با استفاده از قطعه برنامه زیر، متن کامل برنامه را در محیط VS بنویسید. در این قطعه برنامه از کاربر سؤالی پرسیده می شود، کاربر در پاسخ به سؤال، کلمه ای را وارد می کند. اگر کاربر کلمه Yes و یا کلمه maybe را وارد نماید، هر دو یک نتیجه را خواهد داشت و پیام Great! بر روی صفحه نشان داده می شود.

```
Console.Write("Do you enjoy C# ? (yes/no/maybe):");
string input = Console.ReadLine();
switch (input)
{
case "yes":
case "maybe":
Console.WriteLine("Great!");
break;
```

```
case ''no'':
Console.WriteLine(''Too bad!'');
break;
}
```

گام ششم

دستورات تکرار(حلقه ها)

دستورات تکرار شرطی

فرض کنید می خواهیم عبارت "amirmohammad" را در خروجی ۱۰ بار چاپ کنیم، با دستوراتی که تا اینجا گفتیم مجبوریم به تعداد ۱۰ بار دستور زیر را در خروجی چاپ کنیم:

```
Console.WriteLine(''AmirMohammad'');
```

در مثال بالا برای نوشتن ۱۰بار عبارت مورد نظر ۱۰خط برنامه در خروجی نوشته ایم، حال اگر بخواهیم ۱۰۰۰ بار عبارت را در خروجی چاپ کنیم، باید ۱۰۰۰ خط برنامه بنویسیم آیا این کار معقولانه به نظر می رسد، به طور حتم پاسخ این سؤال منفی می باشد برای حل این مشکل در برنامه نویسی از حلقه های تکرار استفاده می کنیم.

در چنین برنامه هایی، عمل دریافت اطلاعات، ممکن است تکرار گردد. تکرار دستورات یک برنامه، بسته به نوع الگوریتم آن، می تواند با دفعات معین و یا نامعین باشد. زمانی که تعداد دفعات نامشخص است، توقف و یا تکرار بستگی به برقراری یک شرط دارد. در این گونه موارد از دستورات حلقه شرطی مانند while یا do while استفاده می کنیم. اگر تعداد دفعات تکرار مشخص باشد، مثلاً حداکثر ۳ بار نام کاربری و کلمه عبور دریافت گردد، از دستور حلقه معین for استفاده می شود.

دستور حلقه شرطی while

ساختار کلی دستور while، در زیر نشان داده شده است:

While (عبارت منطقی)

دستور;

دستور while از سه بخش تشکیل شده است:

۱- کلمه رزرو شده while

۲- عبارت منطقی در داخل پرانتز

۳- دستوری که در صورت درست بودنِ نتیجه عبارت، اجرا خواهد شد.

مثال – در برنامه زیر با استفاده از دستور while اعداد ۱ تا ۱۰۰۰ در خروجی نمایش داده می شود.

```
int x=۱;
while (x < =۱۰۰۰)
Console.WriteLine("x=" + x++);
```

٪ به علامت نقطه ویرگول در دستور while توجه کنید. بعد از علامت پرانتز علامت نقطه ویرگول نگذارید، زیرا دستور while هنوز تمام نشده است. علامت نقطه ویرگول باید در انتهای دستور نوشته شود.

While (عبارت منطقی)

دستور;

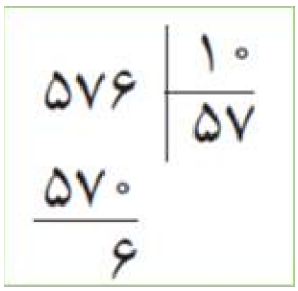
هنگامی که کامپیوتر در حال اجرای برنامه است، با رسیدن به دستور while، ابتدا مقدار عبارت را بررسی می کند. در صورتی که مقدار عبارت true باشد، دستور (یا بلاک)نوشته شده بعد از while، اجرا می شود.پس از آن، دوباره مقدار عبارت محاسبه می شود و تا زمانی که ارزش آن true باشد، دستورمذکور، اجرا خواهد شد. در این حالت می گوییم حلقه ایجاد شده است.

دستور یا دستوراتی که مکرر اجرا می گردند در بدنه حلقه قرار دارد. اگر در ارزیابی عبارت، مقدار false حاصل شود، دستورات بدنه حلقه دیگر اجرا نخواهند شد. برنامه از حلقه خارج می شود و دستورات بعدی اجرا می شوند.

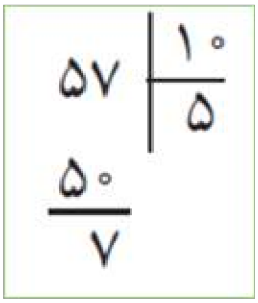
٪ اگر بخواهید بیش از یک دستور تکرار گردد، باید آنها را به صورت یک بلاک بنویسید. یعنی آنها را در داخل علامت های آکولاد باز و بسته قرار دهید.

مثال- می خواهیم برنامه ای بنویسیم که ارقام یک عدد را جدا نموده و آنها را نمایش دهد.

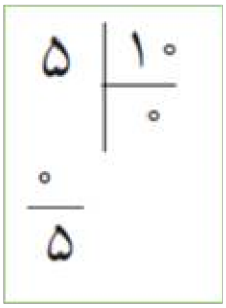
روش انجام کار: با توجه به آن که رقم یکان هر عدد، باقیمانده تقسیم آن عدد صحیح بر ۱۰ است، کافی است عدد دریافتی را بر ۱۰ تقسیم و باقیمانده آن را نمایش دهیم. به عنوان مثال اگر عدد دریافتی ۵۷۶ باشد باقیمانده تقسیم آن بر عدد ۱۰، عدد ۶ است که رقم یکان عدد است.



اگر دوباره خارج قسمت بدست آمده یعنی ۵۷ را بر عدد ۱۰ تقسیم کنیم، خواهیم داشت:



اگر به باقیمانده تقسیم بالا توجه کنید، متوجه می شوید که عدد ۷، رقم ده گان عدد دریافتی است. به همین ترتیب ادامه می دهیم و عمل تقسیم را تکرار کرده و باقیمانده تقسیم را به دست می آوریم.



باقیمانده تقسیم بالا را در نظر بگیرید. در اینجا توانستیم آخرین رقم عدد یعنی ۵ را نیز جداکنیم.

با دقت در عملیات فوق متوجه می شویم که عمل تقسیم، عملی تکراری است و تا زمانی انجام می شود که مقسوم آن بزرگتر از صفر باشد.

```
class Numbers
{
static void Main()
{
int number, digit;
string input;
Console.Write("Enter a number:");
input = Console.ReadLine();
number = int.Parse(input);
while (number > ۰)
{
digit = number ٪ ۱۰;
Console.WriteLine(digit);
number /= ۱۰;
}
Console.WriteLine("Press any key to continue...");
```

```
Console.ReadKey();
}
}
```

دستور حلقه شرطی do-while

ساختار این حلقه شبیه حلقه ی while می باشد با این تفاوت که در حلقه do -while- شرط در انتهای حلقه کنترل می شود.

ساختار حلقه do - while به صورت زیر است:

```
do
دستور;
)عبارت منطقی;(While
```

دستور do- while از چهار بخش تشکیل شده است:

۱- کلمه رزرو شده do

۲- دستور داخل حلقه

۳- کلمه رزرو شده while

۴- عبارت منطقی داخل پرانتز، که در صورت درست بودن آن، دستور داخل حلقه تکرار می شود.

مثال – در برنامه زیر با استفاده از دستور do-while اعداد ۱ تا ۱۰۰۰ در خروجی نمایش داده می شود.

```
int x=۱;
do
Console.WriteLine("x=" + x++);
while (x <= ۱۰۰۰);
```

٪ اگر بخواهید بیش از یک دستور در حلقه قرار گیرد، باید آنها را در یک بلاک قرار دهید.

به محل نوشتن علامت ; در دستور do- while توجه کنید. این علامت بعد از عبارت منطقی باید نوشته شود.

کامپیوتر با رسیدن به دستور do-while، ابتدا دستور داخل حلقه را اجرا می کند که از کلمه do شروع می شود و سپس با رسیدن به کلمه while، مقدار عبارت منطقی را ارزیابی می نماید. اگر حاصل عبارت true باشد، آنگاه به قسمت do برمی گردد و دستور بدنه حلقه اجرا می شود. تا زمانی که حاصل عبارت true است حلقه تکرار می شود. اگر حاصل ارزیابی عبارت false شود، دیگر به کلمه do برنمی گردد و کنترل برنامه به خط بعد از while واگذار شده و دستورات دیگرِ برنامه اجرا می شود.

٪ در دستور do … while در صورت برقرار نبودن شرط، حلقه حداقل یک بار اجرا می شود در صورتی که در حلقه while در صورتی که شرط برقرار نباشد، دستورات بدنه حلقه هرگز اجرا نمی شود. به مثال زیر توجه کنید:

```
int x=۱;
do
Console.WriteLine("x=" + x++);
while (x <= ۱۰۰۰);
```

در مثال بالا اگر دقت کنید شرط حلقه برقرار نمی باشد (x=>۱۰۰۰)، ولی با این حال مقدار x=۱ در خروجی چاپ می شود، دلیل آن هم کاملاً واضح است چون در ساختار do- while شرط حلقه در پایان آن بررسی می شود و در صورت برقرار نبودن شرط حلقه حداقل یک بار اجرا می شود.

دستور حلقه for

همان طور که گفتیم از حلقه های تکرار زمانی استفاده می کنیم که بخواهیم مجمو عه ای از عملیات را به دفعات معینی تکرار کنیم. از حلقه for هم برای همین منظور استفاده می کنیم.

شکل کلی دستور for چنین است:

for (نام متغیر = مقدار اولیه ; عبارت منطقی ;تغییرمقدار متغیر)

دستور;

برای یادگیری بهتر حلقه for به برنامه زیر توجه کنید:

مثال – می خواهیم اعداد طبیعی از ۱ تا ۱۰ را روی صفحه نمایش، نشان دهیم. از دستور for به صورت زیر استفاده می کنیم:

```
for (int a = ۱ ; a<= ۱۰ ; a++ )
Console.WriteLine( a);
```

حلقه for از ۳ قسمت تشکیل شده است :

۱- مقدار اولیه متغیر: که برای شروع اجرای حلقه استفاده می شود ،در مثال بالا مقدار اولیه (a=۱) است.

۲- عبارت شرطی: که در صورت برقراری شرط این عبارت حلقه ادامه می یابد در غیر این صورت اجرای حلقه متوقف می شود، در مثال بالا عبارت کنترلی (a<=۱۰) می باشد.

۳- میزان افزایش یا کاهش: در هر بار اجرای دستورات این مقدار افزایش یا کاهش می یابد، در مثال بالا میزان افزایشی (a++) است.

برای درک بیشتر حلقه for در جدول زیر نحوه اجرای برنامه بالا توضیح داده شده است.

اول	دوم	سوم	چهارم
a	شرط	خروجی	میزان افزایش
1	1<=10 درست	a=1	a++
2	2<=10 درست	a=2	a++
3	3<=10 درست	a=3	a++
4	4<=10 درست	a=4	a++
5	5<=10 درست	a=5	a++
6	6<=10 درست	a=6	a++
7	7<=10	a=7	a++
8	8<=10 درست	a=8	a++
9	9<=10 درست	a=9	a++
10	10<=10 درست	a=10	a++
11	11<=10 غلط		

مثال – حال همان مثال بالا را بر روی اعداد ۱۰ تا ۱ به صورت نزولی می نویسیم:

```
for (int a = ۱۰ ; a>= ۱۰ ; a-- )
Console.WriteLine( a);
```

همان طور که در مثال بالا مشاهده می کنید برای اینکه اعداد ۱ تا ۱۰ به صورت نزولی در خروجی نمایش داده شود مقدار اولیه برابر ۱۰، شرط تغییر کرده و میزان افزایش به کاهش تبدیل می شود.

مثال – می خواهیم برنامه ای برای ATM بنویسیم که سه بار اجازه ورود گذرواژه را به کاربر بدهد و در صورتی که هر سه بار گذرواژه اشتباه وارد شود، کارت توسط دستگاه عابربانک با صدور پیغامی ضبط می شود.

روش انجام کار: دستگاه عابر بانک از شما سه بار گذرواژه ورود می گیرد. در صورتی که مقدار ورودی صحیح باشد، شما با پیغام خوش آمدگویی مواجه می شوید و با دستور break از حلقه خارج می شوید و در صورتی که مقدار ورودی درست نباشد، تا ۳ بار اجازه وارد کردن دارید وگرنه کارت با پیغامی ضبط می شود. در این

برنامه برای دریافتِ ۳ بار گذرواژه، از حلقه for استفاده شده است. هر بار که گذرواژه از ورودی دریافت می شود، با گذرواژه اصلی مقایسه می شود و در صورت صحیح بودن علامتی به نام متغیر flag مقدار true می گیرد. این متغیر در آغاز برنامه مقدار false دارد و تا زمانی که گذرواژه درست وارد نشود همچنان false می ماند.

```
using System;
class ATMGetPass
{
static void Main ()
{
int i;
string mainPassword = "۱۴۰۴", password;
bool flag = false;
for (i = ۳; i >=۱; i--)
{
Console.Write("Enter Password:");
password = Console.ReadLine( );
Console.WriteLine();
if (password == mainpassword)
{
flag = true;
Console.BackgroundColor = ConsoleColor.DarkGreen;
Console.WriteLine(" That is right...");
Console.Beep( );
```

```
break; // خروج از حلقه
}
else
Console.WriteLine(" PLZ try again...");
}
if (flag = false)
{
Console.BackgroundColor = ConsoleColor.Red;
Console.WriteLine(" Your card will no longer work!!!");
}
}
}
```

٪ توجه داشته باشید، که هیچ کدام از سه قسمت داخل پرانتز، در دستور for اجباری نیستند. حتی دستور for، می تواند به صورت زیر نوشته شود که در این صورت، یک حلقه تمام نشدنی و بی نهایت ایجاد می شود.

```
for ( ; ; )
Console.WriteLine("Infinite Loop! ");
```

کاربرد دستور continue در ساختار for

مثال – همه شما بازی هپ را می شناسید و بارها برای تمرین جدول ضرب در دوران ابتدایی این بازی را انجام داده اید. در این برنامه می خواهیم بازی هپ را برای عدد ۵ شبیه سازی کنیم.

روش انجام کار: در بازی هپ، هرجا که به مضرب عدد تعیین شده می رسیم باید پیغام هپ را چاپ کنیم. برای این کار، حلقه ای را تا مثلاً ۵۰ در نظر می گیریم تا هرجا به مضرب ۵ رسید، هپ با رنگ دیگری چاپ

شود. برای رسیدن به مضرب ۵ از شرطی با عملگر ٪ برای باقیمانده کمک می گیریم. در نگاه اول برنامه با آنچه تاکنون آموخته اید قابل نوشتن است.

```
using System;
class hop
{
static void Main (string [ ] args)
{
for (int i = ١; i<=٢٠; i++)
{
if (i ٪ ۵ ==٠)
{
Console.ForegroundColor = ConsoleColor.Yellow;
Console.WriteLine("hop");
Console.ForegroundColor = ConsoleColor.Gray;
}
Console.Write(" {٠}" , i);
}
Console.ReadKey();
}
}
```

حلقه های متداخل

درون حلقه ممکن است دستور یا دستوراتی وجود داشته باشند. دستور یا دستورات داخل حلقه به تعداد تکرار حلقه، انجام می شود. بنابراین اگر حلقه ای مثلاً ۵ بار تکرار شود، دستورات داخل آن ۵ بار انجام می شوند.

دستور داخل حلقه ممکن است خودش، یک حلقه باشد. بنابراین حلقه هایی تودرتو خواهیم داشت که حلقهٔ داخلی به تعدادِ دفعاتِ تکرارِ حلقهٔ بیرونی، تکرار می شود.

مثال – در این برنامه قصد داریم یک ساعت دیجیتال را با ساعت و دقیقه شبیه سازی کنیم.

روش انجام کار: ابتدا بخش نمایش دقیقه را کدنویسی می کنیم. برای نمایش دقیقه نیاز به حلقه ای داریم که ۶۰ بار کار کند (چرا؟) . این بخش با حلقهٔ بسیار ساده قابل کدنویسی است.

```
for (int min = ۰; min<۶۰; min++)
Console.WriteLine("{۰}", min);
```

با اجرای کد بالا دقیقه های یک ساعت شبیه سازی می شود. در صورتی که بخواهیم ۱۲ ساعت را شبیه سازی کنیم کافیست دستورات بالا را ۱۲ بار تکرار کنیم. بنابراین دستورات را درون حلقه ای می نویسیم که ۱۲ بار تکرار می شود.

```
for (int hour = ۱; hour <= ۱۲; hour++)
for (int min = ۰; min<۶۰; min++)
Console.WriteLine("{۰}", min);
```

با تغییراتی در دستور WriteLine می توانیم ساعت را هم نمایش دهیم. بنابراین برنامه به شکل زیر خواهد بود:

```
using System;
class clock
{
static void Main()
{
```

```
for (int hour = ۱; hour <= ۱۲; hour++)

for (int min = ۰; min<۶۰; min++)

Console.WriteLine("{۱}: {۱}" , hour , min);

{

{
```

منابع

برنامه نویسی به زبان C، مؤلف جعفرنژاد قمی

آموزش گام به گام برنامه سازی به زبان C#، مؤلف جعفرنژاد قمی