دانشگاه سیستان و بلوچستان
دانشکده مهندسی شهید نیکبخت
گروه مهندسی فناوری اطلاعات

برنامه سازی پیشرفته

نیمسال دوم ۸۵

زمانبندی کلاس

هفته	تاریخ	تعداد جلسه	مبحث
اول	۲۳ تا ۲۶ بهمن	۱	سیلابس، مراجع، بارم‌بندی،قوانین
دوم	۲۸ بهمن تا ۳ اسفند	۱+۲ جبرانی	معرفیNet.،مبانی C# و کار در کارگاه (حل تمرین)
سوم	۵ تا ۱۰ اسفند	۲	متدها
چهارم	۱۲ تا ۱۷ اسفند	۱+۲ جبرانی	رشته‌ها، متدهای بازگشتی، انواع مقداری و ارجاعی
پنجم	۲۰ تا ۲۴ اسفند	۲	کلاسها، سطوح دسترسی، فیلدهای Static
ششم	۱۸ تا ۲۳ فروردین	۱+۲ جبرانی	Property، Indexer،سربارگذاریعملگر ، enum وStruct
هفتم	۲۵ تا ۳۰ فروردین	۲	آرایه ها
هشتم	۱ تا ۶ اردیبهشت	۱+۲ جبرانی	Collectionها
نهم	۸ تا ۱۳ اردیبهشت	۲	مدیریت خطا،آرگومان سطر فرمان، پارامترهای Params+وراثت
دهم	۱۵ تا ۲۰ اردیبهشت	۲	ادامه وراثت + فایل
یازدهم	۲۲ تا ۲۷ اردیبهشت	۲	ادامه فایلها
دوازدهم	۲۹ اردیبهشت تا ۳ خرداد	۲	بخش ویندوز
سیزدهم	۵ تا ۱۰ خرداد	۲	
چهاردهم	۱۲ تا ۱۶ خرداد	۱	
تعداد کل جلسات		۳۰	

مراجع

1. C# Console Application, Satish Talim.(Study note)
2. Windows Form Programming With C#, Erick Brown
3. C# How to Program, Deitel (ترجمه مولاناپور)
4. MS Visual C# 2005, John Sharp (ترجمه مهرداد توانا)
5. My Lecture Note (جزوه‌های کلاسی)

بارم بندی

نوع آزمون	حجم آزمون	درصد
تمرین و کوئیز	٤ کوئیز +هر فصل تمرین جداگانه	٪۱۰
پروژه	٤	٪۲۵
میان‌ترم	تا پایان هفته هفتم	٪۲۵
پایان‌ترم	از ابتدا تا انتهای مباحث	٪٤۰
نظم و انظباط، فعالیت کلاسی	حداکثر تا سقف ۱ نمره علاوه بر ۲۰ نمره	

مباحث کوئیزها:

هفته اول تا چهارم، پنجم و ششم، هفتم و هشتم ، نهم تا یازدهم

مباحث پروژه‌ها:

هفته اول تا چهارم، پنجم تا هشتم ، نهم تا یازدهم ، دوازدهم تا چهاردهم

فصل اول: مقدمه ای بر .Net

۱. .Net چیست؟

اصولا هر تکنولوژی جدیدی درجهت رفع نواقص و کمبودهای تکنولوژیهای قبلی بوجود می آید. قبل از ایجاد تکنولوژی .Net محیط مجتمع و یکپارچه ای برای برنامه نویسی تحت اینترنت وجود نداشت و این مساله یکی از دغدغه های اصلی برنامه نویسان وب به حساب می آمد.

با وجود اینکه تلاشهای زیادی در جهت رفع این مشکل در تکنولوژیهای قبلی انجام گرفته بود اما کماکان مشکلاتی برای برنامه نویسان اینترنت وجود داشت که در زیر به پاره ای از آنها اشاره می شود:

از سال ۱۹۹۵ به بعد شرکت ماکروسافت تلاشهایی را در جهت برنامه نویسی در محیط اینترنت آغاز نمود. که نمونه ای از این تلاشها استفاده از COM و +COM می باشد که اغلب سایتهای بزرگ اینترنتی از آنها استفاده می کردند. با وجود ارائه COM باز هم یکی از عوامل فراگیر نشدن برنامه نویسی در محیط اینترنت، پیچیده بودن برنامه نویسی با COM بود که ماکروسافت برای رفع این مشکل تکنولوژی ASP را ارائه داد. با ارائه ASP ،انجام عمل برنامه نویسی در محیط اینترنت تا حد زیادی ساده تر گردید. اما هنوز مشکلاتی نیز در این تکنولوژی وجود داشت که از بارزترین آنها می توان به عدم پشتیبانی ASP از شیءگرایی اشاره نمود.

یکی دیگر از مشکلات برنامه نویسی با ASP ، کمبود رابط کاربر مناسب برای برنامه نویسی در محیط اینترنت می باشد.

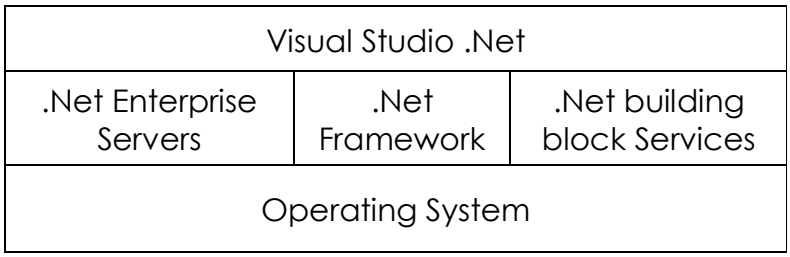
از دیگر مشکلات ASP می توان به این نکته اشاره نمود که کاربر مجبور به یادگیری یک زبان اسکریپت نویسی است که این کار نیز نیاز به صرف هزینه زمانی دارد.

با وجود مشکلات فوق ماکروسافت تصمیم به ایجاد یک تکنولوژی جدید گرفت که در آن هم بتوان برنامه نویسی کاربردی در محیط ویندوز را انجام داد و هم اینکه بدون داشتن مشکلات تکنولوژیهای قبلی به برنامه نویسی در محیط اینترنت پرداخت این تکنولوژی موسوم به .Net می باشد که در ادامه بصورت کاملتر به آن پرداخته خواهد شد.

همچنین در این تکنولوژی کدهای اجرایی تولید شده وابسته به ماشین و یا سیستم عامل نمی باشند که همین عمل سبب شده است تا کدهای .Net دارای قابلیت حمل بالایی باشند.

۲. .Net Platform

مجموعهٔ .Net دارای پنج جزء اصلی می باشد که عبارتند از:



شکل ۱-۱: Microsoft .Net Platform

۱. سیستم عامل

در پایین ترین لایه سیستم عامل قرار گرفته است که عامل اجرایی برنامه های نوشته شده تحت .Net می باشد. سیستم عامل می تواند یکی از انواع سیستم عاملهایی باشد که با .Net سازگار است.

در حال حاضر می توان از یکی از انواع ویندوزها به جای سیستم عامل استفاده نمود اما ماکروسافت با همکاری شرکتهایی از قبیل IBM، Intel و ... در حال ایجاد framework برای سیستم عاملهایی از قبیل Linux، Free-BSD و ... می باشد که نسخهٔ لینوکس آن تحت عنوان Mono در حال بهره برداری است.

۲. .Net Enterprise Servers

بر روی سیستم عامل مجموعه ای از سرورهای حرفه ای ماکروسافت قرار گرفته است که برای انجام پروژه های حرفه ای الزامی می باشند. از این قبیل می توان به SQL Server که برای ایجاد و مدیریت بانکهای اطلاعاتی بکار می رود، اشاره نمود.

۳. .Net Building Block service

ماکروسافت با توجه به نیاز برنامه نویسان وب، اقدام به ایجاد و ارائه سرویسهایی نموده است که برنامه نویسان می توانند در ایجاد و پیاده سازی برنامه های تحت وب خویش از آنها استفاده نمایند. نمونه ای از این سرویسها Microsoft Passport service می باشد که به کاربر اجازه می دهد تا با کلمه عبور و رمز یکسانی به منابع کلیه سایتهایی که از این سرویس پشتیبانی می کنند، دسترسی داشته باشد. از دیگر سرویسهایی که در حال طراحی می باشند می توان به سرویس تقویم و سرویس جستجو اشاره نمود.

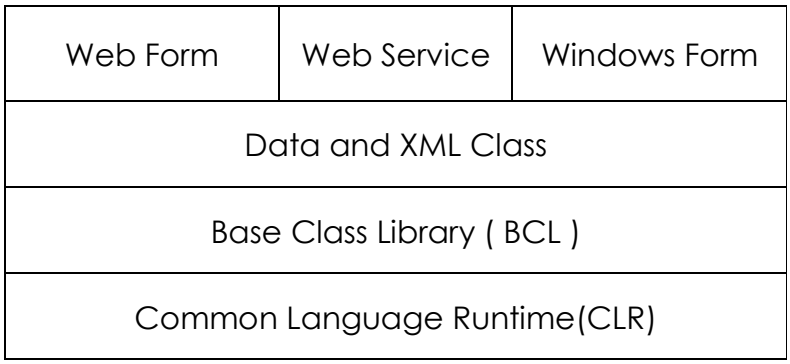
۴. Visual Studio .Net

در بالاترین لایه ویژوال استودیو دات نت قرار دارد که ابزاری بسیار قدرتمند جهت پیاده سازی برنامه های تحت وب و کاربردی می باشد. از این ابزار می توان جهت تسریع در پیاده سازی سرویسهای وب و برنامه های کاربردی بهره برد که با قابلیت های بیشتر و سرعت بالاتری نسبت به ویژوال استودیوی ۶ ارائه شده است.

۵. .Net Framework

مرکز .Net دارای بخشی با نام .Net Framework می باشد که یک محیط و زیرساخت جدید برای ایجاد و اجرای برنامه های نوشته شدهٔ تحت .Net است.

این جزء حاوی چهار بخش اصلی می باشد که عبارتند از:



شکل ۱-۲: .Net Framework

۵-۱. CLR

می توان CLR را به عنوان قلب .Net در نظر گرفت که در واقع عمل اجرای برنامه های نوشته شده به کمک .Net را بر عهده دارد. CLR به هنگام اجرای برنامه ها اعمالی از قبیل : تخصیص حافظه، آزاد سازی حافظه، چک کردن خطاها، محاورهٔ با سیستم عامل و غیره را انجام می دهد.

۵-۲. BCL

این بخش حاوی مجموعه ای غنی از کلاسها می باشد که برنامه نویسان می توانند از آنها استفاده نموده و با سرعت بیشتری برنامه های خویش را پیاده سازی نمایند. در واقع با بهره گیری از این مجموعهٔ غنی می توان قابلیت استفاده مجدد از نرم افزارها را بهبود بخشید. در این مجموعهٔ گرانبها کلاسهایی برای پشتیبانی از اعمال ورودی/خروجی، کار با رشته ها،

مدیریت امنیت، برنامه نویسی شبکه، کار با فایلها و مجموعه ها و هزاران قابلیت دیگر نهفته است.

۵-۳. Data and XML Class

لایهٔ بالاتر حاوی مجموعه کلاسهایی است که برای مدیریت داده ها و کار کردن با XML مناسب می باشند. کلاسهای بخش مدیریت داده ها حاوی کلاسهایی جهت کار با پایگاههای داده می باشند که از آن جمله می توان به کلاسهای ADO.Net اشاره نمود که امکان ذخیره سازی، بازیابی و تغییر داده های پایگاه داده را به برنامه نویس می دهد.

همچنین این بخش حاوی کلاسهایی است که کارکردن با داده های نوع XML را تسهیل می نماید.

۵-۴. Windows/Web Forms and Web service

این بخش حاوی کلاسهایی است که کلاسهای دو لایهٔ قبلی را توسعه می دهند. به عنوان مثال بخش Windows form حاوی مجموعه کلاسهایی است که کمک می کند تا طراحی فرمهای ویندوز با سرعت بالاتری انجام پذیرد.

همچنین بخش Web Form به برنامه نویسان امکان Drag کردن کامپوننت ها را مشابه فرمهای ویندوز می دهد در حالیکه در تکنولوژیهای قبلی چنین قابلیتی وجود نداشته است.

بخش Web service حاوی مجموعه کلاسهایی است که به برنامه نویسان امکان ایجاد کامپوننت های توزیع شده را می دهد که می توانند در نرم افزارهای مربوط به دیواره آتش و NAT مورد استفاده قرار گیرند.

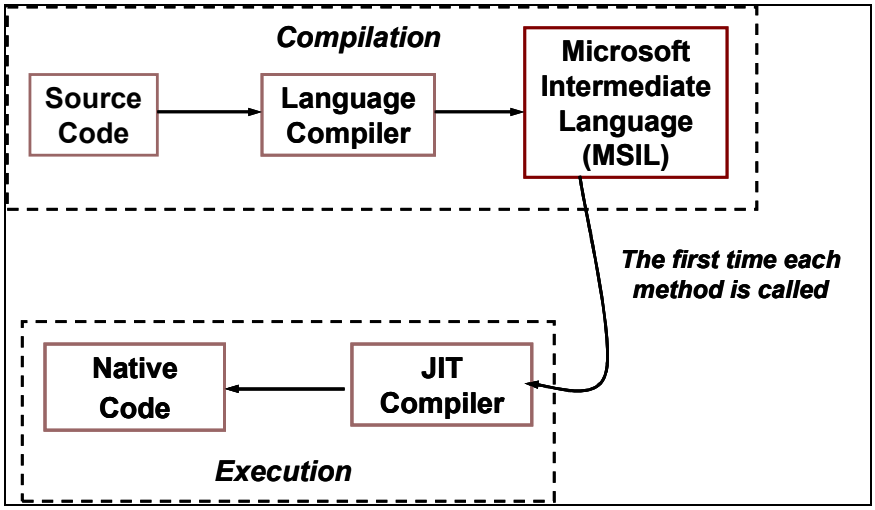
۳. Common Language Runtime (CLR)

CLR بعنوان هستهٔ مرکزی .Net Framework. مدیریت اجرای برنامه ها را بر عهده دارد و در واقع CLR همان نقش JVM را در سیستمهای مبتنی بر Java بازی می کند.

قبل از اینکه به جزئیات CLR پرداخته شود به نحوهٔ کامپایل و اجرای یک برنامه .Net. می پردازیم.

همانگونه که در شکل ۱-۳ ملاحظه می نمایید هر برنامه ای که تحت .Net. نوشته شود، پس از کامپایل تبدیل به یک فایل EXE و یا DLL می شود که این فایل با فایلهای اجرایی ویندوز متفاوت می باشد. به این فایلها اصطلاحا اسمبلی گفته می شود که هم حاوی کد و هم حاوی داده هایی اضافی به نام Metadata می باشند.

کدهای تولید شده در این مرحله وابسته به ماشینی که برنامه تحت آن کامپایل می شود ندارند، بلکه تحت زبانی به نام Microsoft Intermediate Language می باشند که یک زبان شی گرا با گرامری مشابه زبان اسمبلی می باشد.



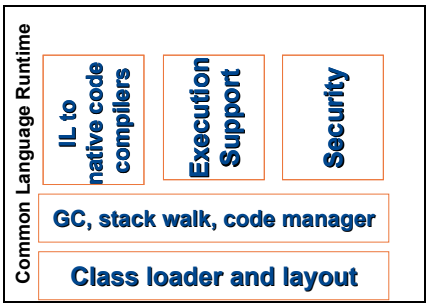
شکل ۱-۳: کامپایل و اجرای یک برنامه تحت .Net

این کدها برای کلیهٔ زبانها یکسان می باشند و بطور خلاصه به آنها کدهای MSIL و یا IL گفته می شود. پس برنامه های نوشته شده تحت کلیه زبانهای مجموعه .Net. پس از کامپایل به این زبان سطح میانی تبدیل می شوند.

در زمان اجرا، CLR به کمک کامپایلر JIT ، این کدهای میانی را با توجه به ماشینی که برنامه بر روی آن اجرا می شود به دستورات اجرایی آن ماشین تبدیل می نماید. برای انجام این کار فقط کافی است که بر روی ماشین مقصد .Net Framework نصب شده باشد.

در واقع با این روش کامپایل و اجرای برنامه، دیگر کدهای تولید شده به کمک .Net به ماشین وابسته نمی باشد یا به اصطلاح کدهای تولید شده با این روش قابل حمل می باشند.

CLR از اجزاء زیر تشکیل شده است:



شکل ۱-۴: اجزاء CLR

همانطور که در شکل ۱-۴ مشاهده می کنید CLR از سه لایه تشکیل شده است که بصورت خلاصه به هر کدام از آنها پرداخته می شود.

۱. در لایه زیرین بارکننده کلاسها قرار دارد که پیش از تبدیل کدهای IL به کد ماشین مقصد، برای بارنمودن کلاسهای مورد نیاز از آن استفاده می شود. همچنین کلاسهایی را که اطلاعات امنیتی نیز در آنها گنجانده شده است می توان توسط این بخش و قبل از کلاسهای اصلی برنامه بار نمود.
۲. در لایه بالاتر در واقع بخش مدیریتی CLR واقع شده است که شامل Garbage Collector جهت مدیریت منابع تخصیص داده شده، Code Manager جهت مدیریت کدها تا اینکه کدهای نوشته شده از منابع مجاز خود استفاده نمایند و Stack Wall که برای نگهداری متغیرهای نوع مقداری برنامه مورد استفاده واقع می شود.
۳. در لایه بالاتر کامپایلر Just In Time قرار دارد که برای تبدیل کدهای IL به کدهای ماشین خاص از آن استفاده می شود و همچنین مسایل امنیتی در این بخش چک می شوند.

۴. Base Class Library

.Net حاوی مجموعه ای غنی از کلاسها است که برنامه نویسان می توانند از این مجموعه کلاسها برای ایجاد برنامه های خود استفاده نمایند. فضاهای نامی که در این مجموعه واقع شده و حاوی کلاسها می باشند در زیر لیست شده اند:

۱. در فضای نام Collection مجموعه ای از کلاسها وجود دارند که برای نگهداری مجموعه ای از داده ها و انجام عملیات بر روی داده ها از آن کلاسها استفاده می گردد. مانند ArrayList، Hashtable و غیره
۲. در IO مجموعه ای از کلاسها برای کار بر روی انواع فایلهای متنی و باینری گنجانده شده است.
۳. در Net مجموعه ای از کلاسها برای برنامه نویسی تحت شبکه قرار داده شده است که با بهره گیری از این مجموعه کلاسها براحتی می توان برنامه های سوکت تحت شبکه را نوشت.
۴. در Globalization نیز مجموعه ای از کلاسها که عمومیت بیشتری دارند گنجانده شده اند مانند الگوریتمهای مرتب سازی، تقویم های مختلف از جمله کلاس PersianCalendar که برای ایجاد تقویم فارسی از آن استفاده می شود.
۵. و بقیه کلاسها که به تدریج با آنها اشنا خواهید شد.

System

Collections	Security
Configuration	ServiceProcess
Diagnostics	Text
Globalization	Threading
IO	Runtime
Net	InteropServices
Reflection	Remoting
Resources	Serialization

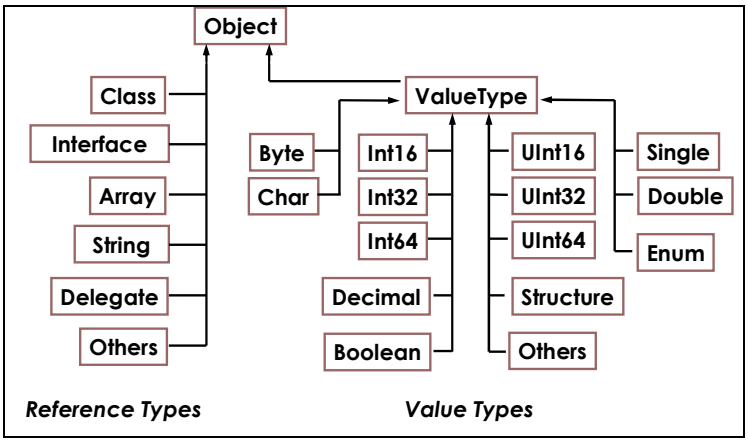
شکل ۱-۵: فضاهای نام BCL

۵. چند زبانه بودن

یکی از خواص اصلی .Net چند زبانه بودن آن است، یعنی می توان در هنگام استفاده از یک زبان از کامپوننت های نوشته شده به زبان دیگر نیز استفاده نمود. همچنین می توان از کامپوننت های زبان دیگر ارث برد و یا خطاهای مربوط به آن کامپوننت ها را کنترل نمود.

علت اصلی استفاده از چند زبان در این تکنولوژی این است که کلیه زبانها پس از کامپایل به زبان یکسانی (IL) ترجمه می شوند. و همه این زبانها پس از ترجمه از انواع یکسانی بهره می برند . در واقع انواع موجود در زبانها نیز پس از کامپایل به انواع یکسانی تبدیل می شوند(به یکی از انواع موجود در CTS).

انواع موجود در .Net که بقیه انواع تعریف شده در زبانها به این انواع تبدیل می شوند در شکل ۱-۶ آمده اند.



مثلا نوع int در C# پس از کامپایل به معادل Int32 و نوع float به معادل Decimal تبدیل می گردد. به همین صورت نیز انواع بقیه زبانها پس از کامپایل باید به این انواع تبدیل گردد.

باید توجه شود که اگر قصد نوشتن کامپوننتی را داریم که می خواهیم در چندین زبان از آن استفاده نماییم باید از استاندارد Common Language specification (CLS) پیروی نماییم.

مثلا در این حالت نمی توان از اشاره گر های C# در واسط کلاسها استفاده نمود، چون زبانهایی از قبیل VB.Net که در این مجموعه وجود دارند نمی توانند از اشاره گر استفاده نمایند.

۶. مدیریت خودکار حافظه

در .Net بر خلاف تکنولوژیهای قبلی مدیریت حافظه بصورت خودکار انجام می گیرد. یعنی پس از اخذ حافظه از سیستم برنامه نویس مجاز به اینکه دوباره صریحا حافظه را آزاد نماید، نیست. بلکه خود سیستم (CLR) پس از تشخیص حافظه های بلااستفاده، آنها را آزاد می نماید.

در واقع هر گاه حافظه کافی جهت تخصیص به شیء جدیدی وجود نداشته باشد، سیستم اقدام به آزادسازی حافظه های بلا استفاده می نماید(با فراخوانی Garbage Collection) تا بقیه اشیاء از این حافظه های آزاد شده استفاده نمایند.

در این حالت سیستم همیشه لیستی از منابع مورد استفاده برنامه ها را در اختیار دارد و هر گاه با کمبود حافظه مواجه گردد لیست موردنظر را چک کرده و چنانچه منبعی وجود داشته باشد که هیچ ارجاعی به آن نداشته باشیم آن منبع را آزاد نموده و به درخواست کننده تخصیص می دهد.

زمان اجرای دقیق Garbage Collector مشخص نیست و خود CLR در صورت نیاز آنرا فراخوانی می نماید. اما اگر برنامه نویس تشخیص دهد که در مقطعی از زمان حافظه های بلااستفاده داریم می تواند صریحا اقدام به فراخوانی Garbage Collection نماید.

۷. کدهای مدیریت شده و مدیریت نشده

به کلیه کدهای نوشته شده به کمک زبانهای موجود در مجموعه .Net. که توسط CLR اجرا می گردند، اصطلاحا کدهای مدیریت شده گفته می شود. یعنی رفتار و اجرای این کدها توسط CLR مدیریت می شود.

در اینگونه کدها داده هایی تحت عنوان Metadata وجود دارد که CLR با کمک گرفتن از این داده ها چگونگی اجرای کدها را تحت نظر گرفته و مدیریت می نماید تا اجرای آنها در حالت امن صورت پذیرد. منظور و هدف از اجرای امن کدها این است که:

- مدیریت حافظه این مجموعه کدها به درستی انجام گیرد.
- چکهای امنیتی لازم نیز به درستی کنترل گردد.
- تبدیل انواع بصورت صحیح انجام پذیرد.
- از کامپوننت های نوشته شدۀ یک زبان در زبان دیگر به درستی استفاده گردد.

اما .Net به کاربر این امکان را می دهد تا بتواند کدهای مدیریت نشده هم بنویسد یعنی کدهایی که ممکن است به نواحی غیر مجاز حافظه دسترسی داشته باشند.

۸. Assembly

در واقع با کامپایل یک برنامۀ نوشته شده تحت .Net یک اسمبلی تولید می شود که هر اسمبلی می تواند یک فایل اجرایی و یا حاوی چندین کامپوننت (DLL) باشد. علاوه بر اینها در هر اسمبلی داده هایی تحت عنوان Metadata ذخیره می گردد که CLR از این داده ها در اجرای صحیح برنامه بهره می برد.

این داده ها می توانند در چک کردن انواع، مسایل امنیتی و غیره مورد استفاده واقع شوند.

۹. Metadata

مجموعه داده هایی هستند که به CLR اجازه می دهند تا به جزئیات مربوط به یک کامپوننت دسترسی داشته باشد. این داده ها برای هر کاپوننت در زمان کامپایل ایجاد شده و توسط CLR در زمان اجرا مورد استفاده قرار می گیرند. که CLR به کمک این دادها می توانند به متدها و خواص مربوط به اشیاء ایجاد شده دسترسی داشته باشد.

این اطلاعات درون خود کامپوننت ذخیره می شوند که شامل:

- نام و نوع کلیه اجزاء موجود در کامپوننت مانند متدها، فیلدها ، خواص ، رخدادها و غیره می باشند
- به ازای هر متد حاوی اطلاعاتی است که بارکننده برای پیداکردن بخش پیاده سازی شده متد از آنها استفاده می نماید.
- همچنین می تواند حاوی اطلاعاتی در مورد پارامترهای متد و انواع آنها باشد.
- نام و نسخه اسمبلی
- کلیه اسمبلی هایی که از طریق این اسمبلی مورد ارجاع واقع شده اند
- اجازه دسترسی ها

- اطلاعاتی در مورد شرکت سازنده این محصول
- و غیره

۱۰. Assembly Cache

یک دایرکتوری بر روی سیستم که معمولا در دایرکتوری\Assembly\ درون دایرکتوری ویندوز قرار دارد و هرگاه که یک اسمبلی بر روی سیستم نصب می شود، در واقع آن اسمبلی به این دایرکتوری اضافه می گردد.

این دایرکتوری می تواند همزمان چندین نسخه از یک اسمبلی یکسان را در خود نگهداری نماید. در واقع با نصب یک برنامه جدید، هرکز نسخه قبلی اسمبلی مورد استفاده این برناکه که قبلا در این دایرکتوری وجود دارد، از بین نمی رود. در واقع با این عمل از Versioning در .Net پشتیبانی می شود.

۱۱. چرا #C ؟

با توجه به اینکه زبانهای زیادی درمجموعهٔ .Net وجود دارد، چرا زبان#C برای تدریس انتخاب شده است؟

#C به دلایل زیر نسبت به بقیهٔ زبانها ترجیح داده شده است:

۱. سادگی #C
۲. شناخته شدن به عنوان زبان مادر .Net
۳. استاندارد شدن توسط انجمن تولید کنندگان کامپیوتر اروپا(ECMA) با شماره ECMA-334
۴. در حال استاندارد شدن توسط موسسه ANSI
۵. و غیره

فصل دوم: مقدمه ای بر C#

۱) اجزاء یک برنامه

جهت آشنایی با اجزاء یک برنامهٔ ساده با C#، ابتدا به ذکر یک مثال پرداخته و سپس به کمک آن اجزاء یک برنامه را شرح می دهیم.

```
class Welcome
{
    static void Main()
    {
        System.Console.WriteLine("Welcome to USB C# Course!");
    }
}
```

این برنامه عبارت Welcome to USB C# Course را در خروجی چاپ می نماید.

در نوشتن برنامه های C# به نکات زیر توجه شود:

- هر برنامه حداقل حاوی یک کلاس می باشد که نحوهٔ تعریف کلاس بصورت زیر است.

```
class Welcome
{

}
```

- حداقل یکی از کلاسهای برنامه حاوی متدی (تابعی) به نام Main می باشد که این تابع نقطهٔ شروع اجرای برنامه می باشد. (entry point)
- متد Main باید بصورت static تعریف گردد. (در ادامه با علّت آن آشنا خواهید شد.)

حال به دستوری که جهت چاپ عبارت مورد نظر بکار می رود، دقت نمایید.

```
System.Console.WriteLine
```

مشاهده می گردد که دستور چاپ از سه بخش تشکیل شده است:

- WriteLine: نام تابعی است که جهت چاپ عبارت مورد نظر بکار می رود.
- Console: نام کلاسی است که تابع مورد نظر در آن قرار دارد. قابل ذکر است که هر کلاس می تواند شامل چندین متد/تابع باشد.
- System: نام فضای نام (Namespace)ی است که کلاس مورد نظر در آن واقع شده است. هر فضای نام می تواند شامل چندین کلاس باشد.

فضای نام (Namespace) چیست ؟

مکانیزمی است سبب سازماندهی کدها که باعث بالا رفتن قابلیت استفاده مجدد کدها می گردد.

فصل دوم: مقدمه ای بر C#

چرا به فضاهای نام احتیاج داریم؟

علت اصلی نیاز برنامه نویسان به فضاهای نام، جلوگیری از تداخل نامها در برنامه ها و کدهاست.

برای درک بهتر مطلب فرض نمایید که می خواهیم برنامه ای به زبان C بنویسیم. در این برنامه نیاز به تعریف متغیر و تابع ضروری به نظر می رسد. یکی از اصول برنامه نویسی نامگذاری مناسب شناسه ها در برنامه می باشد بگونه ای که در نامگذاری آنها باید کمال دقت به عمل آید تا با هیچکدام از شناسه هایی که در کتابخانه های مورد استفاده برنامه بکار رفته اند، همنام نباشند. در غیر اینصورت برنامه به درستی کامپایل و اجرا نخواهد شد.

یا اگر بخواهیم از یک کامپوننت نوشته شده توسط شرکتی دیگر در برنامه خویش استفاده نماییم، مشابه قبل باید توجه خاصی را به تداخل نامها مبذول داریم. که این عمل تا اندازه ای وقتگیر و مشکل خواهد بود. بخصوص اگر حجم برنامه ، تعداد توابع و متغیرها و تعداد کتابخانه های مورد استفاده زیاد باشد.

برای رفع این مشکل در .Net از فضاهای نام استفاده می گردد. به اینصورت که مجموعه ای از کلاسهای مرتبط را در یک فضای نام قرار می دهد. که در این حالت فقط کافی است تا فضاهای نام با همدیگر همنام نباشند و همچنین کلاسهای درون یک فضای نام مشترک و شناسه های درون یک کلاس نیز با همدیگر همنام نباشند.

معمولا برای هر برنامه نوشته شده یک فضای نام درنظر می گیریم که نام آن را اسم سازمان و یا اسم پروژه انتخاب می نماییم.

```
namespace USBCSharp
{
    class Welcome
    {
        static void Main()
        {
            System.Console.WriteLine("Welcome to USB C# Course!");
        }
    }
}
```

می توان فضاهای نام تودرتو (متداخل) نیز تعریف نمود. به کد زیر دقت نمایید که در آن:

فضای نام NS1 بالاترین سطح ممکن می باشد که درون آن دو فضای نام دیگر با اسامی NS1-1 و NS1-2 قرار دارند. همچنین درون هر کدام از این فضاهای نام دو کلاس وجود دارد و هر کلاس نیز دارای توابعی می باشد.

توجه نمایید که همنامی دو کلاس درون دو فضای نام متفاوت و یا همنامی دو متد درون دو کلاس متفاوت هیچ مشکلی را برای زمان اجرا و یا زمان کامپایل ایجاد نمی نماید.

```
namespace NS1
{
    namespace NS1_1
    {
        class c1
        {
            static void tst1()
            { }
            int tst2()
            { }
        }
        class c2
        {
            int tst2()
            { }
        }
    }
    namespace NS1_2
    {
        class c1 {/*...*/ }
        class c3 {/*...*/ }
    }
}
```

همانگونه که در کد بالا مشاهده می شود، کلاس c1 در دو فضای نام مشترک می باشد و همچنین متد tst2 در دو کلاس c1 و c2 مشترک است.

حال برای اجرای متد tst2 درون کلاس c1 از فضای نام NS1-1 باید بصورت زیر عمل نمود:

```
NS1.NS1_1.c1.tst2 ();
```

با توجه به وجود فضاهای نام متداخل، فراخوانی متدها با این شیوه اندکی مشکل به نظر می رسد که می توان آن را با استفاده از هدایتگر using برطرف نمود. به عنوان نمونه:

```
using System;
namespace USBCSharp
{
    class Welcome
    {
        static void Main()
        {
            Console.WriteLine("Welcome to USB C# Course!");
        }
    }
}
```

همانگونه که مشاهده می شود در این حالت دیگر نیازی به ذکر نام فضای نام قبل از نام کلاس وجود ندارد.

علاوه بر این از using می توان جهت ایجاد یک نام مستعار (alias) برای کلاس استفاده نمود که در این حالت باید بصورت زیر از آن استفاده گردد:

```
using  alias = class ;
```

به عنوان مثال:

```
using   TstC1= NS1.NS1_1.C1 ;
```

حال نحوهٔ اجرای متد tst2() بصورت زیر خواهد بود.

```
TstC1.tst2();
```

۲) قوانین نامگذاری شناسه ها

برای نامگذاری شناسه ها (متغیرها، متدها، کلاسها ، فضاهای نام و غیره) مجموعه قواعدی وجود دارد که بهتر است از آنها استفاده گردد. این قواعد به شرح زیر می باشند:

- طول نام شناسه ها محدود به ۲۵۵ کارکتر می باشد.
- نام شناسه ها باید با یک حرف شروع شود، اما بقیه کارکترها می توانند حرف/ عدد و یا کارکتر _ باشند.

علاوه بر دو قاعده فوق که باید رعایت شوند، مجموعه توصیه هایی جهت انتخاب نام مناسب برای شناسه ها وجود دارد که به قرار زیر می باشند:

- **فضای نام:**
  - نام انتخابی برای فضای نام بهتر است نام سازمان و یا شرکتی باشد که آن را می نویسد.
  - بهتر است که حروف اول کلیه کلمات نام انتخاب شده، حروف بزرگ باشند.

  مثال)CsharpTech، IranSystem و غیره

- **کلاس:**
  - نام انتخابی برای کلاس بهتر است نامی گویا جهت نمایاندن عملکرد کلاس باشد.
  - بهتر است که حروف اول کلیه کلمات نام انتخاب شده، حروف بزرگ باشند.

  مثال) Console، MyClass، FirstCsharpClass و غیره

- **متد:**
  - نام انتخابی برای متد بهتر است فعلی انتخاب گردد که نمایانگر عملکرد آن باشد.
  - بهتر است که حروف اول کلیه کلمات نام انتخاب شده، حروف بزرگ باشند.

  مثال) Write، WriteLine، ReadLine، GetValue و غیره

- **پارامترهای متد:**
  - این بخش مشابه کلاس و فضای نام می باشد.
- **متغیرها:**
  - نام انتخابی بهتر است نامی متناسب با عملکرد متغیر باشد.
  - بهتر است حرف اول اولین کلمه متغیر، کوچک اما حروف اول بقیهٔ کلمات بزرگ درنظر گرفته شود.

  مثال) myVar، prime، localVariable

فصل دوم: مقدمه ای بر C#

۳) متغیرها و ثابت ها

برای تعریف متغیرها در برنامه از گرامر زیر پیروی می شود:

```
<type>      نام متغیر ;
```

مثال)

```
int   myInt ;
bool flag;
```

جهت تعریف مقادیر ثابت در برنامه از گرامر زیر استفاده می گردد:

```
const  <type>  نام متغیر = مقدار ;
```

مثال)

```
const  int myConst = 4 ;
const  double  myPI = 3.1415 ;
```

۴) انواع اولیه

انواع اولیه در C# را به چهار گروه تقسیم می نماییم:

الف – انواع عددی صحیح

نام نوع علامت دار	معادل بدون علامت	اندازه بر حسب بایت
sbyte	byte	۱
short	ushort	۲
int	uint	۴
long	ulong	۸

مثال)

```
sbyte  mySignedByte = -125; // مقادیر مثبت و منفی
byte  myByte= 125U; // فقط مقادیر مثبت
short myShort = 31000;  // مقادیر مثبت و منفی
uint  myUnsignedInt = 65000U ; // فقط مقادیر مثبت
```

ب – نوع بولی

نوع بولی فقط دو مقدار ثابت true و false را می تواند در خود ذخیره نماید.

مثال)

```
bool  myFlag = true ;
bool  myCheck= false ;
```

ج – انواع عددی اعشاری

نام نوع	اندازه بر حسب بایت
float	۴
double	۸
decimal	۱۶

مثال)

```
float  myFloat= 3.456 f ;
```

توجه: برای مقداردهی متغیرهای نوع float حتما باید از پسوند f ویا F استفاده نمود در غیر اینصورت مقدار نسبت داده شده از نوع double تلقی شده و در تبدیل نوع از double به float خطا بوجود می آید.

```
double  myDouble = 3.14567 ;
double   yourDouble = 3.12323D ;
decimal  myDecimal =  23.4356m ;
```

توجه: مشابه نوع float برای نوع decimal نیز پسوند وجود دارد که m ویا M می باشد.

د- انواع کارکتری

انواع کارکتری حاوی دو نوع می باشند:

- نوع char: برای ذخیره یک کارکتر یونی کد به کار می رود و اندازه آن ۲ بایت می باشد.
- نوع string: برای ذخیره مجموعه ای از کارکترهای یونی کد به کار می رود.

مثال)

```
char  myChar = 'A';
string  myName = 'Nik';
```

با اینکه هر زبانی انواع مختص به خود را دارد اما کلیه این انواعی که در زبانهای مجموعه .Net بکار می روند، در واقع یک نام دیگر برای انواعی هستند که در مجموعه .Net از پیش تعریف شده و وجود دارند.

در جدول زیر هر کدام از انواع زبان C# و معادل تعریف شدهٔ آن در .Net لیست شده اند:

نوع بکار رفته در C#	معادل موجود در .Net
Sbyte	System.Sbyte
Byte	System.Byte
Short	System.Int16
Ushort	System.UInt16
Int	System.Int32
Uint	System.UInt32
Long	System.Int64
Ulong	System.UInt64
Bool	System.Boolean
Float	System.Single
Double	System.Double
Decimal	System.Decimal
Char	System.Char
String	System.String

می توان در تعریف متغیرها به جای استفاده از انواع موجود در زبان از همان انواع تعریف شده در .Net بهره برد. به عنوان مثال:

```
System.Int32   myInt = 43564 ;
System.Single  myFloat = 3.14f ;
System.Boolean myBoolean = true ;
```

۵) عملگرها

عملگرهای موجود در C# اغلب همان عملگرهای زبان C می باشند.

در زیر عملگرهای موجود در C# با توجه به اولویت آنها ذکر شده اند و در ادامه به بعضی از مهمترین آنها اشاره می گردد.

نام عملگر(ها)	عملگر(ها)	کاهش اولویت
اصلی	new, sizeof , typeof ,checked, unchecked	
تکی	+ - ! ~ ++ -- cast	
ضربی	* / %	
جمعی	+ -	
شیفت	<< >>	
رابطه ای	< > <= >=	
تساوی	== !=	
AND منطقی	&	
XOR منطقی	^	
OR منطقی	|	
AND شرطی	&&	
OR شرطی	||	
شرطی سه تایی	?	
انتساب	= *= += -= /= %= <<= >>= &= |= ^=	

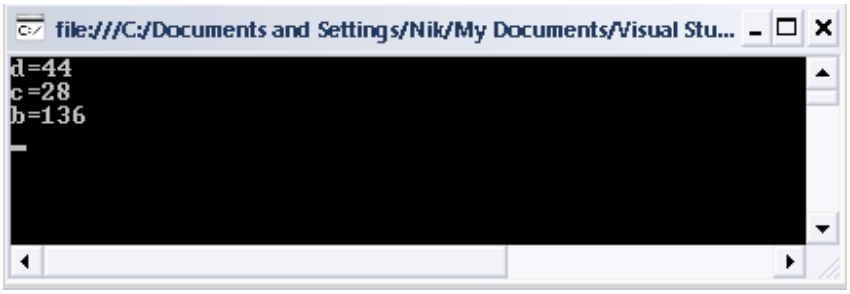
عملکرد کلیه عملگرها با C یکسان می باشد، تنها دو عملگر checked و unchecked هستند که که معادلی برای آنها در C وجود ندارد که در بخش مدیریت خطاها به بررسی آنها پرداخته خواهد شد.

کاربرد بقیه عملگرها را با ذکر چند مثال یادآوری می کنیم:

مثال۱) با توجه به اولویت عملگرها، خروجی برنامه زیر را محاسبه نمایید.

```
using System;
namespace Operator
{
    class Program
    {
        static void Main()
        {
            byte a = 17, b = 49, c = 60, d = 0;
            d=(byte)(( a & b ) ^ ( c | a ));
            Console.WriteLine("d=" + d);
            c -= (byte)(d > b ? a - b : b - a);
            Console.WriteLine("c=" + c);
            b <<= 3;
            Console.WriteLine("b=" + b);
            Console.ReadLine();
        }
     }
}
```

که خروجی آن پس از اجرا طبق شکل زیر خواهد بود:



آیا مقدار b صحیح می باشد؟

اگر نوع کلیه متغیرها بجای byte ، sbyte قرار داده شود، دراینصورت خروجی برنامه فوق را تعیین نمایید.

۶) دستورات کنسول

در کلاس کنسول مجموعه متدهایی جهت کار با محیط کنسول گنجانده شده است که در زیر به پاره ای از این متدها اشاره می شود:

متد Write: این متد جهت چاپ مقدار متغیرها و یا چاپ مقادیر ثابت بکار می رود.

متد WriteLine: عملکرد این متد مشابه متد Write می باشد با این تفاوت که پس از پایان عمل چاپ مکان نما را با ابتدای خط بعدی منتقل می نماید.

متد ReadLine: جهت اخذ یک عبارت رشته ای از ورودی بکار می رود.

متد Clear: جهت پاک کردن محتوای کنسول بکار می رود.

متد Beep: جهت به صدا در آوردن بوق سیستم بکار می رود.

متد SetCursorPosition: جهت انتقال مکان نما به سطر و ستون خاصی مورد استفاده واقع می شود.

متد Write / WriteLine:

از این متد جهت چاپ مقادیر متغیرها و مقادیر ثابت استفاده می شود و کاربرد آن بصورت زیر می باشد. این متد معمولا بصورتهای زیر مورد استفاده قرار می گیرد:

- **بدون فرمت بندی:** در این حالت نام متغیر بعنوان پارامتر متد ذکر می شود.

```
int x = 320;
float f = 32.43 f  ;
Console.WriteLine( x );
Console.WriteLine( f );
```

- **فرمت بندی** : در این حالت می توان چندین متغیر را همزمان با یک عبارت WriteLine چاپ نمود.

```
int x = 320;
float f = 32.43 f ;
Console.WriteLine ( " X= {0}      Y= {1}", x , f);
```

در مثال فوق عبارت بین علامات کوتیشن عینا به خروجی منتقل می گردد، فقط به جای عبارات فرمت بندی شده (آنهایی که مابین علامت های }و{ واقع شده اند) معادل آنها از سمت راست عبارت جایگزین می گردد که اولین عبارت سمت راست(X) به جای {0} و دومین عبارت (Y) به جای {1} قرار داده می شود.

خروجی بصورت X = 320 Y = 32.43 خواهد بود.

تذکر: از کارکترهای کنترلی موجود در زبان C مانند \n و \t می توان در بخش فرمت دهی استفاده نمود.

```
int x = 320;
float f = 32.43 f ;
Console.WriteLine ("  X= {0} \n Y= {1}", x , f);
```

در این حالت خروجی در دو خط جداگانه چاپ می شود.

متد ReadLine: از این متد جهت اخذ مقادیر بصورت رشته ای از ورودی استفاده می شود. در واقع مقدار برگشتی این متد یک رشته می باشد.

```
string  name;
name= Console.ReadLine();
```

متد SetWindowSize: از این متد جهت تنظیم اندازه پنجره خروجی استفاده می گردد که دو پارامتر ورودی آن به ترتیب عبارتند از اندازه های افقی و عمودی پنجره.

خاصیت Title: از این خاصیت جهت تعیین عنوان پنجره استفاده می کردد.

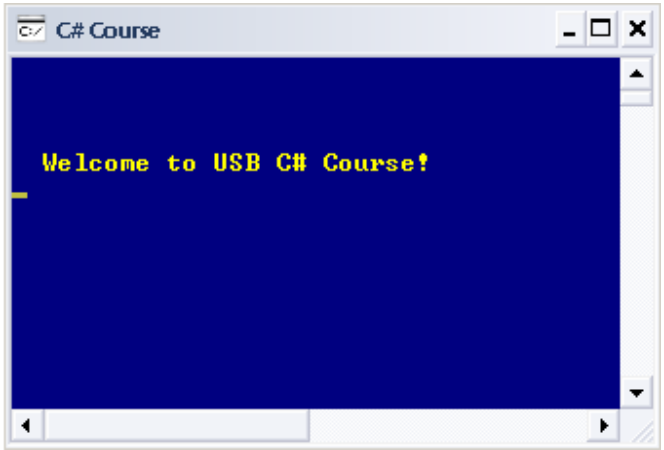
خاصیت BackgroundColor: از این خاصیت جهت تعیین رنگ زمینه پنجره استفاده می کردد.

خاصیت ForegroundColor: از این خاصیت جهت تعیین رنگ قلم استفاده می کردد.

برنامه ای که چگونگی استفاده از متدها و خواص کلاس کنسول را نشان می‌دهد، در زیر آورده شده است:

```
using System;
namespace Operator
{
    class Program
    {
        static void Main()
        {
            Console.Title = "C# Course";
            Console.BackgroundColor = ConsoleColor.DarkBlue;
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.Clear();
            Console.SetCursorPosition(2, 4);
            Console.WriteLine("Welcome to USB C# Course!");
            Console.ReadLine();
        }
     }
}
```

در شکل زیر خروجی حاصل از برنامه فوق نشان داده شده است:



فصل دوم: مقدمه ای بر C#

تبدیل انواع در C#

- جهت تبدیل انواع عددی به همدیگر باید توجه کرد که همواره انواع کوچکتر به انواع بزرگتر بصورت ضمنی تبدیل می شوند. به عنوان مثال می توان مقدار یک متغیر از نوع short را در متغیری از نوع int و یا long ذخیره نمود.

```
short  myShort = 234;
int myInt = myShort;
```

- اما عکس این تبدیل صحیح نمی باشد و چنین عملی در زمان کامپایل با شکست مواجه می شود. برای انجام عکس تبدیل فوق باید این عمل را بصورت صریح انجام داد که با این وجود نیز چون نوع بزرگتری در نوع کوچکتر قرار داده می شود، ممکن است با از دست دادن اطلاعات مواجه شویم.

```
int x = 897000;
short   myS ;
myS = ( short ) x;
Console.WriteLine ( myS );
```

که خروجی مجموعه دستورات فوق مقدار نامعتبر 20504- می باشد که به خاطر سرریز اتفاق افتاده است.

- اما جهت تبدیل مقادیر رشته ای به معادل صحیح آنها ویا تبدیل از هر نوعی به نوع دیگر از روشهای زیر استفاده می شود:

  - هر کدام از انواع اولیه موجود، دارای متدی به نام Parse می باشند که جهت تبدیل نوع رشته ای به نوع اولیه موردنظر بکار می رود. به مثال زیر دقت نمایید:

```
string  myVal = Console.ReadLine(); // خروجی این مرحله یک رشته عددی است
int  myInt = int.Parse ( myVal ); // جهت تبدیل به نوع صحیح ۴ بایتی
float myFloat = float.Parse( myVal ); // جهت تبدیل به نوع اعشاری
```

  - راه حل دیگر که برای هرگونه تبدیل نوعی می توان از آن بهره برد، استفاده از متدهای موجود در کلاس Convert می باشد که جهت انجام کلیه تبدیلات مناسب می باشند.

```
string  myVal= Console.ReadLine(); // خروجی این مرحله یک رشته عددی است

int  myInt = Convert.ToInt 32 (  myVal );
float myFloat = Convert.ToSingle ( myVal );
```

۷) کامپایل و اجرای برنامه های C#

جهت کامپایل و اجرای برنامه های نوشته شده به زبان C# باید به نکات زیر توجه نمود:

۱. ابتدا باید نسخه مناسبی از .Net Framework را بر روی سیستم نصب نموده سپس به روش زیر برنامه را کامپایل و اجرا نماییم. نسخه های .Net Framework از روی سایت ماکروسافت قابل دانلود می باشد. با توجه به سرعت پایین دانلود، بر روی CD درس نیز قرار داده شده است.

۲. برنامه را با یک ویرایشگر مناسب مانند Notepad تایپ نموده و با پسوند .cs ذخیره می نماییم.

۳. در سطر فرمان به کمک کامپایلر زبان C# آنرا کامپایل می نماییم که در صورت عدم وجود خطا در برنامه، فایل EXE آن که یک اسمبلی است، تولید می شود.

```
C:\مسیر فایل> csc   filename
```

فصل سوم: ساختارهای تصمیم و حلقه های تکرار در C#

با توجه به اینکه ساختارهای تصمیم و حلقه های تکرار C# با C یکسان می باشد، در اینجا بصورت خلاصه فقط به معرفی ساختارهای تصمیم و حلقه های تکرار پرداخته و به تفاوت آنها با ساختارهای متناظر در C با ذکر چند مثال بسنده می کنیم.

الف) ساختارهای تصمیم

این ساختارها دارای گرامری مشابه نظیر آنها در C می باشند. در این قسمت ابتدا ساختار سلسله مراتبی if-else و سپس ساختار switch را معرفی می نماییم.

ساختار تصمیم if:

گرامر این ساختار بصورت زیر می باشد:

```
if ( Cond-1 )
{
     Statement-1
}
else if ( Cond-2 )
{
     Statement-2

}
…
else
{
     Statement-n

}
```

با توجه به ساختار فوق هنگام اجرا، ابتدا Cond-1 چک می شود، اگر شرط برقرار باشد مجموعه دستورات Statement-1 اجرا می گردد و اگر چنانچه شرط مورد نظر نادرست باشد، شرط Cond-2 چک می شود که درصورت درستی شرط مجموعه دستورات Statement-2 اجرا می گردد. در غیر اینصورت روند مورد نظر ادامه پیدا می کند تا به شرطی درست در ساختار برخورد نماییم. اگر هیچ شرطی برقرار نباشد، مجموعه دستورات بخش else (Statement-n) اجرا می گردد.

تذکر۱: بخش else اختیاری بوده و درصورت عدم وجود آن و عدم برقراری هیچکدام از شرطهای درون if ، دستور بعد از if اجرا خواهد شد.

تذکر۲: برخلاف C نوع هر کدام از شرطها صرفا باید از نوع bool باشند. یعنی مقادیر int و غیره را نمی توان مشابه C در شرط درون if قرار داد.

مثال۱)مجموعه دستورات زیر را درنظر بگیرید:

```
int flag=1 , y=5 ;
if ( flag)
  Console.Write(y);
else
  Console.Write(-y);
```

فصل سوم: ساختارهای تصمیم و حلقه های تکرار در C#

مجموعه دستورات فوق به درستی کامپایل نخواهد شد، زیرا مقدار درون شرط باید از نوع bool باشد در صورتیکه مقدار درون شرط عدد صحیح یک می باشد.

می توان مجموعه دستورات فوق را بصورت زیر تغیر داد تا بدرستی کامپایل و اجرا گردند:

```
int flag=1 , y=5 ;
if ( flag==1 )
  Console.Write(y);
else
  Console.Write(-y)
```

ساختار تصمیم switch:

این ساختار یکی دیگر از ساختارهای تصمیم گیری است که رفتار و عملکرد آن مشابه ساختار if می باشد. زمانی از این ساختار استفاده می گردد که بخواهیم مقدار متغیری از نوع **بولی**، **عددی صحیح**، **کارکتری** و یا **رشته ای** را با **مجموعه ای از مقادیر ثابت**، فقط از نظر مساوی بودن با همدیگر مقایسه نماییم. در واقع شرط مقایسه فقط برقراری تساوی می باشد و شرطهایی از قبیل کوچکتر، بزرگتر و غیره را نمی توان در این ساختار استفاده نمود.

گرامر این ساختار بصورت زیر می باشد:

```
switch (متغیر )
{
 case مقدارثابت۱:
      Statements1
      break;
case مقدارثابت۲:
      Statements2
      break;
case مقدارثابت۳:
      Statements3
      break;
…
default:
      Statement-n
      break;
}
```

در این حالت مقدار متغیر درون switch به ترتیب با مقادیر ثابت از بالا به پایین چک می شود، اگر مقدار متغیر با یکی از مقادیر ثابت برابر باشد، مجموعه دستورات متعلق به آن case اجرا می گردد و سپس توسط دستور break کنترل به بیرون switch منتقل می گردد.

اگر مقدار متغیر با هیچکدام از مقادیر ثابت یکسان نباشد، مجموعه دستورات بخش default اجرا شده و به کمک دستور break این بخش کنترل به بیرون switch منتقل می شود.

تذکر۱: بعد از هر دستور case و بعد از default حتما باید دستور break وجود داشته باشد.

تذکر۲: نوع متغیر ذکر شده در switch باید یکی از انواع بولی، صحیح، کارکتر و رشته ای باشد.

تذکر۳: اگر بخواهیم چند شرط را با همدیگر یای منطقی نماییم، فقط کافی است که case های متناظر با آنها را پشت سرهم ذکر نماییم.

```
class Program
{
    static void Main()
    {
        string s=Console.ReadLine();
        switch (s)
        {
            case "USB":
            case "usb":
                Console.WriteLine("University of Sistan & Balouchistan");
                break;
            case "IUST":
            case "iust":
                Console.WriteLine("Iran University of Science & Technology");
                break;
            case "UT":
            case "ut":
                Console.WriteLine("University of Tehran");
                break;
            default:
                Console.WriteLine("Others University");
                break;
        }

        Console.ReadLine();
    }
}
```

مساله فوق را با استفاده از توابع رشته ای می توان حل کرد که در آن ابتدا تابع ToLower کلیه حروف رشته را به کوچک تبدیل می نماید، سپس عمل مقایسه انجام می گیرد.

```
class Program
{
    static void Main()
    {
        string s=Console.ReadLine();
        switch (s.ToLower())
        {
            case "usb":
                Console.WriteLine("University of Sistan & Balouchistan");
                break;
            case "iust":
                Console.WriteLine("Iran University of Science & Technology");
                break;
            case "ut":
                Console.WriteLine("University of Tehran");
                break;
            default:
                Console.WriteLine("Others University");
                break;
        }

        Console.ReadLine();
    }
}
```

تذکر۴: استفاده از چند مقدار در یک case با بکار بردن عملگر کاما مجاز نمی باشد. مانند:

```
case 2,3:
     Console.Write(" 2,3");
     break;
```

تذکر۵: یکی از کاربردهای ساختار switch در طراحی منوهای ساده می باشد که با توجه به وارد نمودن مقداری توسط کاربر، عملی توسط برنامه انجام می پذیرد.

ب) حلقه های تکرار

از حلقه های تکرار برای جهت تکرار انجام دستورالعملها استفاده می شود. که با توجه به ماهیت مساله می توان یکی از انواع حلقه ها را جهت انجام آن برگزید.

حلقه تکرار for: از این ساختار زمانی استفاده می شود که تعداد دفعات تکرار مجموعه عملیات مشخص باشد.

توجه: در مثالهای زیر فقط بدنهٔ برنامهٔ اصلی نوشته شده است.

مثال۱) محاسبه مجموع اعداد یک تا n

```
int s=0 , i;
for ( i=1 ; i<=n ; i++)
     s+= i;
```

مثال۲) با دریافت مقدار n از ورودی، مقدار سری زیر را محاسبه نمایید.

$$s=\sum_{i=1}^{n}\frac{2^i}{i!}$$

```
float s=0f;
int i,  p1=1, p2=1 ;
for ( i=1 ; i<=n ; i++)
  {
     p1*= 2 ;
     p2*=i ;
     s+=  p1 / p2 ;
  }
```

مثال ۳) تعداد اعداد تام (و یا اول) سه رقمی (۴ یا ۵ رقمی) را محاسبه نمایید.(در این حالت بازهٔ اعداد مورد بررسی مشخص می باشد.)

```
int count=0 ;
int i;
for ( i=100; i<1000 ; i++)
 {
   int j,  s=1;
     for( j=2; j<=i/2 ;j++)
          if (i % j ==0 )
               s += j ;
   if (i == s)
    ++count ;
 }
```

مثال ۴) محاسبه جملهٔ nام سری فیبوناچی

```
if (n ==1 || n==2 )
  Console.WriteLine(  "1" );
else
  {
    int f=1  , f2=1 ,i ,f;
    for ( i=2 ; i<n ; i++)
     {
       f= f1 + f2 ;
       f1 = f2 ;
       f2=f;
     }
    Console.WriteLine ( f ) ;
  }
```

حلقه تکرار while: از این ساختار زمانی استفاده می شود که تعداد دفعات تکرار مجموعه عملیات مشخص نباشد.

مثال۵) محاسبه مجموع ارقام عدد طبیعی n

```
int n;
n= int.Parse ( Console.ReadLine() );
int t= n , s = 0;
while ( t>0 )
 {
   S += t%10 ;
   t /= 10 ;
 }
 Console.Write ( "Sum of Digits{0}={1}" , n ,s );
```

حلقه تکرار do-while: از این ساختار زمانی استفاده می شود که تعداد دفعات تکرار مجموعه عملیات مشخص نباشد و حلقه حداقل یکبار نیاز به اجرا داشته باشند.

مثال۶) محاسبه تعداد ارقام عدد طبیعی n

```
int n;
n= int.Parse ( Console.ReadLine() );
int t= n , count = 0;
do{
   count++  ;
   t /= 10 ;
 } while ( t>0 );
 Console.Write ( "Number of Digits {0}={1}", n ,count );
```

مثال۷) تعیین بزرگترین عدد چهاررقمی که بر مجموع ارقام فردش بخش پذیر باشد.

```
int i= 9999 , s= 0 ,t;
while ( i>= 1000)
{
t=i ;
while ( t>0)
 {
   if ( (t%10)%2 == 1)
     s += t%10 ;
   t /= 10 ;
 }
 if ( i%s == 0)
   break ;
 i--;
}
Console.Write(i);
```

تذکر۱: عملکرد `break` و `continue` مشابه زبان C می باشد. با اجرای دستور `break` کنترل به بیرون حلقه جاری منتقل می شود و با اجرای دستور `continue` کنترل به ابتدای حلقه جاری منتقل می شود.

تذکر۲: دستور `continue` فقط در حلقه ها کاربرد دارد اما از دستور `break` هم در حلقه ها و هم در ساختار `switch` می توان استفاده نمود.

نمونه سوالات فصل ساختارهای شرطی و حلقه های تکرار

۱. مجموع کلیه اعداد چهار رقمی بدون رقم صفر را محاسبه نمایید که بر ۷ بخش پذیر باشند.
۲. تبدیل عددی از مبنای ده به مبنای a .
۳. تبدیل عددی از مبنای a به مبنای ده .
۴. محاسبه کلیه زیرمجموعه های مجموعه n عضوی، با فرض اینکه حداکثر مقدار n برابر ۸ باشد.
۵. تعداد اعداد تام درون عدد داده شدهٔ n را محاسبه نمایید.

مثلا اگر n= 628 باشد، اعداد تام عبارتند از ۶ (تک رقمی) و ۲۸ (دو رقمی) و عدد تام سه رقمی نداریم.

وما توفیق الا بالله العلی العظیم

۸۵/۷/۱۶

کار در کارگاه ۱: نحوه نصب Net Framework. و چگونگی کامپایل و اجرای یک برنامه نمونه در C#

در این بخش نحوهٔ نصب .Net Framework و چگونگی ایجاد و اجرای برنامه های C# همراه با مثال شرح داده می شود.

نحوه نصب .Net Framework

بر روی CD دو نسخه از .Net Framework قرار داده شده است که می توانید هر یکی از این دو نسخه را نصب نمایید(نسخه های ۲ و ۱.۱). برای این کار ممکن است که لازم باشد، ابتدا Windows Installer را نصب نمایید که کلیه این فایلها درون دایرکتوری Dot Net Framework گنجانده شده اند.

پس از نصب، فایلهای نصب شده در دایرکتوری Microsoft.Net درون دایرکتوری ویندوز قرار خواهند گرفت که از جمله این فایلها می توان به فایل csc.exe که کامپایلر زبان C# است، اشاره نمود.

ایجاد اولین برنامه C# و نحوه کامپایل و اجرای آن

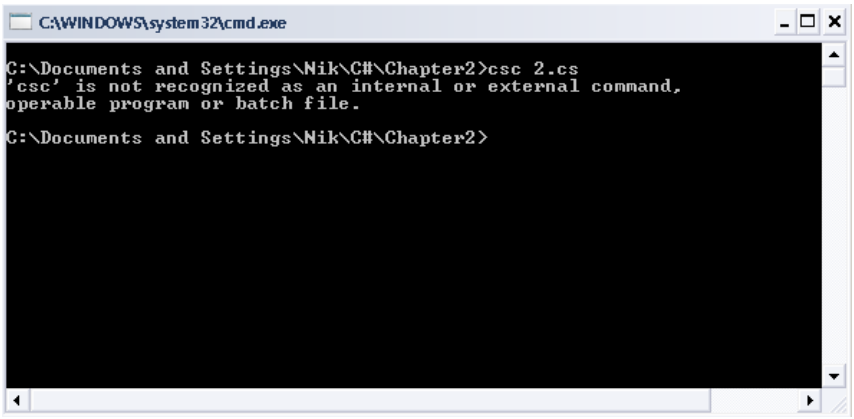
جهت ایجاد نمودن یک برنامه C# می توانید از یک ویرایشگر مناسب مانند Notepad استفاده نمایید. برای این کار برنامهٔ زیر را در محیط Notepad تایپ نموده و سپس در مسیر C:\C#\Chapter#1\ زیر با نام First.cs ذخیره نمایید. (ابتدا مسیر فوق را ایجاد نمایید.)

```
using System;
class Hello{
          static void Main()
            {
             Console.WriteLine("USB C# Course!");
            }
           }
```

اگر بخواهیم برنامه را کامپایل نماییم باید بصورت زیر عمل کنیم:

```
C:\C#\Chapter#1> csc  First.cs
```

که پس از اجرای دستور با خطایی مبنی بر اینکه csc شناخته شده نیست(شکل زیر) مواجه خواهیم شد که جهت رفع این مشکل باید مسیر فایل csc را به متتغیر محیطی path ویندوز اضافه نمود.



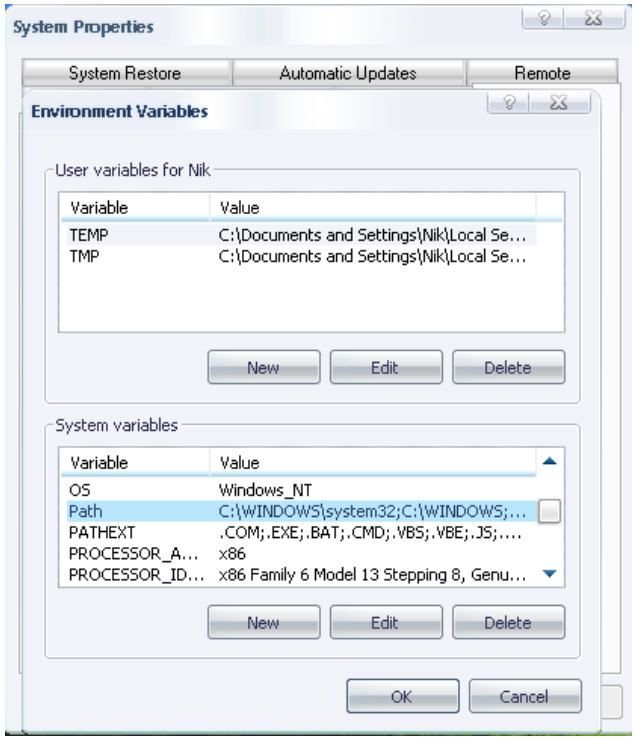
برای انجام این کار ابتدا مسیر فایل csc را پیدا می کنیم که بصورت زیر می باشد(با انجام یک جستجوی ساده بر روی درایو ویندوز براحتی می توان این مسیر را یافت).

```
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\
```

که این مسیر پس از نصب .Net framework v2.0 ایجاد می گردد.

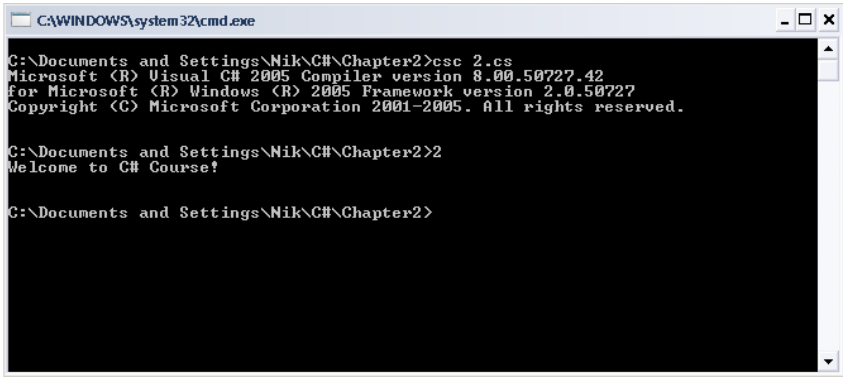
برای اضافه نمودن این مسیر به متغیرهای محیطی ویندوز بصورت زیر عمل می نماییم:

- بر روی My Computer کلیک راست می کنیم.
- از این بخش گزینه properties و از آن گزینهٔ Advanced را انتخاب می نماییم.
- از بخش Advanced گزینه Environment Variables را انتخاب می نماییم.
- سپس پنجره ای مشابه شکل زیر باز می شود:



در این بخش با انتخاب Path و کلیک بر روی Edit می توانیم مسیر فوق را به path اضافه نماییم. برای این منظور به انتهای جمله یک ; اضافه نموده سپس مسیر فوق را پس از ; Paste می نماییم. بعد از آن با کلیک نمودن بر روی OK های متوالی این مسیر به متغیرهای محیطی افزوده شده و در هر مسیری از بخش cmd می توان به فایلهای درون این بخش دسترسی داشت.

حال اگر این بار دستورات کامپایل اجرا گردد، دیگر پیام خطای قبلی تکرار نخواهد شد.

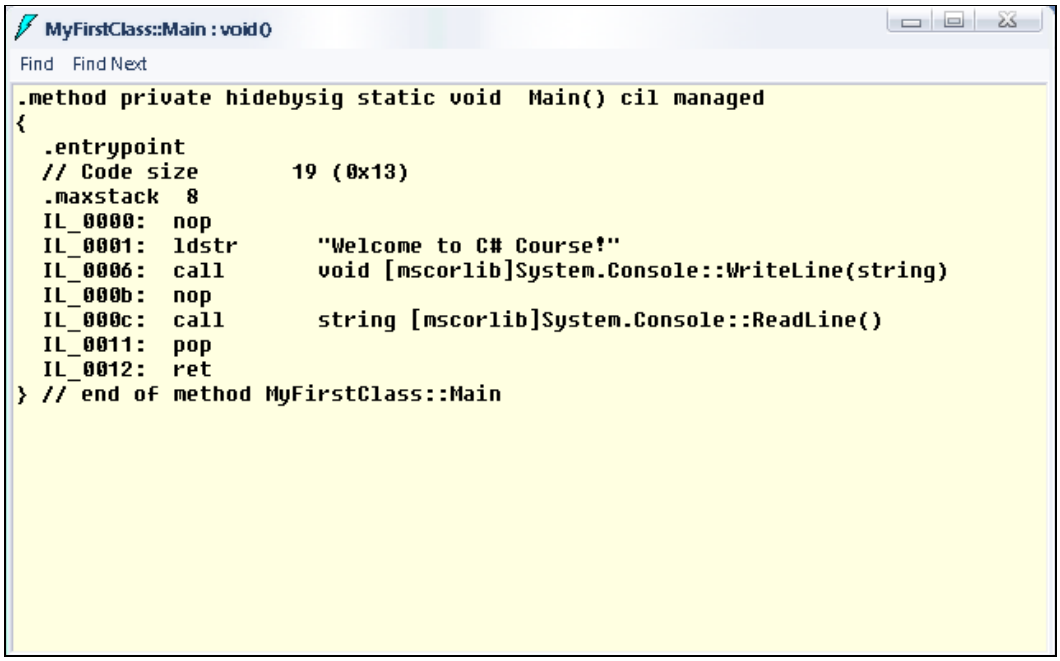


که نمونه کامپایل و اجرای آن در شکل فوق مشاهده می گردد.

- حال برنامه کامپایل شده و فایل EXE آن بر روی همین درایو و با نام 2.EXE ایجاد شده است که جهت نمایش کدهای IL آن باید از نرم افزار ILDASM استفاده نماییم که این نرم افزار نیز بر روی CD درس موجود می باشد.

ابتدا ابزار موردنظر را اجرا نموده، سپس فایل 2.EXE را به درون آن Drag نمایید. پس از این مرحله محیط نرم افزار مشابه شکل زیر خواهد بود.

- حال جهت نمایش کدهای IL بر روی Main کلیک می نماییم. که کدهای IL مربوطه مشابه شکل زیر می باشد.



تمرین

۱. حال برنامه زیر را کامپایل و اجرا نموده ، سپس کدهای IL آن را مشاهده نمایید.

```
using System;
class tst{
        static void Main()
        {
              int first, second, sum;
              first= int.Parse(Console.ReadLine());
              second= int.Parse(Console.ReadLine());
              sum = first + second ;
              Console.WriteLine (" Sum= " + sum);
              Console.ReadLine();
        }
}
```

۲. برنامه فوق را با استفاده از متدهای کلاس Convert بازنویسی نموده، سپس کدهای آن را مشاهده نمایید.

۳. برنامه را دوباره بگونه ای بازنویسی نمایید که از انواع پایه .Net استفاده نماید، مثلا بجای استفاده از int از System.Int32 استفاده نمایید. سپس کدهای IL آن را مشاهده نمایید، آیا تفاوتی در برنامه ایجاد شده است؟

۴. برنامه ای بنویسید که در آن از تبدیل انواع ضمنی و صریح استفاده شود، سپس سعی نمایید که یک نوع بزرگتر را به نوع کوچکتر بصورت صریح تبدیل نمایید.

- ✓ در این تبدیل در چه مواقعی مقدار تبدیل شده صحیح نمی باشد؟
- ✓ آیا می توان انواع بدون علامت را به علامت دار و یا بالعکس تبدیل نمود؟ در چه صورت؟
- ✓ در تبدیل انواع اعشاری به صحیح چه اتفاقی می افتد؟
- ✓ چه انواعی را می توان به نوع بولی تبدیل نمود؟

راهنمای کار با #Develop

نسخه های متفاوتی از کامپایلر SharpDevelop وجود دارد که می توان از هر یکی از آنها جهت ایجاد و کامپایل برنامه های C# استفاده نمود. سه نسخه از کامپایلر مزبور برروی CD گنجانده شده است که تفاوت چندانی با همدیگر ندارند فقط ممکن است که بعضی از نسخه ها مجموعه بیشتری از کلاسها را شامل باشند.

به عنوان مثال نسخه 1.0.3 بعضی از متدهای کلاس کنسول را دارا نمی باشد در حالیکه نسخه های جدیدتر مانند 2.2.0 شامل کلیه این متدها می باشند.

در زیر به مراحل نصب کامپایلر و ایجاد و کامپایل چند نمونه برنامه C# به کمک #Develop2.2.0 پرداخته می شود:

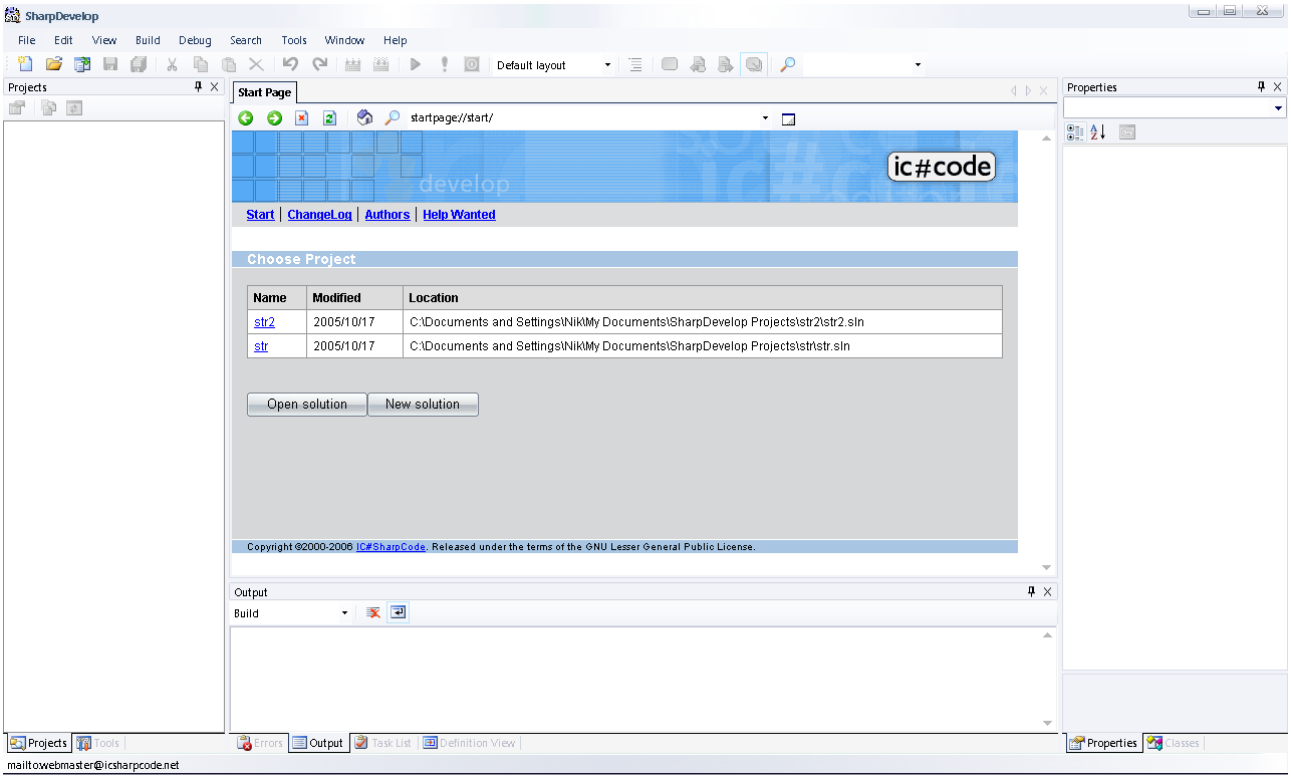
1. **نصب کامپایلر #Develop2.2.0**

قبل از نصب کامپایلر SharpDevelop باید .Net Framework2.0 را نصب نمود. بعد از آن جهت نصب کامپایلر فوق فایل SharpDevelop2_2.0.1.1710_Setup exe را اجرا نموده و مراحل نصب آنرا دنبال نمایید.

پس از تکمیل نصب برنامه در پوشه SharpDevelop درون program files قرار می گیرد و معمولاً shortcut آن نیز بر روی Desktop قرار داده می شود.

2. **اجرای IDE و نحوهٔ کار با آن**

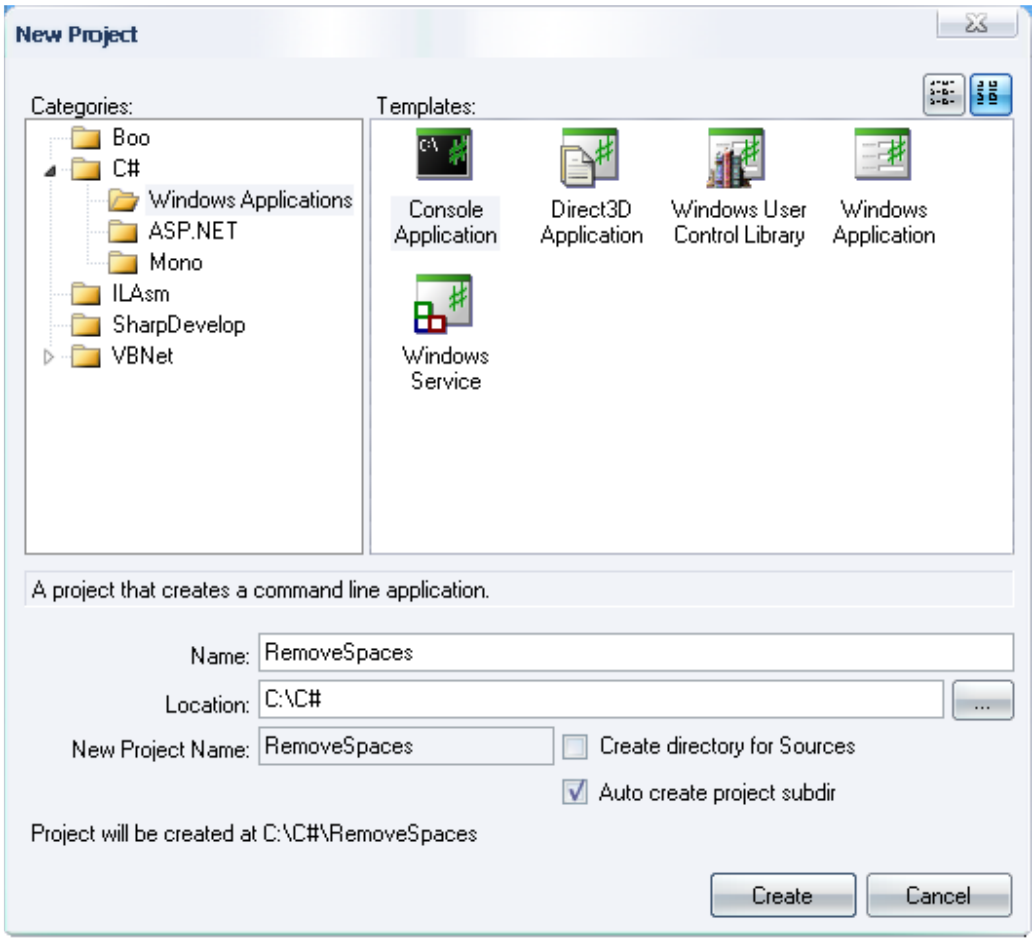
برای کار با IDE نصب شده، فایل SharpDevelop.exe را از روی desktop و یا از مسیر نصب شده در دایرکتوری ویندوز، اجرا نمایید. پس از اجرا صفحهٔ Start Page مشابه شکل زیر باز خواهد شد.



پس از اجرای IDE می توان یک Solution از قبل ایجاد شده را باز نمود و یا اینکه یک Solution جدید ایجاد نماییم. که در این بخش یک Solution جدید را با انتخاب New Solution ایجاد می نماییم. پس از ایجاد Solution جدید باید نوع پروژه را انتخاب نماییم.

راهنمای کار با #Develop

نوع پروژه را از بخش C#، سپس بخش Windows Applications انتخاب می نماییم. که فعلا نوع پروژه را Console Application انتخاب می نماییم. در بخش Name نام مناسبی برای Solution انتخاب می نماییم. سپس در بخش Location مکان ذخیره شدن Solution را انتخاب می نماییم. بعد از آن با کلیک نمودن دکمه Create پروژه موردنظر ایجاد می شود.



بعد از ایجاد پروژه، می توانیم در بخش متد Main می توانیم دستورات برنامه را تایپ نماییم.

مثال۱) برنامه ای بنویسید که رشته ای را از ورودی گرفته و کلیّهٔ فاصله های اضافی آنرا حذف نماید.

برای نوشتن برنامه ابتدا دستور اضافی درون متد Main را حذف می نماییم سپس دستورات مورد نیاز برای انجام این کار را در Main می نویسیم.

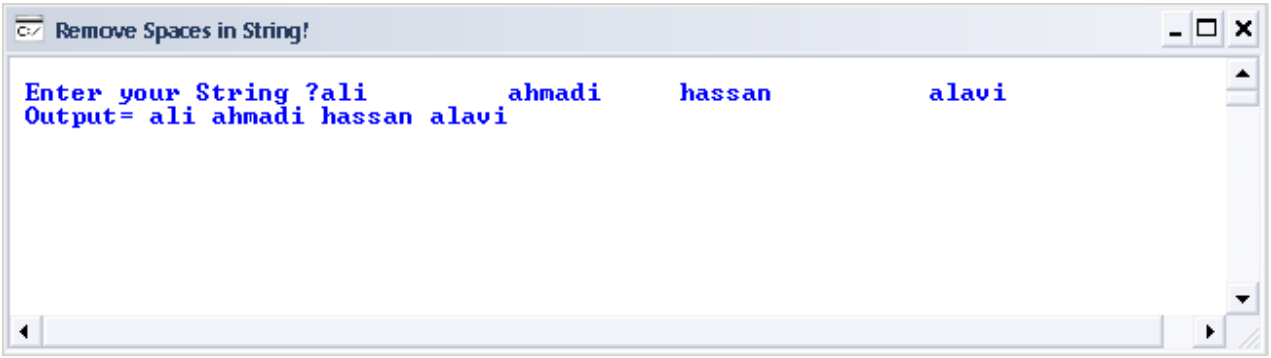
```
Console.BackgroundColor = ConsoleColor.White;
Console.ForegroundColor=ConsoleColor.Blue;
Console.Title= "Remove Spaces in String!";
Console.Clear();
Console.Write("\n Enter your String ?");
string  s=Console.ReadLine();
s=s.Trim();
int x=s.IndexOf("  ");
while  ( x != -1 )
   {
         s=s.Replace("  "," ");
         x=s.IndexOf("  ");
   }
Console.WriteLine(" Output= "+ s);
Console.ReadLine();
```

در حل مساله فوق با جایگزینی دو کارکتر فاصله با یک کارکتر فاصله، کلیه فاصله های اضافی رشته حذف شده اند.

۳. **کامپایل و اجرای برنامه**

جهت کامپایل و اجرای برنامه می توان از کلید F5 استفاده نمود و یا اینکه از منوی Debug گزینهٔ Run را انتخاب نماییم.

در این حالت درصورت وجود خطا در برنامه، خطاها به کاربر گزارش می شود که بعد از رفع کلیه خطاها برنامه اجرا می شود. نمونه اجرای برنامه در زیر مشاهده می شود:



۴. **اجرای خط به خط برنامه**

برای اجرای خط به خط برنامه ابتدا می توانیم به مکانهایی از برنامه Breakpoint اضافه نماییم که این عمل با رفتن بر روی دستور مورد نظر و زدن دکمه F7 صورت می پذیرد. بعد از آن با زدن دکمه F5 برنامه تا اولین Breakpoint اجرا می شود. سپس جهت اجرای دستورات بعدی می توان از کلیدهای F10 و F11 استفاده نمود.

برای برداشتن Breakpoint ها مشابه حالت قبل، باید مکان نما را بر روی دستوری که دارای Breakpoint است منتقل نمود، سپس دکمه F7 را فشار داد .

۵. **نمایش دستورات IL**

برای این منظور می توان از منوی Tools گزینه ILDASM را انتخاب نمود. که در اینصورت دستورات IL مربوط به متدهای پروژه را مشاهده نمود.

۶. **برای هر کدام از مسائل زیر یک برنامه بنویسید.**

۱. مجموع اعداد فرد سه رقمی
۲. مجموع ارقام مکانهای فرد عدد صحیح ورودی
۳. مجموع اعداد درون رشته ورودی
۴. کار با متدهای StartWith، EndWith، Trim، TrimEnd، TrimStart و غیره

وما توفیقی الا بالله

بلوچ زهی

۸۵/۷/۲۵

فصل چهارم: رشته ها

یکی از انواع داده ای بسیار مهم که بیشترین کاربرد را در برنامه ها دارد، نوع داده ای string (رشته) می باشد که هم در پردازش داده ها و هم در ذخیره سازی داده ها از اهمیت بسزایی برخوردار است.
یکی از علل اصلی در کاربرد فراوان این نوع داده ای این است که اغلب اطلاعات مورد استفاده برنامه ها از قبیل نام، نام خانوادگی، آدرس و غیره از نوع کارکتری می باشند و حتی انواع داده ای عددی که نیازی به انجام عملیات محاسباتی بر روی آنها نیست، نیز بصورت رشته ای مورد استفاده واقع می شوند.

تعریف متغیرهای نوع رشته ای در برنامه

نحوهٔ تعریف این نوع متغیرها، مشابه تعریف بقیه انواع داده ها در C# می باشد و بر عکس زبان C در اینجا رشته را بصورت آرایه ای از کارکترها که به کارکتر \0 و یا null ختم می شود، درنظر نمی گیرند.

```
string  نام متغیرها ;
```

مثال ۱)

```
string  FirstStr = "USB C# Course!";
string  SecondStr = "Advanced Programming";
```

نحوه دسترسی به کارکترهای رشته

برای دسترسی به کارکترهای درون رشته، از عملگر [] استفاده می شود. مثلا برای دسترسی به کارکتر iام رشته s از عبارت s[i] استفاده می شود.

تذکر۱: مقدار اولیه اندیس برابر صفر می باشد.

مثال۲) چاپ رشته بصورت کارکتر به کارکتر با حلقه for

```
int i;
string s = Console.ReadLine();
for (i=0 ; i< s.Length ;i++)
  Console.Write ( s[i] );
```

مثال۳) چاپ رشته بصورت کارکتر به کارکتر با حلقه foreach

```
string s = Console.ReadLine();
foreach (char ch in s)
  Console.Write ( ch );
```

تذکر۲:دسترسی به کارکترهای درون رشته بصورت فقط-خواندنی (read-only) می باشد و نمی توان مقادیر کارکترها را تغیر داد.

مثال۴) دستور زیر بدرستی کامپایل نمی شود.

```
s[i] = 'A' ;
```

فصل چهارم: رشته ها

برای کار با رشته ها، مجموعه ای از متدها فراهم شده است تا کار برنامه نویسان تا حد زیادی تسهیل گردد که در زیر به تعدادی از این متدها پرداخته می شود.

۱. الحاق نمودن رشته ها

دو روش جهت الحاق رشته ها به همدیگر وجود دارد که زیر به آنها اشاره می شود:

روش الف) در این روش از تابع Concat استفاده می شود که نحوهٔ استفاده از آن در زیر بیان شده است:

```
رشته جدید = string.Concat( string1, string2,string3,... )
```

مثال۵) پس از اجرای مجموعه دستورات زیر مقدار رشتهٔ s برابر چه مقداری خواهد بود؟

```
string s1= "C# Course", s2="USB";
string s=string.Concat( s1, s2 );
```

رشتهٔ S حاوی الحاق رشته های s1 وs2 خواهد بود که عبارت است از: C# CourseUSB

روش ب)در روش از عملگر + جهت الحاق رشته ها استفاده می شود. که چگونگی استفاده از آن در زیر ذکر می شود:

```
رشته جدید = string1 + string2 + string3 + ...
```

مثال۶) مقدار رشتهٔ s را پس از اجرای‌مجموعه دستورات زیر تعیین نمایید.

```
string s1= "Software" , s2= "IT";
string s = s1 + s2 ;
```

رشتهٔ S حاوی الحاق رشته های s1 وs2 خواهد بود که عبارت است از SoftwareIT که اگر بخواهیم بین دو کلمه فاصله وجود داشته باشد، می توانیم بصورت زیر عمل نماییم.

```
string s= s1 + " " + s2 ;
```

۲. متد Contain

به کمک این متد می توان تعیین نمود که آیا یک رشته حاوی یک زیر رشتهٔ مشخص می باشد یا خیر؟ که اگر زیر رشته در رشتهٔ موردنظر وجود داشته باشد نتیجهٔ تابع مقدار true و درغیر اینصورت برابر مقدار false خواهد بود. گرامر استفاده از متد فوق بصورت زیر می باشد:

```
bool   متغیر = رشته اصلی.Contain( زیر رشته ) ;
```

مثال۷) در مجموعه دستورات زیر مقدار متغیر flag برابر چند خواهد بود؟

```
string s1="USB C# Course", s2= "Course";
bool flag = s1.Contain( s2 );
```

در مثال فوق چون رشتهٔ s2 در رشتهٔ s1 وجود دارد، مقدار flag برابر true خواهد بود.

تذکر۳: تابع فوق نسبت به حروف کوچک و بزرگ حساس می باشد.

۳. متد Replace

با این متد می توان یک کارکتر از رشته را با یک کارکتر جدید ویا یک زیر رشته از رشته را با یک زیر رشته جدید جایگزین نمود.

الف)جایگزینی کارکتر

گرامر چگونگی انجام عمل جایگزینی در زیر آمده است:

```
رشته جدید = رشته اصلی.Replace ( کارکتر اصلی , کارکتر جدید );
```

در عبارت فوق کارکتر اصلی از رشتهٔ اصلی با کارکتر جدید جایگزین شده و نتیجه در رشته جدید قرار می گیرد .

تذکر۴: در این دستور رشته اصلی بدون تغیر باقی می ماند.

مثال۸)

```
string s1= "USB C# Course", s2= "Advanced Programming"
string s=s1.Replace ('B', 'A');
```

با اجرای این دستور، کلیه کارکترهای B درون رشته s1 با کارکتر A جایگزین شده و نتیجه به رشته s انتساب داده می شود. در نتیجه مقدار رشته s برابر "USA C# Course" خواهد شد.

ب) جایگزینی زیررشته

گرامر چگونگی انجام این کار در زیر آمده است:

```
رشته جدید = رشته اصلی.Replace ( زیر رشته اصلی , زیر رشته جدید );
```

در عبارت فوق زیررشته اصلی از رشتهٔ اصلی با زیررشته جدید جایگزین شده و نتیجه در رشته جدید قرار می گیرد .

تذکر۵: در این دستور رشته اصلی بدون تغیر باقی می ماند.

مثال۹)

```
string s1= "USB C# Course", s2= "Advanced Programming"
string s=s1.Replace ("C#", "CSharp");
```

در این حالت زیررشته C# درون s1 با زیررشته CSharp جایگزین شده و نتیجه به s نسبت داده می شود. در انتها مقدار رشته s برابر "USB CSharp Course" می شود.

۴. متد Insert

با این متد می توان زیر رشته ای را در مکان خاصی از رشته موجود درج نمود.

نحوه استفاده از این متد طبق گرامر زیر می باشد:

```
رشته جدید = رشته اصلی.Insert( اندیس , زیررشته );
```

با متد فوق در مکان مشخص شده در بخش اندیس، زیررشته داده شده درج می گردد و رشته تولید شده به بخش رشته جدید انتساب داده می شود.

تذکر۶:رشته اصلی بدون تغیر باقی می ماند.

مثال۱۰)

```
string s1="USB C# Course";
string s= s1.Insert( 4, "Basic " );
```

زیر رشته "Basic " در اندیس ۴ رشته s1 درج می شود و رشته ایجاد شده به s نسبت داده می شود، درنتیجه رشته s در انتها برابر "USB Basic C# Course" می شود.

تذکر۷: جهت اضافه نمودن زیررشته به ابتدای رشته موجود مقدار اندیس را برابر **صفر** قرار می دهیم.

تذکر۸: جهت اضافه نمودن زیررشته به انتهای رشته موجود مقدار اندیس را برابر **طول رشته** قرار می دهیم.

مثال۱۱)

```
string s1 = "CSharp";
string s2=s1.Insert( 0, "USB " );
```

در این حالت زیررشته "USB " به ابتدای رشته اضافه شده و مقدار رشته s2 برابر "USB CSharp" می شود.

```
string s=s2.Insert ( s2.Length , " Course") ;
```

با اجرای این دستور عبارت "Course" به انتهای رشته s2 اضافه می گردد که در انتها مقدار رشته s برابر "USB CSharp Course" می شود.

۵. **متد ToLower**

با استفاده از این متد می توان کلیه حروف موجود در رشته را به حروف کوچک معادل آنها تبدیل کرد.

مثال۱۲)

```
string s1= "ADVANCED C#" ;
string s= s1.ToLower();
```

مقدار رشته s برابر "advanced c#" می شود.

۶. **متد ToUpper**

با استفاده از این متد می توان کلیه حروف موجود در رشته را به حروف بزرگ معادل آنها تبدیل کرد.

مثال۱۳)

```
string s1= "basic c#" ;
string s= s1.ToUpper() ;
```

پس از اجرای دستور فوق مقدار رشته s برابر "BASIC C#" می شود.

۷. متد CompareTo

با این متد می توان دو رشته را با همدیگر مقایسه نمود. چنانچه دو رشته یکسان باشند مقدار نتیجه برابر **صفر** و درغیر اینصورت برابر **منفی یک** می باشد.

گرامر استفاده از این متد بصورت زیر می باشد:

```
int check  = رشته۱.CompareTo (رشته ۲ );
```

که رشته۱ و رشته ۲ را با همدیگر مقایسه می نماید و چنانچه دو رشته دقیقا با همدیگر یکسان باشند، مقدار عدد صحیح نتیجه برابر صفر و درغیراینصورت مقدار خروجی برابر یک (در حالتیکه رشته اولی از دومی بزرگتر باشد) و یا منفی یک (در حالتیکه رشته اولی از دومی کوچکتر باشد) خواهد شد.

تذکر۹: این تابع نسبت به حروف کوچک و بزرگ حساس می باشد.

مثال۱۴)

```
string  s1="USB Software", s2="USB Hardware";
int  x = s1.CompareTo ( s2 );
```

پس از اجرای دستورات فوق مقدار x برابر ۱ می باشد، چون دو رشته s1 و s2 یکسان نمی باشند.

۸. متد Equals

عملکرد این متد مشابه متد فوق می باشد با این تفاوت که خروجی تابع مقداری بولی می باشد که اگر دو رشته یکسان باشند، مقدار true و درغیر اینصورت مقدار fale در خروجی قرار می گیرد.

تذکر۱۰: این تابع نسبت به حروف کوچک و بزرگ حساس می باشد.

گرامر استفاده از تابع بصورت زیر می باشد:

```
bool  check = رشته۱.Equals (رشته ۲ );
```

مثال۱۵)

```
string  s1="USB Software", s2="USB Hardware";
bool x= s1.Equals ( s2 );
bool y=s1.Equals ( s1 );
```

پس از اجرای مجموعه دستورات فوق مقادیر متغیرهای x و y به ترتیب برابر false و true خواهد بود.

۹. متد Compare

با تابع مشابه دو تابع فوق می توان یکسان بودن دو رشته را بررسی نمود که گرامر استفاده از این تابع تا اندازه ای با دو تابع فوق متفاوت می باشد. همچنین در این تابع بصورت صریح می توان تعیین نمود که در بررسی یکسان بودن دو تابع، نسبت به حروف کوچک و یا بزرگ حساس باشد یا آنها را یکسان درنظر بگیرد.

گرامر استفاده از متد فوق در زیر آمده است:

```
int check = string.Compare ( رشته۱ , رشته۲ , flag);
```

تابع فوق رشته۱ و رشته۲ را باتوجه به فیلد flag با همدیگر مقایسه می نماید و چنانچه دو رشته یکسان باشند مقدار check برابر صفر و درغیراینصورت برابر یک و یا منفی یک خواهد بود. (مشابه CompareTo)

تذکر۱۱: اگر مقدار flag برابر true باشد، در مقایسه دو رشته حروف کوچک و بزرگ یکسان درنظر گرفته می شوند و اگر flag برابر false باشد، تابع در انجام عمل مقایسه نسبت به حروف کوچک و بزرگ حساس خواهد بود.

مثال۱۶)

```
string  s1 =”New C# Course” , s2= “new c# course” ;
int x = string.Compare ( s1, s2 ,true);
int y = string.Compare ( s1, s2 , false );
```

در انجام عمل مقایسه در دستور دوم حروف کوچک و بزرگ یکسان درنظر گرفته می شوند ولی در دستور سوم بین حروف کوچک و بزرگ معادل تفاوت قایل می شود. که با توجه به این مطلب مقدار x برابر صفر و مقدار y برابر یک خواهد بود.

۱۰. متد IndexOf

با این متد می توان مکان زیررشته ای را در رشته دیگری بدست آورد. که گرامر استفاده از آن در زیر آورده شده است:

```
int  index = رشته۱.IndexOf ( رشته۲ );
```

با توجه به گرامر فوق می توان گفت که متد فوق به دنبال مکان رشته۲ در رشته۱ می گردد و چنانچه رشته۲ در رشته۱ وجود داشته باشد، مکان شروع آن در index قرار داده می شود، درغیر اینصورت مقدار index برابر منفی یک خواهد بود.

مثال۱۷)

```
string s1 = “USB CSharp Course”,  s2= “CSharp” ;
int index=  s1. IndexOf ( s2 );
```

با توجه به اینکه s2 زیر رشته s1 می باشد، پس مکان شروع آن یعنی عدد ۴ در index قرار داده می شود.

تذکر۱۲: همچنین به کمک این تابع می توان عمل جستجو را از مکان خاصی از رشته اصلی شروع کرد. برای انجام این کار اندیس مکانی را که عمل جستجو باید از آنجا شروع شود، به عنوان پارامتر دوم به تابع می دهیم.

مثال۱۸)

```
string s1 = “USB CSharp Course”,  s2= “CSharp” ;
int index1 = s1. IndexOf ( s2 );
int index2 = s1.IndexOf( s2, 5 );
```

با توجه به دستور سوم باید عمل جستجو از اندیس ۵ به بعد شروع شود و چون از مکان ۵ به بعد رشته s2 در s1 وجود ندارد، مقدار index برابر ۱- خواهد شد.

تذکر۱۳: اگر رشته دوم چندین بار در رشته اول تکرار شده باشد، متد فوق مکان اولین تکرار را بر می گرداند.

۱۱. متد LastIndexOf

عملکرد این متد مشابه متد فوق می باشد، فقط این متد مکان آخرین تکرار رشته دوم را در رشته اول بر می گرداند.

مثال۱۹)

```
string s1 = "USB C# Course at USB Univ.", s2 = "USB" ;
int index=  s1.LastIndexOf (s2 );
```

با توجه به توضیحات بالا، مقدار index برابر ۱۷ خواهد بود.

۱۲. متد Remove

این متد جهت حذف بخشی از یک رشته بکار برده می شود.

از این متد به دو صورت می توان استفاده نمود:

- ✓ از مکان خاصی تا انتهای رشته را حذف نماید.
- ✓ از مکان خاصی به تعداد مشخصی کارکتر حذف نماید.

که گرامر استفاده از متد مزبور در زیر آمده است:

```
رشته۲ = رشته۱.Remove ( index , count );
```

در گرامر فوق استفاده از count اختیاری است و درصورت مشخص بودن آن، نمایانگر تعداد کارکترهایی است که باید حذف شوند.

مثال۲۰)

```
string s= "USB CSharp Programming Course!" ;
string s1=s.Remove( 4 );
string s2=s.Remove ( 4,20 );
```

با توجه به مثال فوق در دستور دوم مقدار رشته s1 برابر USB خواهد بود، زیرا کلیه کارکترهای بعد از مکان ۴ام رشته s حذف خواهند شد. همچنین مقدار رشته s2 برابر !USB Course می باشد، زیرا ۲۰ کارکتر از رشته s با شروع از مکان ۴ حذف خواهد شد.

۱۳. متد SubString

از این متد جهت جدا ساختن بخشی از یک رشته استفاده می شود که مشابه متد فوق به دو شیوه می توان از ان بهره برد.

گرامر استفاده از متد فوق در زیر آمده است:

```
رشته۲ = رشته۱.SubString ( index , count );
```

مشابه متد Remove چنانچه تعداد کارکترها مشخص نباشد(در بخش count)، از مکان مشخص شده در index تا انتهای رشته بعنوان زیر رشته درنظر گرفته می شود. مثال ۲۱ بیانگر این موضوع می باشد.

مثال۲۱)

```
string s= "USB CSharp Programming Course!" ;
string s1=s.SubString( 23 );
string s2=s.Remove ( 3 );
```

با توجه به مثال فوق مقدار رشته s1 برابر Course! و مقدار رشته s2 برابر USB خواهد شد.

۱۴. متد PadLeft

با این متد می توان به هر تعداد فاصله و یا هر کارکتر دیگری را به ابتدای رشته اضافه نمود.

گرامر استفاده از این متد بصورت زیر می باشد:

```
رشته۲ = رشته۱.PadLeft ( Count , char );
```

با متد فوق، به تعداد Count کارکتر به ابتدای رشته۲ افزوده می شود. که اگر پارامتر دوم مشخص نشود، دراینصورت فاصله به ابتدای آن افزوده می شود.

مثال۲۲)

```
string  s="C#" ;
string s1=s.PadLeft ( 10, '&' );
```

پس از اجرای دستورات فوق، به تعداد ۸ کارکتر & با ابتدای رشته C# افزوده می شود که رشته s1 برابر &&&&&&&&C# خواهد شد.(طول کل رشته باید به ۱۰ برسد)

۱۵. متد PadRight

با این متد می توان به هر تعداد فاصله و یا هر کارکتر دیگری را به انتهای رشته اضافه نمود.

گرامر استفاده از این متد بصورت زیر می باشد:

```
رشته۲ = رشته۱.PadRight ( Count , char );
```

با متد فوق، به تعداد Count کارکتر به انتهای رشته۲ افزوده می شود. که اگر پارامتر دوم مشخص نشود، دراینصورت فاصله به انتهای آن افزوده می شود.

مثال۲۳)

```
string  s="C#" ;
string s1=s.PadRight ( 10, '&' );
```

پس از اجرای دستورات فوق، به تعداد ۸ کارکتر & با انتهای رشته C# افزوده می شود که رشته s1 برابر C#&&&&&&&& خواهد شد.(طول کل رشته باید به ۱۰ برسد)

۱۶. متد های Trim، TrimEnd و TrimStart

از این متدها می توان جهت حذف مجموعه ای از کارکترها از ابتدا و یا انتهای رشته استفاده نمود.
متد Trim هم کارکترهای ابتدایی و هم کارکترهای انتهایی را حذف می نماید در صورتیکه متد TrimEnd فقط کارکترهای انتهایی و متد TrimStart فقط کارکترهای ابتدایی را حذف می نماید.

مثال۲۴)

```
string s= "&&&000C# Course@@&&&&&  ;
string s1= s.TrimStart('&' , '0' );
```

با اجرای این دستور، به هر تعداد کارکتر & و 0 که در ابتدای رشته وجود داشته باشد، از ابتدای آن حذف می شود. که خروجی برابر C# Course@@&&&&& می شود.

```
string s2= s.Trim ( '&','@' );
```

در این حالت کلیه کارکترهای & و @ از ابتدا و انتهای رشته s حذف می شوند. که خروجی برابر 000C# Course می شود.

۱۷. متدهای EndWith/StartWith

به کمک این دو متد می توان چک نمود که آیا زیررشته ای در ابتدا و یا انتهای رشته وجود دارد یا خیر؟ که خروجی این متد مقدار true/false می باشد.
گرامر استفاده از این متدها بصورت زیر می باشد:

```
bool check = رشته۱.EndWith/StartWith ( رشته۲ );
```

در گرامر فوق چک می شود که آیا رشته۲ بعنوان زیر رشته ای در ابتدا و یا انتهای رشته۱ وجود دارد یا خیر ؟

مثال۲۵)

```
string s= "C# Course";
string s1= "USB " + s + " Programming";
bool b1=s1.StartWith ("USB");
bool b2=s.EndWith( "Course");
bool b3=s.StartWith( "USB");
```

با توجه به توضیحات فوق و مقادیر متغیرهای s و s1، مقادیر متغیرهای b1 و b2 برابر true و مقدار متغیر b3 برابر false خواهد بود.

فصل چهارم: رشته ها

مثال۲۶) برنامه ای بنویسید که رشته های s1 و s2 را از کاربر گرفته، سپس تعداد دفعات تکرار رشته s2 در رشته s1 را محاسبه نموده و چاپ نماید.

```
namespace StrTst
{
    class Program
    {
        static void Main(string[] args)
        {
            int l = 0, count = 0;
            string s1 = Console.ReadLine();
            string s2 = Console.ReadLine();

            int x = s1.IndexOf(s2);

            while (x != -1)
            {
                count++;
                l = x + s2.Length;
                x=s1.IndexOf(s2 ,l);
            }
            Console.WriteLine(count);
            Console.Read();
        }
    }
}
```

مثال۲۷) برنامه ای بنویسید که رشته های s1 ، s2 و s3 را از کاربر گرفته، سپس در رشته s1 کلیه زیر رشته های s2 را با رشته s3 جایگزین نماید.

```
using System;
namespace StrTst
{
    class Program
    {
        static void Main(string[] args)
        {
            int l = 0;
            string s1 = Console.ReadLine();
            string s2 = Console.ReadLine();
            string s3 = Console.ReadLine();
            int x = s1.IndexOf(s2);

            while (x != -1)
            {
                s1 = s1.Remove(x, s2.Length);
                s1 = s1.Insert(x, s3);
                l = x + s3.Length;
                x=s1.IndexOf(s2 ,l);
            }
            Console.WriteLine(s1);
            Console.Read();
        }
    }
}
```

فصل چهارم: رشته ها

شرح برنامه :

در برنامه فوق ابتدا مکان رشته s2 در رشته s1 پیدا شده، سپس رشته s2 از آنجا حذف می گردد. سپس رشته s3 در مکان مورد نظر درج می گردد و این عمل تا زمانیکه رشته s2 در رشته s1 وجود دارد، ادامه پیدا می کند.

مثال۲۸) برنامه ای بنویسید که کلیه فاصله های اضافی درون رشته s را حذف نماید، بگونه ای که بین کلمات فقط یک فاصله وجود داشته باشد.

```
using System;
namespace StrTst
{
    class Program
    {
        static void Main(string[] args)
        {
            int x,l=0;
            string s = Console.ReadLine();

            s = s.Trim();
            x = s.IndexOf(' ',l);

            while (x != -1)
            {
                while (s[x+1]==' ')
                    s=s.Remove(x,1);

                l = x+1;
                x=s.IndexOf(' ' ,l);
            }
            Console.WriteLine(s);
            Console.Read();
        }
    }
}
```

شرح برنامه:

در این برنامه ابتدا فاصله های ابتدایی و انتهایی رشته ورودی حذف می شود، سپس حلقه while بیرونی تا زمانی که فاصله اضافی در رشته وجود داشته باشد، تکرار می شود.

با پیدا شدن یک فاصله در رشته به کمک حلقه while داخلی کلیه فاصله های پشت سرهم بعد از فاصله پیدا شده حذف می گردد. با حذف فاصله های پشت سرهم، به دنبال کارکتر فاصله در ادامه رشته می گردیم و روند فوق تا حذف کلیه فاصله های اضافی ادامه پیدا می کند.

مثال۲۹) برنامه ای بنویسید که تعداد کلمه های درون رشته ورودی را بشمارد. با این فرض که کلمه ها با فاصله از همدیگر جدا شده اند.

شرح برنامه:

ابتدا مشابه مثال قبل فاصله های اضافی را حذف نموده، سپس تعداد کارکترهای فاصله درون رشته را می شماریم.

فصل چهارم: رشته ها

مثال ۳۰) برنامه ای بنویسید که رشته ای از ورودی دریافت نموده، مجموع ارقام درون رشته را چاپ نماید.

```
using System;
class Program
{
    static void Main(string[] args)
    {
        string s = Console.ReadLine();
        int x = 0;

        foreach (char ch in s)
            if (ch >= '0' && ch <= '9')
                x += ch-'0';

        Console.WriteLine(x);
        Console.Read();
    }
}
```

شرح برنامه:

همانگونه که مشاهده می نمایید در این برنامه ابتدا به کمک یک حلقهٔ `foreach` به کلیه کارکترهای درون رشته ورودی دسترسی پیدا می کنیم، سپس چک می شود که آیا کارکتر جزو کارکترهای رقمی هست یا نه؟ اگر کارکتر مزبور جزو کارکترهای رقمی باشد، پس از تبدیل شدن آن به معادل رقمی به مجموع افزوده می شود.

تمرین

۱. برنامه ای بنویسید که با دریافت یک رشته و عدد `k` تعداد کلمات `k` حرفی رشته ورودی را چاپ نماید.

۲. برنامه ای بنویسید که با دریافت یک رشته و عدد `k`، کلیه کلمات `k` حرفی درون رشته ورودی را حذف نماید.

۳. برنامه ای بنویسید که با دریافت دو رشته `s1` و `s2` و عدد `k`، تعیین نماید آیا زیر رشته ای مشترک و با طول `k` در هر دو رشته وجود دارد یا خیر ؟

به عنوان مثال اگر `s1="USB C# Base Course"` و `s2="MIT Open Course"` و `k=6` باشد، در اینصورت جواب سوال فوق درست می باشد زیرا زیر رشته `Course` دارای طول ۶ می باشد که بین دو رشته مشترک می باشد.

۴. برنامه ای بنویسید که یک عبارت ورودی را که دارای شرایط داده شده در زیر می باشد، از ورودی دریافت نموده، سپس آنرا طبق روش داده شده بصورت رمز در آورد.

رشته ورودی دارای شرایط زیر می باشد:

- ✓ کلمات فقط با فاصله از همدیکر جدا شده اند
- ✓ حداقل طول کلمات برابر دو می باشد
- ✓ هیچگونه علامت نقطه گذاری در عبارت ورودی وجود ندارد.

روش رمزنگاری:

- ✓ اولین حرف هر کلمه را به انتهای آن کلمه منتقل نمایید
- ✓ به عبارت بدست آمده از مرحلهٔ قبل حروف ay را اضافه نمایید.
- ✓ کلیه حروف j را به H و حروف H را به j تبدیل نمایید.

به عنوان مثال چنانچه کلمه the با روش فوق رمز گردد به کلمه jetay تبدیل می شود.

۵. برنامه ای بنویسید که یک عبارت رشته ای از ورودی دریافت نموده، سپس تعداد کلمات یک حرفی، دو حرفی، سه حرفی و غیره آنرا چاپ نماید.
بعنوان مثال رشته s="my First C# Note" دارای دو کلمه دو حرفی و یک کلمه چهار حرفی و یک کلمه پنج حرفی می باشد.

وما توفیقی الا باالله

بلوچ زهی

۸۵/۷/۲۲

فصل پنجم: متدها

اغلب برنامه های کامپیوتری که حل مسائل دنیای واقع را بر عهده دارند، بسیار بزرگتر و پیچیده تر از برنامه هایی هستند که تا این فصل ارائه شده اند. تجربه نشان داده است که یکی از بهترین روشها برای توسعه و نگهداری برنامه های پیچیده و بزرگ، ساختن آنها از تکه ها و اجزای کوچکتر می باشد که کار با هر یک از آنها آسانتر از کار با برنامهٔ اولیه است. به این روش ، روش « تقسیم و غلبه » گفته می شود.

در C# این اجزاء و واحدها عبارتند از کلاسها و متدها که کلاسها در فصلهای بعد به تفصیل مورد بررسی قرار می گیرند و در این فصل فقط به متدها پرداخته می شود. باید متذکر شد که در C# متدها بخشی از کلاسها می باشند و به عنوان یک واحد جداگانه درنظر گرفته نمی شوند.

برنامه نویس می تواند متدهایی بنویسد که کارهای معینی انجام می دهند، سپس این متدها را در نقاطی از برنامه که موردنیاز می باشند، فراخوانی نماید.

تذکر۱: در C# متدها فقط باید درون کلاسها تعریف شوند و در این زبان متدهای عمومی مانند زبان C وجود ندارد.

۱. انواع متدها

متدهای مورد استفاده در C# بر دو نوع می باشند:

- متدهای استاتیک

این متدها مشابه متدهای مورد استفاده در زبان C بوده و بدون اینکه شیئی از نوع کلاس حاوی متد (متغیری از نوع کلاس حاوی متد) وجود داشته باشد، می توانیم آنها را فراخوانی نماییم.

- متدهای غیر استاتیک

جهت استفاده از این متدها حتما باید شیئی از کلاس حاوی این متدها ایجاد گردد، سپس به کمک شیء مزبور می توان این متدها را فراخوانی نمود.

در این فصل به چگونگی ایجاد و استفاده از متدهای استاتیک پرداخته می شود و بحث در مورد متدهای غیر استاتیک را به فصل مربوط به کلاسها موکول می کنیم.

۲. نحوه تعریف و فراخوانی متدها

برای تعریف متدهای نوع استاتیک درون یک کلاس، که قابل استفاده در آن کلاس باشند از گرامر زیر پیروی می شود:

```
static   <نوع بازگشتی>  <نام متد>  (  لیست پارامترها  )
{
        بدنه متد
}
```

- **نوع بازگشتی**

نوع بازگشتی می تواند یکی از انواع موجود در C# باشد و مبین نوع مقداری است که متد به عنوان نتیجه به برنامه فراخواننده برمی گرداند. چنانچه متد هیچ مقداری را به عنوان نتیجه بر نگرداند، نوع بازگشتی متد void درنظر گرفته می شود.

فصل پنجم: متدها

- **نام متد**

نام متد نامی است متناسب با عملکرد متد که به آن نسبت داده می شود و در فراخوانی متد از آن استفاده می شود. توصیه می گردد که در نامگذاری متدها از قوانین نامگذاری ذکر شده در فصل دوم استفاده گردد.

- **لیست پارامترها**

این بخش حاوی لیست پارامترهای ورودی و خروجی متد می باشد. یعنی هم می توان به کمک پارامترها مقداری را به متد ارسال کرد و هم می توان از پارامترها جهت دریافت نتایج بدست آمده از متد استفاده نمود که چگونگی تعریف پارامترهای ورودی و خروجی در ادامه این فصل مورد بررسی قرار می گیرد.

- **بدنه متد**

بدنه متد مشخص کننده عملکرد متد می باشد و حاوی مجموعه دستوراتی است که متد جهت رسیدن به هدف مورد نیاز خویش از آنها بهره می برد.

در زیر به ارائه چند مثال از نحوه تعریف متدها می پردازیم:

مثال۱) متدی بنویسید که دو عدد صحیح را بعنوان پارامترهای ورودی دریافت نموده و ماکزیمم آنها را برگرداند.

شرح متد:

پارامترهای متد متغیرهای صحیح می باشند و چون ماکزیمم دو عدد نیز یکی از دو عدد ورودی است، پس ماکزیمم نیز عددی صحیح است. یعنی نوع مقدار بازگشتی متد با نوع ورودی ها یکسان می باشد.(این موضوع همیشه صادق نیست).

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

    }

    static int Max(int m, int n)
    {
        int tmp;
        if (m > n)
            tmp = m;
        else
            tmp = n;
        return tmp;
    }
}
```

از دستور return جهت بازگرداندن مقدار نتیجه از متد استفاده می شود. که در مثال فوق پس از محاسبه مقدار ماکزیمم، این مقدار توسط دستور return به برنامه فراخواننده برگردانده می شود.

پس از تعریف متد باید از این متد به نحو مناسبی استفاده گردد که گرامر فراخوانی متد بصورت زیر می باشد:

<مقدار بازگشتی> = <نام کلاس>.<نام متد> (نام پارامترها) ;

مقدار بازگشتی نام متغیری است که مقدار بازگردانده شده از متد را در خود نگهداری می نماید.

تذکر۲: در C# می توان از مقدار بازگشتی صرفنظر کرد.

مثال۲) متد Main مثال قبل را به گونه ای بازنویسی نمایید که با ارسال دو مقدار به متد Max و دریافت مقدار ماکزیمم از این متد، ماکزیمم مقادیر دریافتی را با کمک گرفتن از این متد به انجام برساند.

```
using System;
class Program
{
    static void Main()
    {
        int x = int.Parse(Console.ReadLine());
        int y = int.Parse(Console.ReadLine());
        int result;
        result = Max(x, y);
        Console.WriteLine(result);
        Console.Read();
    }

    static int Max(int m, int n)
    {
        int tmp;
        if (m > n)
            tmp = m;
        else
            tmp = n;
        return tmp;
    }
}
```

شرح برنامه:

در متد Main ابتدا دو مقدار صحیح از کاربر دریافت شده، سپس این دو مقدار جهت محاسبه مقدار ماکزیمم آن دو به متد Max نوشته شده در مرحله قبل ارسال شده اند و نتیجه برگشت داده شده از متد Max در متغیر result قرار داده شده است.

```
result = Program.Max ( x , y );
```

چون در این مثال متدها با متد Main درون یک کلاس واقع شده اند، می توان از نام کلاس در ابتدای فراخوانی صرفنظر کرد.

```
result = Max ( x , y );
```

سپس مقدار نتیجه در خروجی چاپ شده است. این عمل را با دستور زیر نیز می توان انجام داد و نیازی به متغیر کمکی result نمی باشد:

```
Console.WriteLine (Max ( x , y )  );
```

می توان متد فوق را بصورت ; (x , y) Max نیز فراخوانی نمود که در این حالت از مقدار بازگشتی متد صرفنظر شده است.

مثال۳) برنامه ای بنویسید که با دریافت مقادیر n و x مقدار سری زیر را محاسبه نماید.

$$S=\sum_{i=1}^{n}\frac{x^{3i+1}}{3i+1}=\frac{x^4}{4}+\frac{x^7}{7}+\frac{x^{10}}{10}+...+\frac{x^{3n+1}}{3n+1}$$

ابتدا متدی به نام CalPow می نویسیم که مقدار پارامتر اول را به توان پارامتر د.م می رساند. سپس از این متد در محاسبه مقدار سری بهره می بریم

```
using System;
class Program
{
    static void Main()
    {
        Console.Write("Enter n?");
        uint n=uint.Parse(Console.ReadLine());
        Console.Write("Enter x?");
        uint x=uint.Parse(Console.ReadLine());
        double sum = 0.0;

        for (uint i = 1; i <= n; i++)
            sum += CalPow(x, 3 * i + 1) / (3*i +1);

        Console.WriteLine(sum);
        Console.Read();
    }

    static ulong CalPow(uint m, uint n)
    {
        ulong tmp=1;
        for (uint i = 1; i <= n; i++)
            tmp *= m;

        return tmp;
    }
}
```

همانگونه که مشاهده می شود در این مثال پس از دریافت مقادیر n و x، در یک حلقه for با کمک گرفتن از متد CalPow مجموع n جمله اول سری محاسبه شده است.

نمونه ای از خروجی برنامه فوق در زیر نشان داده شده است:

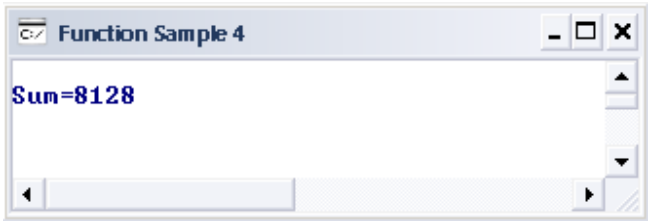
مثال۴) برنامه ای بنویسید که با بهره گرفتن از متدی با نام IsComplete که تشخیص می دهد آیا پارامتر ورودی آن عددی کامل هست یا نه، مجموع اعداد کامل (تام) چهار رقمی را محاسبه نماید.

شرح برنامه:

متد IsComplete با دریافت عددی صحیح و مثبت و چک تام بودن آن مقدار true و یا false را برمی گرداند. که اگر عدد تام باشد مقدار true و درغیر اینصورت مقدار false را برگرداند.

```
using System;
class Program
{
    static void Main()
    {
        ulong sum = 0;
        for ( uint i=1000 ; i< 9999 ; i++)
            if (IsComplete(i))
                sum += i;
        Console.WriteLine("\nSum=" + sum);
        Console.Read();
    }

    static bool IsComplete(uint n)
    {
        uint sum = 0;
        for (uint i = 1; i <= n / 2; i++)
            if (n % i == 0)
                sum += i;
        if (sum == n)
            return (true);
        else
            return (false);
    }
}
```

نمونه خروجی برنامه فوق در زیر نشان داده شده است:



۳. روشهای ارسال پارامترها در متدها

برای ارسال پارامترها به متدها در زبانهای متفاوت روشهای گوناگونی وجود دارد . در C# نیز روشهای گوناگونی برای ارسال پارامترها وجود دارد که در زیر به آنها اشاره می شود:

الف)ارسال پارامتر با مقدار(pass by value)

این شیوه همان روشی است که تا بحال در متدها جهت ارسال آرگومانها از آن استفاده می شد. هنگامی که آرگومانی بصورت فراخوانی با مقدار به متد ارسال می شود، یک کپی از مقدار آرگومان ایجاد شده و به متد

فراخوانده شده ارسال می شود. تغیرات انجام شده در این کپی، تغیری در مقدار متغیر اصلی متد فراخواننده ندارد.

کلیه آرگومانهای ارسالی در این فصل(تا بحال) بصورت فراخوانی با مقدار به متدها رد شده اند.

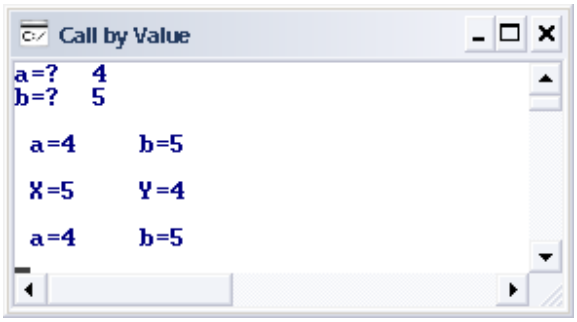
مثال۵) این مثال چگونگی استفاده از متدی را که آرگومانها بصورت مقدار به آن رد شده اند نشان می دهد. همچنین عدم تغیر آرگومانهای متد فراخواننده در هنگام تغیر پارامترهای متد فراخوانده شده مورد بررسی واقع می شود.

```
using System;
class Program
{
    static void Main()
    {
        Console.Write("a=?  ");
        int a = int.Parse(Console.ReadLine());
        Console.Write("b=?  ");
        int b = int.Parse(Console.ReadLine());

        Console.WriteLine("\n a={0}\tb={1}", a, b);
        Swap(a, b);
        Console.WriteLine("\n a={0}\tb={1}", a, b);
        Console.Read();
    }

    static void Swap(int x, int y)
    {
        int tmp = x;
        x = y ;
        y = tmp ;
        Console.WriteLine("\n X={0}\tY={1}", x, y);
    }
}
```

در مثال فوق ابتدا دو عدد از کاربر دریافت شده، سپس این دو عدد به متد Swap ارسال می شوند. در متد موردنظر مقادیر پارامترهای ورودی با همدیگر عوض می شود. سپس کنترل به متد Main برمی گردد.

مقادیر متغیرها در کلیه مراحل فوق به چاپ رسیده است و همانگونه که خروجی برنامه نشان می دهد، تغیرات انجام شده درون متد Swap به آرگومانهای نظیر آنها در متد Main اعمال نمی شوند و این متغیرها همان مقادیر اولیه خود را دارا می باشند.

نمونه ای از خروجی برنامه در زیر نشان داده شده است:



ب) ارسال پارامتر بصورت ارجاع (call by reference)

اگر بخواهیم تغیرات انجام شده بر روی متغیرها در متد فراخوانده شده به آرگومانهای نظیر آنها در متد فراخواننده اعمال شود، می توان از این نوع فراخونی استفاده نمود.

دو روش ارسال پارامتر بصورت ارجاع وجود دارد که عبارتند از:

۱. استفاده از کلمه کلیدی ref

در این حالت هم در زمان تعریف متد و هم در زمان فراخوانی آن، باید قبل از پارامترهایی که از نوع ارجاع هستند، کلمه کلیدی ref را بکار برد.

تذکر۳: در این حالت آرگومانهای ارسالی به متد قبل از ارسال حتماً باید دارای مقدار اولیه باشند. در غیر اینصورت با خطای زمان کامپایل مواجه خواهیم شد.

مثال۶) این مثال چگونگی استفاده از متدی را که آرگومانها بصورت ref به آن رد شده اند نشان می دهد. همچنین چگونگی تغیر آرگومانهای متد فراخواننده در هنگام تغیر پارامترهای متد فراخوانده شده را بررسی می نماییم.

```
using System;
class Program
{
    static void Main()
    {
        Console.Write("a=?  ");
        int a = int.Parse(Console.ReadLine());
        Console.Write("b=?  ");
        int b = int.Parse(Console.ReadLine());

        Console.WriteLine("\n a={0}\tb={1}", a, b);
        Swap(ref a, ref b);
        Console.WriteLine("\n a={0}\tb={1}", a, b);
        Console.Read();
    }

 static void Swap(ref int  x,ref int y)
    {
        int tmp = x;
        x = y ;
        y = tmp ;
        Console.WriteLine("\n X={0}\tY={1}", x, y);
    }
}
```

مشاهده می شود در ارسال آرگومانها بصورت ref باید کلمه کلیدی ref را هم در زمان فراخوانی متد و هم در زمان تعریف متد، ذکر کرد.

همانگونه که در خروجی برنامه مشاهده می شود، اگر پارامترها از نوع ref معرفی گردند، تغیر مقدار آنها بر روی آرگومانهای نظیر تاثیر می گذارد.

۲. استفاده از کلمه کلیدی out

یکی دیگر از روشهایی که می توان به کمک آن تغیرات انجام شده بر روی پارامترها را به آرگومانهای نظیر اعمال نمود، ارسال پارامترها بصورت خروجی (out) می باشد.

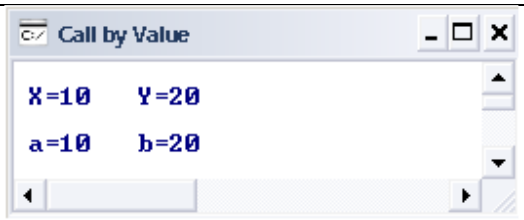
در این حالت نیز مشابه حالت قبل، باید قبل از نام آرگومانها از کلمه کلیدی out استفاده نمود. همچنین باید قبل از نوع پارامترها در تعریف متد از این کلمه استفاده نمود.

تذکر۴: در این حالت نیازی به مقداردهی آرگومانها قبل از ارسال به متد نمی باشد. بلکه باید پارامترها را در داخل بدنه متد مقداردهی نماییم. درغیر اینصورت (چنانچه پارامترهای نوع out را در درون متد مقداردهی اولیه ننماییم) با خطای زمان کامپایل مواجه خواهیم شد.

مثال۷) این مثال چگونگی استفاده از متدی را که آرگومانها بصورت out به آن رد شده اند نشان می دهد. همچنین چگونگی تغیر آرگومانهای متد فراخواننده در هنگام تغیر پارامترهای متد فراخوانده شده را بررسی می نماییم.

```
using System;
class Program
{
    static void Main()
    {
        int a, b;

        Swap(out a, out b);
        Console.WriteLine("\n a={0}\tb={1}", a, b);
        Console.Read();
    }

 static void Swap(out int  x,out int y)
    {
        x=10 ;;
        y=20 ;
        Console.WriteLine("\n X={0}\tY={1}", x, y);
    }
}
```

در مثال فوق مشاهده می گردد که پارامترهای نوع out درون متد مقداردهی اولیه شده اند و چون پارامترها از نوع out می باشند، این مقادیر به آرگومانهای نظیر یعنی a و b نیز اعمال می گردند. نتیجه برنامه فوق در زیر نشان داده شده است.



مثال۸) خروجی برنامه زیر را تعیین نمایید.

```
using System;
class Program
{
    static void Main()
    {
        int a=1, b=2;
        Console.WriteLine("\n Main:\n a={0}\tb={1}", a, b);
        Tst1(a, ref b);
        Console.WriteLine("\n Main:\n a={0}\tb={1}", a, b);
        Tst2(ref a, out b);
        Console.WriteLine("\n Main:\n a={0}\tb={1}", a, b);
        Console.Read();
    }



  static void Tst1(int  x,ref int y)
    {
        x += y;
        y *= x;
        Console.WriteLine("\n Tst1=> \n a={0}\tb={1}", x, y);
    }
    static void Tst2(ref int x,out int y)
    {
        x += 4;
        y = x + 2;
        Console.WriteLine("\n Tst2=>\n a={0}\tb={1}", x, y);
    }
}
```

برنامه فوق حاوی دو متد به نامهای Tst1 و Tst2 می باشد که پارامترهای متد Tst1 به ترتیب از نوع مقداری و ارجاعی(ref) و پارامترهای متد Tst2 از نوع ارجاعی(ref) و خروجی (out) می باشند.

توجه: پارامترهای نوع خروجی قبل از استفاده در متد حتماً باید مقدار اولیه بگیرند. (دستور **y=x+2** در متد).

متد Main

	a	b
مقادیر اولیه	1	2
بعد از اجرای Tst1	1	6
بعد از اجرای Tst2	5	7

متد Tst2

x (ref)	y (out)
1	6
5	7

متد Tst1

X	y (ref)
1	2
3	6

با توجه به مقادیر فوق مشاهده می گردد که تغیر مقادیر پارامترهای نوع مقداری بر آرگومانهای متناظر تاثیری ندارد در حالیکه اگر نوع پارامتر ref و یا out باشد، این تغیرات به آرگومانهای نظیر اعمال می گردد.

فصل پنجم: متدها

۳. سربارگذاری متدها [1]

در C# تعریف چندین متد با نامهای یکسان مادامی که مجموعهٔ پارامترهای آنها متفاوت باشد، امکان پذیر می باشد. به این قابلیت سربارگذاری (گرانبار کردن) متد گفته می شود. هنگامی که یک متد سربارگذاری شده فراخوانده می شود، کامپایلر با بررسی تعداد، نوع و ترتیب آرگومانهای به کار رفته در فراخوانی، متد مناسب را انتخاب می نماید.

معمولاً سربارگذاری متدها برای ایجاد چند متد با نامهای یکسان که بر روی انواع دادهٔ متفاوتی وظایف مشابهی را انجام می دهند، به کار می روند.

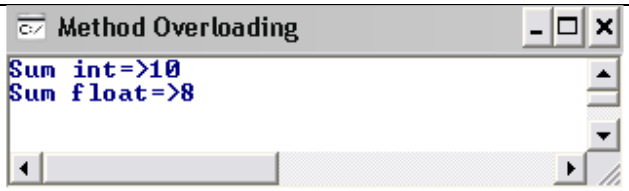
ساده ترین نمونه از متدهای سربارگذاری شده متد WriteLine می باشد که در C# با ۱۹ حالت متفاوت سربرگذاری شده است که با چند نمونه از آن آشنا شده اید. بعنوان مثال برای چاپ مقادیر بولی،صحیح، اعشاری، رشته ای و غیره از این متد استفاده می گردد.

مثال۹) در کد زیر متد Sum بگونه ای سربرگذاری شده است که هم مجموع دو عدد صحیح را محاسبه می نماید و هم مجموع دو عدد از نوع float را.

```
using System;
class MainClass
{
      static void Main()
      {
       int a=4, b=6;
       float c=3.4f, d=4.6f;
       int SumInt=Sum(a,b);
       float SumFloat=Sum(c,d);
       Console.WriteLine("Sum int=>{0}\nSum float=>{1}\n",SumInt,SumFloat);
       Console.ReadLine();
      }

      static int Sum(int x, int y)
      {
            return x+y;
      }

      static float Sum(float x, float y)
      {
            return x+y;
      }
      }
```

خروجی برنامه فوق در زیر نمایش داده شده است. در مثال فوق در فراخوانی اول، متد Sum اول که مجموع دو عدد صحیح را محاسبه می نماید فراخوانی می شود و در فراخوانی دوم متد Sum دوم که دو عدد float را با همدیگر جمع می نماید، فراخوانی می شود. لازم است جهت درک بهتر مثال فوق، برنامه را خط به خط کامپایل نمایید تا چگونگی فراخوانی متدهای سربارگذاری شده را مشاهده کنید.

[1] Method Overloading



مثال۱۰) نمونه دیگری از سربارگذاری متدها در برنامه زیر مشهود است که در آن تفاوت متدهای سربارگذاری شده، نه در نوع پارامترها بلکه در تعداد پارامترها می باشد.

```
using System;
class MainClass
{
	static void Main()
	{
	 int a=4, b=6,c=5;;
	 int SumInt1=Sum(a,b);
	 int SumInt2=Sum(a,b,c);
	 Console.WriteLine("Sum int 1=>{0}\nSum int 2=>{1}\n",SumInt1,SumInt2);
	 Console.ReadLine();
	}

	static int Sum(int x, int y)
	{
		return x+y;
	}

	static int Sum(int x, int y, int z)
	{
		return x+y+z;
	}
}
```

در این برنامه بر خلاف برنامهٔ قبلی متد Sum بگونه ای سربارگذاری شده است تا بتواند هم دو عدد صحیح و هم سه عدد صحیح را با همدیگر جمع نماید. با توجه به فراخوانیهای انجام شده در Main، یکبار متد Sum با دو آرگومان صحیح و بار دیگر با سه آرگومان صحیح فراخوانی شده است که در هر مرحله متد متناظر نوشته شده فراخوانی می گردد(با توجه به تعداد پارامترها).

مثال۱۱)نوع دیگری از سربارگذاری می تواند به گونه ای باشد که در آن متدهای سربارگذاری شده در نوع فراخوانی با همدیگر متفاوت باشند. مثلاً در یک متد فراخوانی با مقدار و در دیگری فراخوانی با ارجاع داشته باشیم. برنامه زیر نمونه ای از این نوع سربارگذاری می باشد.

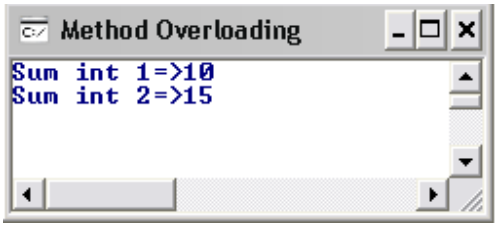
```
using System;
class MainClass
{
	static void Main(string[] args)
	{
	 int a=4, b=6;
	 int SumInt1=Sum(ref a,out b);
	 int SumInt2=Sum( a, b);
	 Console.WriteLine("Sum int 1=>{0}\nSum int 2=>{1}\n",SumInt1,SumInt2);
	 Console.ReadLine();
	}
	static int Sum(int x, int y)
	{
		return x+y;
```

```
        }

        static int Sum(ref int x,out  int y)
        {
              y=5;
              return x+y;
        }
}
```

در برنامه فوق تفاوت دو متد فقط در نوع فراخوانی پارامترهاست . در متد اول دو پارامتر از نوع فراخوانی با مقدار می باشند، درحالیکه در متد دوم هر دو نوع پارامتر از نوع ارجاع می باشد. در متد Main دو فراخوانی انجام شده است که با توجه به اینکه در فراخوانی اول، آرگومانهای متد بصورت ارجاعی ارسال شده اند، متد Sum دوم فراخوانی می شود ولی در فراخوانی دوم چون هر دو نوع پارامتر بصورت مقداری ارسال شده اند، متد Sum اول فراخوانی خواهد شد.

در فراخوانی اول پس از انجام عمل فراخوانی، مقادیر نهایی متغیرهای x و y به متغیرهای نظیر آنها (a و b) اعمال خواهد شد. پس از انجام عمل فراخوانی اول مقادیر a و b به ترتیب برابر با ۴ و ۵ می باشد. که این مقادیر در فراخوانی بعدی به متد Sum ارسال می شوند.

با توجه به مطلب فوق مقادیر بازگشتی دو متد با توجه به اعمال تغیرات پارامترها، یکسان خواهد بود که نتیجه آن در شکل زیر مشهود است.



تذکر۵ : نوشتن متدهای سربارگذاری شده ای که فهرست پارامترهای آنها (از نظر تعداد و نوع) یکسان ولی نوع مقدار بازگشتی توسط آنها با همدیگر متفاوت باشد، مجاز نیست و سبب ایجاد خطای زمان کامپایل می شود.

مثال۱۲) به عنوان مثال تعریف دو متد Sum بصورت زیر سبب بروز خطای زمان کامپایل CS0111 می شود. این خطا نمایانگر این است که در کلاس نوشته شده دو یا چند متد با نام و پارامترهای یکسان وجود دارد.(بدون توجه به نوع بازگشتی دو متد)

```
        static int Sum(int x, int y)
        {
              return x+y;
        }

        static float Sum( int x,  int y)
        {
              return x+y;
        }
```

تذکر۶ : نوشتن متدهای سربارگذاری شده ای که اختلاف پارامترهای آنها فقط در نوع فراخوانی ref و out باشد، مجاز نیست. نمی توان دو متد همنام نوشت که اختلاف پارامترهای متناظر آنها فقط در ref و out باشد.

فصل پنجم: متدها

مثال۱۳) تعریف دو متد بصورت زیر سبب بروز خطای زمان کامپایل CS0663 می گردد. این خطا نمایانگر وجود دو متد همنام در کلاس با پارمترهایی که اختلاف آنها فقط در ref و out است، می باشد.

```
static int Sum(ref int x, int y)
{
      return x+y;
}

static int Sum(out int x,  int y)
{
      return x+y;
}
```

اما اگر دو متد فوق بصورت زیر تعریف شوند خطای ایجاد شده رفع می گردد.

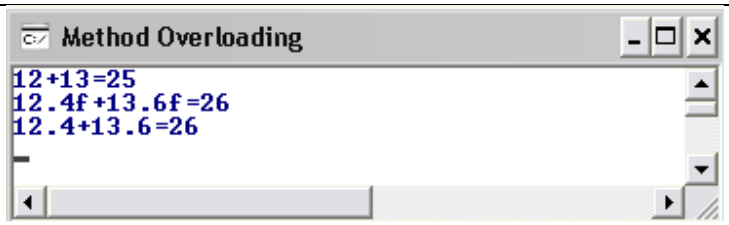
```
static int Sum(ref int x, int y)
{
      return x+y;
}

static int Sum( int x,  int y)
{
      return x+y;
}
```

زیرا در این حالت اختلاف فقط در کلمات ref و out نیست بلکه نوع پارامترها از نظر مقداری و ارجاعی با همدیگر متفاوت می باشند.

مثال۱۴)آیا برنامه زیر به درستی کامپایل می شود یا خیر؟ درصورت کامپایل برنامه خروجی آنرا محاسبه نمایید.

```
using System;
class MainClass
{
      static void Main()
      {
            Console.WriteLine("12+13="+Sum(12,13));
            Console.WriteLine("12.4f+13.6f="+Sum(12.4f,13.6f));
            Console.WriteLine("12.4+13.6="+Sum(12.4,13.6));
            Console.ReadLine();
      }
      static double Sum(double x,double y)
      {
            return x+y;
      }
}
```

در برنامه فوق متد Sum دو پارامتر از نوع double را بعنوان ورودی می پذیرد و خروجی آن نیز از نوع double می باشد. اما در فراخوانی آن در Main سه نوع پارامتر به آن ارسال شده است که نوع پارامترها در فراخوانی اول از نوع صحیح و در فراخوانی دوم از نوع float و در فراخوانی سوم از نوع double می باشند. در این حالت چون نوع همه مقادیری که به متد ارسال شده اند از نوع پارامترهای متد (double) کوچکتر می باشند(int و float)، در هنگام فراخوانی عمل تبدیل نوع کوچکتر به بزرگتر بصورت ضمنی انجام می گیرد و هر کدام از این دو نوع به نوع بزرگتر یعنی double تبدیل می شوند. در نتیجه هر سه فراخوانی بدرستی انجام می پذیرد. خروجی برنامه فوق در زیر نمایش داده شده است.



مثال۱۵)خروجی برنامه زیر را تعیین نمایید.

```
using System;
class MainClass
{
	static void Main()
	{
	 int a=4, b=6, result;
	 result=Sum(out a, b);
	 Console.WriteLine("a={0}\tb={1}\tresult={2}",a,b,result);
	 result=Sum( a,ref b);
	 Console.WriteLine("a={0}\tb={1}\tresult={2}",a,b,result);
	 result=Sum(out a,ref b);
	 Console.WriteLine("a={0}\tb={1}\tresult={2}",a,b,result);
	 result=Sum( a, b);
	 Console.WriteLine("a={0}\tb={1}\tresult={2}",a,b,result);
	 Console.ReadLine();
	}

	static int Sum(int x,ref int y)
	{
		x+=y;
		y++;
		return x+y;
	}

	static int Sum(out int x,ref  int y)
	{
		y+=5;
		x=y-2;
		return x+y;
	}
	static int Sum(out int x, int y)
	{
		x=y+2;
		y+= x;
		return x+y;
	}
	static int Sum( int x, int y)
	{
		x-=y;
		y+=2;
		return x+y;
	}
}
```

برای اینکه شرح برنامه واضح تر بیان گردد، متدهای سربرگذاری شده را به ترتیب به نام متد۱، متد۲، متد۳ و متد ۴ نامگذاری می نماییم.

با توجه به نوع پارامترهای فراخوانی اول، متد شماره ۳ در این حالت فراخوانده می شود و با توجه به بدنه متد مقادیر متغیرها برابر ۸ و ۱۴ خواهد بود. ولی از این دو مقدار فقط مقدار پارامتر اول بر آرگومان متناظر آن

تاثیر می گذارد زیرا فقط این آرگومان از نوع ارجاعی می باشد و آرگومان دیگر از نوع مقداری است. نتیجه تغیرات پارامترها پس از این فراخوانی در جدول زیر مشاهده می گردد که مقادیر متغیرهای a و b پس از این فراخوانی برابر ۸ و ۶ شده است.(فقط پارامتر اول اعمال گردیده است)

در فراخوانی دوم متد شماره یک فراخوانی شده است که مقادیر متغیرها و چگونگی اعمال تغیرات در جدول آمده است. در فراخوانی بعدی متد شماره ۲ و در فراخوانی آخری متد شماره ۴ فراخوانی شده است که مقادیر نهایی متغیرها در متدها و چگونگی اعمال تغیرات آنها در جدول زیر آمده است.

Main

بعد از فراخوانی	A	b	result
Main	4	6	
اول	8	6	22
دوم	8	7	21
سوم	10	12	22
چهارم	12	12	12

Sum #1

x	y
14	7

Sum #2

x	y
10	12

Sum #3

X	y
8	14

Sum #4

x	Y
-2	14

۴. متدهای ریاضی[۲]

کلاس ریاضی (Math Class) حاوی مجموعه ای از متدهای استاتیک می باشد که این متد ها امکانات لازم جهت انجام محاسبات معمول ریاضی را در اختیار برنامه نویس قرار می دهند. در این قسمت به تعدادی از این متدها اشاره می شود.

۱. متد Abs

این متد جهت محاسبه قدر مطلق مورد استفاده واقع می شود.

```
using System;
class MainClass
{
	static void Main()
	{
		double x=10.65 , y= -10.65 , r1, r2;
		r1 = Math.Abs(x);
		r2=Math.Abs(y);
		Console.WriteLine("Abs(10.65)={0}",r1);
		Console.WriteLine("Abs(-10.65)={0}",r2);
		Console.ReadLine();
	}
}
```

با توجه به اینکه قدرمطلق اعداد مثبت برابر با خود عدد و قد مطلق اعداد منفی برابر با قرینه عدد می باشد، مقدار r1 برابر با 10.65 و مقدار r2 نیز برابر با 10.65 خواهد بود.

۲. متد Sqrt

از این متد جهت محاسبه جذر (ریشه دوم) استفاده می شود.

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

[2] Math Method

```
            double x=25 , y= 65 , r1, r2;
            r1 = Math.Sqrt(x);
            r2=Math.Sqrt(y);
            Console.WriteLine("r1={0}",r1);
            Console.WriteLine("r2={0}",r2);
            Console.ReadLine();
      }
}
```

پس از اجرای مجموعه دستورات فوق ، مقادیر r1 و r2 به ترتیب برابر ۵ و ۸.۰۶۲۲۵۷۷ خواهد بود.

۳. متد Max

این متد جهت محاسبه ماکزیمم دو مقدار به کار برده می شود.

۴. متد Min

این متد مشابه متد فوق، جهت محاسبه می نیمم دو مقدار به کار برده می شود.

```
using System;
class MainClass
{
      static void Main()
      {
            double x=25 , y= 65 , r1, r2;
            r1 = Math.Max(x,y);
            r2 = Math.Min(x,y);
            Console.WriteLine("Min={0}",r2);
            Console.WriteLine("Max={0}",r1);
            Console.ReadLine();
      }
}
```

مقادیر r1 و r2 به ترتیب برابر ۶۵ و ۲۵ خواهند بود.

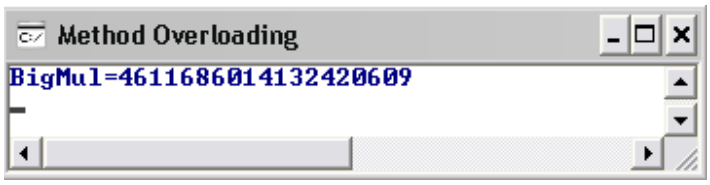
۵. متد BigMul

نتیجه ضرب دو عدد صحیح ۴ بایتی، معمولاً اندازه ای بزرگتر از چهار بایت خواهد داشت. از اینرو نمی توان آنرا در متغیری از نوع int که اندازه آن چهار بایت می باشد، ذخیره نمود.

متد فوق دو عدد صحیح ۴ بایتی را در هم ضرب نموده ونتیجه ۸ بایتی را به ما بر می گرداند که با این کار عملاً مشکل فوق رفع می گردد. چگونگی استفاده از متد فوق به همراه خروجی آن در زیر آمده است.

```
using System;
class MainClass
{
      static void Main()
      {
            int x=int.MaxValue , y=int.MaxValue ;
            long z;
            z=Math.BigMul(x,y);
            Console.WriteLine("BigMul={0}",z);
            Console.ReadLine();
      }
}
```

در مثال فوق ابتدا بزرگترین اعداد صحیح ۴ بایتی به دو متغیر x و y نسبت داده شده اند، سپس توسط متد BigMul حاصلضرب آندو محاسبه شده و نتیجه در متغیری ۸ بایتی به نام z ذخیره شده است. مقدار نتیجه در شکل زیر نشان داده شده است. همانگونه که مشاهده می شود، مقدار نتیجه بیشتر از آن است که بتوان در ۴ بایت آنرا ذخیره نمود.



۶. متد Pow

یکی از عملیات پایه که معمولاً در برنامه نویسی با آن مواجه هستیم، محاسبه توان می باشد. متد فوق جهت رفع این نیاز ایجاد شده است. چگونگی استفاده از این متد در برنامه زیر نشان داده شده است.

```
using System;
class MainClass
{
	static void Main()
	{
		double x=4, y=5;
		double z=Math.Pow(x,y);
		Console.WriteLine("x^y={0}",z);
		Console.ReadLine();
	}
}
```

خروجی برنامه فوق برابر ۱۰۲۴ خواهد بود.

۷. متد Exp

این متد جهت محاسبه تابع نمایی e^x بکار برده می شود.

```
using System;
class MainClass
{
	static void Main()
	{
		double x=4;
		double z=Math.Exp(x);
		Console.WriteLine("e^4={0}",z);
		Console.ReadLine();
	}
}
```

مقدار نتیجه برابر ۵۴.۵۹۸۱۵۰۰۳۳ می باشد.

۸. متد Ceiling

این متد x را به کوچکترین عدد صحیح بزگتر یا مساوی با x گرد می کند.

به عنوان مثال `Math.Ceiling(10.4)` برابر با ۱۱ و `Math.Ceiling(-10.4)` برابر با ۱۰- خواهد بود.

۹. متد Floor

این متد x را به بزرگترین عدد صحیح کوچکتر یا مساوی با x گرد می کند.

به عنوان مثال `Math.Floor( 10.4 )` برابر با ۱۰ و `Math.Floor( -10.4 )` برابر با ۱۱- خواهد بود.

۱۰. متد Truncate

این متد بخش صحیح x را برمی گرداند.

به عنوان مثال `Math.Truncate(10.4)` برابر با ۱۰ و `Math.Truncate(-10.4)` برابر با ۱۰- خواهد بود.

۱۱. متد Round

این متد عدد صحیح x را گرد می کند.

به عنوان مثال `MathRound( 10.48 )` برابر با ۱۰ و `Math.Round( -10.54 )` برابر با ۱۱- خواهد بود. در انجام عمل فوق چنانچه قسمت اعشار کمتر از 0.5 باشد از آن صرفنظر می شود، در غیر اینصورت به رقم قبلی یک واحد افزوده می شود.

۱۲. متد Log

لگاریتم طبیعی عدد x را محاسبه می نماید. (پایه برابر e)

۱۳. متد Log10

لگاریتم عدد x را محاسبه می نماید. (پایه برابر ۱۰)

به عنوان مثال `Math.Log10(100)` برابر با ۲ و `Math.Log(80)` برابر با ۴.۳۸۲۰۲۶۶۳ خواهد بود.

۱۴. متد Sin

سینوس عدد x را محاسبه می نماید. (x بر حسب رادیان)

۱۵. متد Sinh

سینوس هایپربولیک عدد x را محاسبه می نماید. (x بر حسب رادیان)

۱۶. متد Cos

کسینوس عدد x را محاسبه می نماید. (x بر حسب رادیان)

۱۷. متد Cosh

کسینوس هایپربولیک عدد x را محاسبه می نماید. (x بر حسب رادیان)

۱۸. متد Acos

آرک کسینوس عدد x را محاسبه می نماید.

۱۹. متد Asin

آرک سینوس عدد x را محاسبه می نماید.

۲۰. متد Atan

آرک تانژانت عدد x را محاسبه می نماید.

۲۱. متد Tan

تانژانت عدد x را محاسبه می نماید. (x بر حسب رادیان)

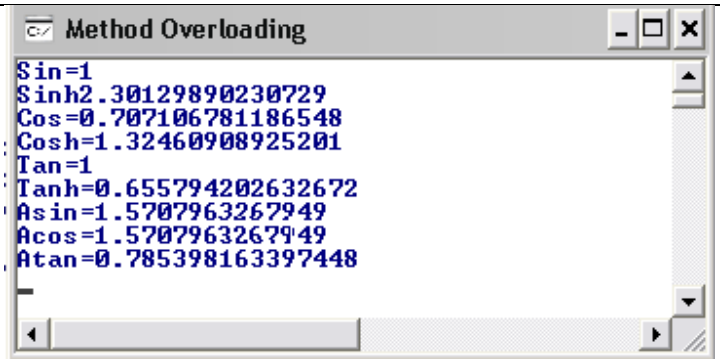
۲۲. متد Tanh

تانژانت هایپربولیک عدد x را محاسبه می نماید. (x بر حسب رادیان)

چگونگی استفاده از توابع مثلثاتی در برنامه زیر نشان داده شده است.

```
using System;
class MainClass
{
	static void Main()
	{
		Console.WriteLine("Sin="+Math.Sin(Math.PI/2));
		Console.WriteLine("Sinh"+Math.Sinh(Math.PI/2));
		Console.WriteLine("Cos="+Math.Cos(Math.PI/4));
		Console.WriteLine("Cosh="+Math.Cosh(Math.PI/4));
		Console.WriteLine("Tan="+Math.Tan(Math.PI/4));
		Console.WriteLine("Tanh="+Math.Tanh(Math.PI/4));
		Console.WriteLine("Asin="+Math.Asin(1));
		Console.WriteLine("Acos="+Math.Acos(0));
		Console.WriteLine("Atan="+Math.Atan(1));
		Console.ReadLine();
	}
}
```

خروجی برنامه فوق مطابق شکل زیر خواهد بود.



۵. متدهای تولید اعداد تصادفی

اعداد تصادفی نقش مهمی در برنامه نویسی دارند. بعنوان مثال برای تولید داده های مورد نیاز جهت تست الگوریتمها می توان از این اعداد استفاده نمود. اغلب زبانها مکانیزمهایی برای تولید اعداد تصادفی دارند.

قابل ذکر است که این اعداد دقیقاً تصادفی نبوده بلکه با توجه به روابط ریاضی تولید می گردد و اغلب به آنها داده های شبه تصادفی گفته می شود.

در C# از کلاس Random جهت تولید اعداد تصادفی استفاده می شود. این کلاس حاوی متدهایی برای تولید اعداد تصادفی است که در زیر به این متدها و چگونگی استفاده از آنها پرداخته می شود.

برای تولید اعداد تصادفی ابتدا باید متغیری از نوع کلاس Random ایجاد نموده سپس از این متغیر جهت اجرای متدهای تولید اعداد تصادفی بهره ببریم. (چون متدها استاتیک نیستند)

چگونگی تعریف متغیر به یکی از دو شکل زیر می باشد:

```
Random  نام متغیر = new Random();
Random  نام متغیر = new Random(int seed);
```

در حالت اول فقط متغیر تصادفی ایجاد می گردد ولی در حالت دوم هستهٔ تولید اعداد تصادفی برای ایجاد مقادیر تصادفی نیز مشخص می شود.

اگر اقدام به تولید اعداد تصادفی چندین بار با هسته یکسان نماییم، اعداد ایجاد شده در اجراهای مختلف یکسان خواهند بود ولی در حالتیکه هسته مشخص نباشد از زمان سیستم جهت تولید هسته استفاده می شود. و با توجه به تغیر زمان سیستم، هسته بکار برده شده متفاوت بوده بنابراین دنباله تولید شده اعداد تصادفی متفاوت خواهد بود.

```
Random rnd = new Random();
```

جهت ایجاد یک عدد تصادفی می توان از متد Next استفاده نمود.

```
int x=rnd.Next();
```

در این حالت x عددی تصادفی در بازهٔ <=x<=int.MaxValue خواهد بود.

اگر بخواهیم عدد تولید شده در بازهٔ خاصی باشد می توان از متد فوق بصورت زیر استفاده نمود:

```
int x=rnd.Next (k);
```

در این حالت عدد تولید شده در بازهٔ صفر تا k-1 می باشد.

همچنین می توان تعیین نمود که عدد تولید شده در بازهٔ [a,b) باشد که برای این منظور از متد فوق بصورت زیر استفاده می شود:

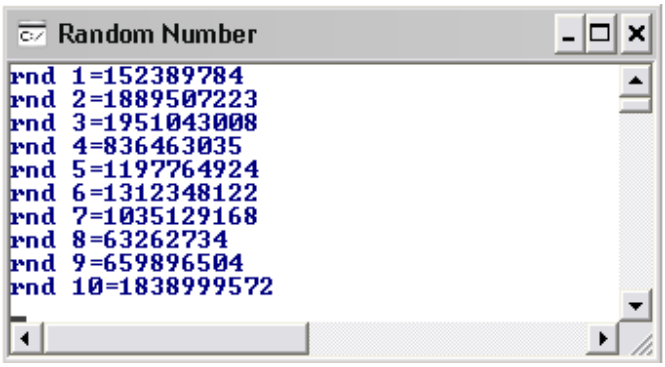
```
int x=rnd.Next(a,b);
```

در زیر چند مثال از نحوهٔ استفادهٔ متد فوق ارائه شده است.

مثال۱۶) بدون تعیین هسته ده اعداد تصادفی ایجاد نمایید.

```
using System;
class MainClass
{
	static void Main()
	{
		Random rnd= new Random();
		int x=rnd.Next();
		for (int i=1; i<=10; i++)
		{
			Console.WriteLine("rnd {0}={1}",i,x);
			x=rnd.Next();
		}
		Console.ReadLine();
	}
}
```

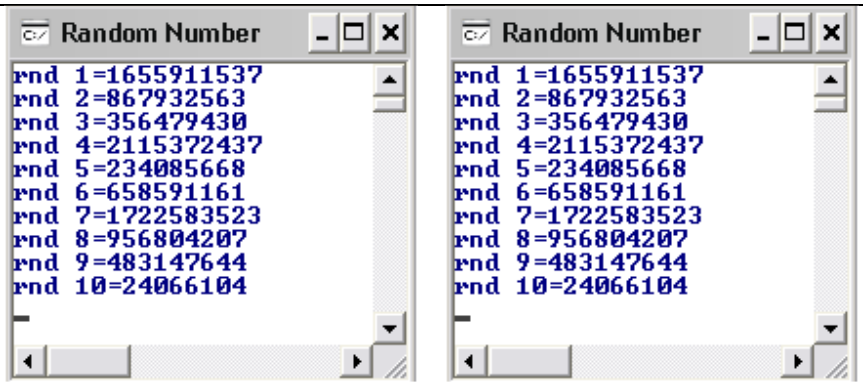
در این حالت ده عدد تصادفی تولید می شود که هسته اولیه آنها نیز با توجه به زمان سیستم تعین گردیده است. شکل زیر نمونه ای از خروجی برنامه فوق می باشد. چنانچه یکبار دیگر برنامه فوق را اجرا نماییم، دنباله اعداد متفاوت از این خروجی خواهد بود زیرا هسته تولیدی اعداد بعدی متفاوت می باشد.



مثال۱۷) ده عدد تصادفی با هسته ۲ ایجاد نمایید.

```
using System;
class MainClass
{
	static void Main()
	{
		Random rnd= new Random(2);
		int x=rnd.Next();
		for (int i=1; i<=10; i++)
		{
			Console.WriteLine("rnd {0}={1}",i,x);
			x=rnd.Next();
		}
		Console.ReadLine();
	}
}
```

در این حالت ده عدد تصادفی تولید می شود که هسته اولیه آنها برابر ۲ می باشد. شکل زیر نمونه هایی از خروجی برنامه فوق می باشد. چنانچه یکبار دیگر برنامه فوق را اجرا نماییم، دنباله اعداد تولید شده با این خروجی یکسان خواهد بود زیرا هسته تولیدی اعداد بعدی نیز ۲ می باشد.



مثال۱۸) ده عدد تصادفی در بازهٔ صفر تا ۵ تولید نمایید.

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

		Random rnd= new Random();
		int x=rnd.Next(6);
		for (int i=1; i<=10; i++)
		{
			Console.WriteLine("rnd {0}={1}",i,x);
			x=rnd.Next(6);
		}
		Console.ReadLine();
	}
}
```

در برنامه فوق ۱۰ عدد تصادفی در بازهٔ صفر تا پنج تولید می شود که دو سری از این اعداد در زیر لیست شده اند.

مثال۱۹) ده عدد تصادفی در بازهٔ ۱۰ تا ۲۰ تولید نمایید.

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

		Random rnd= new Random();
		int x=rnd.Next(10,21);
		for (int i=1; i<=10; i++)
```

```
            {
                  Console.WriteLine("rnd {0}={1}",i,x);
                  x=rnd.Next(10,21);
            }
            Console.ReadLine();
      }
}
```

این مثال مشابه مثال قبل می باشد با این تفاوت که پارامترهای ورودی متد Next بجای یک عدد صحیح دو عدد صحیح شده است. در نتیجه بازه اعداد تولید شده بین این دو عدد خواهند بود.

مثال۲۰) بازی شانسی کراپز[3]

در بازی کراپز دو عدد تاس شش وجهی وجود دارد که روی هر یک از این وجوه ۱،۲،۳،۴،۵و۶ نقطه وجود دارد. پس از ریختن تاسها، مجموع نقاط روی دو وجهی که رو به بالا قرار گرفته اند محاسبه می شود. اگر این مجموع در اولین بار عدد ۷ و یا ۱۱ بیاید، بازیکن برنده است. اما اگر مجموع در اولین بار ریختن تاس ۲، ۳ و یا ۱۲ بیاید، بازیکن بازنده است(که این حالت کراپز شدن نام دارد). در غیر اینصورت چنانچه مجموع در اولین بار ریختن تاس یکی از اعداد ۴ ، ۵ ، ۶ ، ۸ ، ۹ ، ۱۰ بیاید، مجموع بدست آمده امتیاز بازیکن محسوب می شود. بازیکن برای برنده شدن باید تا بدست آوردن این امتیاز به بازی ادامه دهد. اما اگر پیش از بدست آوردن امتیاز مشخص شده، یک مجموع ۷ بیاید بازیکن بازنده می شود.

شرح برنامه:

در بازی فوق ابتدا باید دفعه اول ریختن تاس و دفعه های بعدی را از همدیگر جدا کرد. زیرا شرایط برد و باخت در دو حالت با همدیگر متفاوت می باشد. همچنین متدی می نویسیم که در هر بار فراخوانی، دو عدد تصادفی در بازهٔ [1,6] تولید نموده و مجموع آن دو را به ما برگرداند و در برنامه اصلی از این متد استفاده می نماییم. همچنین از یک متد جهت تنظیم رنگ پس زمینه، رنگ قلم و عنوان کنسول استفاده شده است.
کد برنامه در زیر آمده است:

```
using System;
class MainClass
{
      static void Main()
      {
            byte sum , point=0;
            string flag;
            Random rnd=new Random();
            Start ( " Craps Game" );
            sum=RollDice(rnd);
            switch (sum)
            {
                  case 7:
                  case 11:
                        flag="Won";
                        break;
                  case 2:
                  case 3:
                  case 12:
                        flag="Loss";
                        break;
                  default:
```

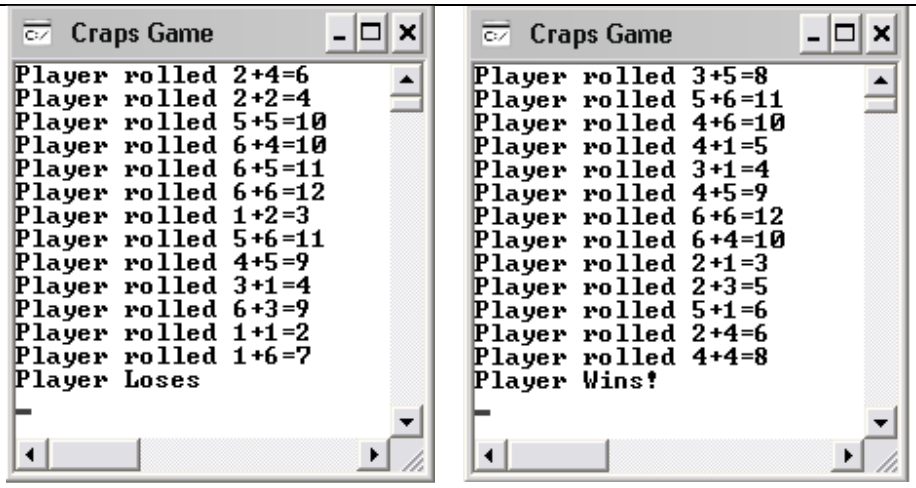
[3] craps

```
                        flag="Continue";
                        point=sum;
                        break;
                }
                while ( flag.Equals("Continue"))
                {
                        sum=RollDice (rnd);
                        if ( sum == point)
                                flag="Won";
                        else if (sum == 7 )
                                flag="Loss";
                }
                if ( flag.Equals("Won"))
                        Console.WriteLine("Player Wins!");
                else
                        Console.WriteLine("Player Loses");
                Console.ReadLine();
        }
        static byte RollDice(Random rnd)
        {
                byte  rnd1, rnd2 , sum;
                rnd1=Convert.ToByte(rnd.Next(1,7)); // or Cast
                rnd2=Convert.ToByte(rnd.Next(1,7));
                sum= ( byte ) (rnd1 + rnd2) ;
                Console.WriteLine("Player rolled "+rnd1 +"+"+rnd2+"="+sum );
                return sum;
        }
        static void Start (string s)
        {
                Console.BackgroundColor=ConsoleColor.White;
                Console.ForegroundColor=ConsoleColor.Black;
                Console.Title=s;
                Console.Clear ( );
        }
}
```

متد RollDice در هر بار فراخوانی دو مقدار تصادفی تولید می نماید و چون نتیجه متد Next مقداری از نوع int می باشد، ابتدا این مقدار به نوع byte تبدیل می شود. سپس مجموع دو عدد نیز پس از تبدیل به byte به متد فراخواننده برگردانده می شود. علت اینکه اعداد تصادفی و مجموع آنها از نوع byte درنظر گرفته شده اند، کوچک بودن خود اعداد و درنتیجه مجموع آنها می باشد. (زیرا اعداد کوچکتر یا مساوی ۶ می باشند.)

برای نگهداری وضعیت بازیکن که آیا برنده شده است و یا بازنده و یا اینکه هنوز وضعیت وی مشخص نشده است و باید به بازی ادامه دهد از متغیر رشته ای flag استفاده شده است که مقدار آن می تواند Won ، Loss و یا Continue باشد. مقدار اولیه flag توسط ساختار switch بعد از بار اول ریختن تاس، مشخص می گردد. پس از آن تا زمانی که وضعیت برابر مقدار Continue باشد حلقه تکرار انجام گرفته و تاس جدیدی را می ریزد و به کمک دستور if بکار رفته چک می نماید که آیا بازیکن باید ادامه دهد یا اینکه برنده و یا بازنده شده است و از ادامه بازی وی جلوگیری به عمل آید.

در شکل زیر دو نمونه از اجرای برنامه نشان داده شده است که در یک حالت بازیکن برنده و در حالت دیگر بازنده شده است.



تمرینات فصل متدها

۱. برنامه ای بنویسید که مجموع اعداد اول مقلوب پذیر چهار رقمی را محاسبه نماید. در این برنامه حداقل از دو متد استفاده نمایید. از یک متد به نام `IsPrime` جهت تشخیص اول بودن یک عدد و از متد دیگر به نام `IsPalindrome` جهت تشخیص مقلوب پذیر بودن یک عدد.

توجه: در حل تمرینات ۲ و ۳ تعیین متدها به عهده برنامه نویس می باشد.

۲. برنامه ای بصورت زیر بنویسید تا به کاربر در حدس یک عدد تصادفی در نظر گرفته شده توسط کامپیوتر کمک نماید. عملکرد برنامه باید بصورت زیر باشد:

- ✓ ابتدا کاربر باید بازهٔ اعداد را با وارد کردن عدد `n` بعنوان ورودی مشخص نماید.(بنابراین بازهٔ اعداد `[1,n]` خواهد بود.)
- ✓ در مرحلهٔ بعد برنامه یک عدد تصادفی در بازهٔ فوق در نظر می گیرد.
- ✓ پس از آن برنامه پیغام زیر را به کاربر نشان می دهد تا کاربر حدس خود را وارد نماید

```
I have a number between 1 and n
Can you guess my number?
Please type your first guess-?
```

- ✓ بعد از نمایش پیغام فوق کاربر حدس خود را وارد می کند و برنامه با یکی از پیغامهای زیر وی را راهنمایی می کند:

```
1. Excellent! You guess the number
   Would you like to play again (y or n)?
2. Too low. Try again.
3. Too high. Try again.
```

- ✓ اگر حدس نادرست باشد برنامه باید آنقدر مرحلهٔ فوق را تکرار نماید تا کاربر بتواند جواب صحیح را حدس بزند.
- ✓ در انتها برنامه باید تعداد دفعات حدس عدد توسط کاربر را بشمارد و با توجه به آن یکی از سه پیغام زیر را چاپ نماید:

اگر تعداد دفعات حدس کمتر از ۱۰ بود پیغام یک و اگر برابر ۱۰ بود پیغام دو و اگر بیشتر از ۱۰ بود پیغام سه را نمایش دهد.

```
1. Either you know the secret or you got lucky?
2. Ahah !You know the secret
3. You should be able to do better!
```

۳. برنامه ای بنویسید تا به دانش آموزان در یادگیری عملیات پایه ریاضی (+، -، × ، / ، ٪) کمک نماید. چگونگی انجام این کار توسط برنامه در زیر تشریح شده است:

- ✓ برنامه در ابتدا باید با تولید یک عدد تصادفی تعیین نماید که کدام عمل را می خواهد به کاربر آموزش دهد. (عددی بین ۱ تا ۵ تولید و بر اساس آن یکی از پنج عمل تعیین شده انتخاب گردد)
- ✓ بعد از انتخاب نوع عمل برنامه باید سوال را بر اساس نوع عمل طرح نماید. که سوال می تواند یکی از حالات زیر باشد که با توجه به مرحله قبل تعیین می گردد:
  - ✓ `a + b = ?`
  - ✓ `a - b = ?`
  - ✓ `a * b = ?`
  - ✓ `a / b = ?`
  - ✓ `a % b = ?`
- ✓ بعد از مشاهده سوال کاربر جواب را وارد می کند و برنامه با توجه به درست ویا نادرست بودن جواب پیغام مناسبی را چاپ می نماید. دقت شود چنانچه پیغام همیشه یکسان باشد، برای کاربر خسته کننده خواهد بود به همین دلیل بهتر است تا برنامه لیستی از پیغامها را داشته باشد و هر بار بصورت تصادفی یکی از آنها را نمایش دهد.
- ✓ لیست پیغامها در صورتیکه پاسخ درست باشد.
  - ✓ `Very good!`
  - ✓ `Excellent!`
  - ✓ `Nice work!`
  - ✓ `Keep up the good work!`
- ✓ لیست پیغامها در صورتیکه پاسخ نادرست باشد.
  - ✓ `No. Please try again.`
  - ✓ `Wrong. Try once more.`
  - ✓ `Don't give up!`
  - ✓ `No. Keep trying.`

- ✓ همچنین برنامه باید تعداد پاسخهای درست و نادرست را بشمارد . پس از آنکه دانش آموز n سوال را پاسخ داد (n در ابتدای برنامه از کاربر دریافت می شود) ، برنامه باید درصد پاسخهای درست را محاسبه نماید و در صورتیکه درصد کمتر از ۷۵٪ بود پیغام زیر را چاپ نماید. در غیر اینصورت پیغام مناسبی را به کاربر نمایش دهد.

`Please ask your instructor for extra help!`

- ✓ در برنامه فوق اعداد a و b دو عدد تصادفی تک رقمی می باشند که توسط برنامه تولید می شوند.
- ✓ به برنامه فوق این قابلیت را اضافه نمایید که در ابتدای برنامه با دریافت عددی از کاربر سطح آموزش مشخص گردد. مثلاً چنانچه کاربر عدد ۱ را وارد نماید، مقادیر a و b تک رقمی باشند و چنانچه ۲ بعنوان ورودی وارد شود، مقادیر a و b دو رقمی باشند.

وما توفیقی الا بالله

بلوچ زهی ۸۵/۷/۳۰

فصل ششم: طبقه بندی انواع

در C# انواع متفاوتی وجود دارد که می توان کلیهٔ این انواع را به دو گروه تقسیم نمود:

الف- انواع مقداری[1]

ب- انواع ارجاعی[2]

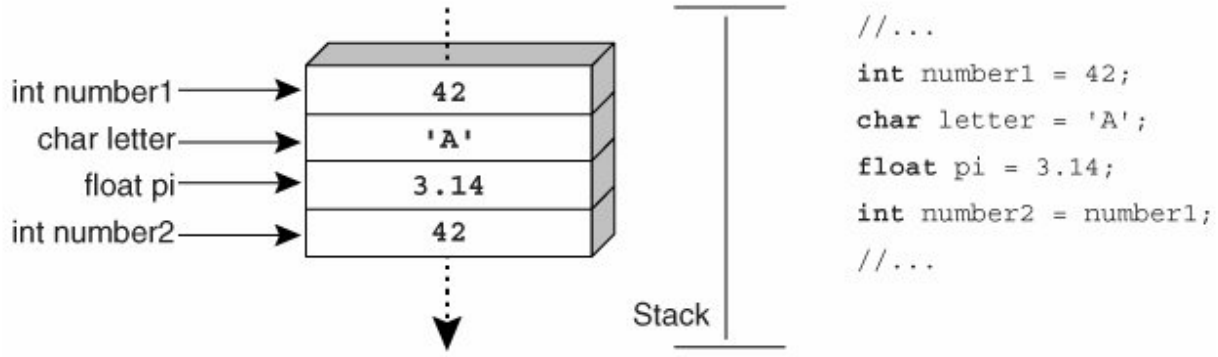
تفاوت اساسی انواع موجود در این است که هر گروه مکان متفاوتی از حافظه را جهت ذخیره سازی مورد استفاده قرار می دهد. به اینصورت که مقادیر انواع مقداری در پشته[3] ذخیره می شوند درحالیکه انواع ارجاعی جهت ذخیره مقادیر خود از هرم[4] بهره می برند.

الف- انواع مقداری

کلیه انواع پایه از پیش تعریف شده در C# بجز رشته[5] ها از نوع مقداری می باشند. این انواع مقادیر را در خود نگهداری می کنند. به فرض زمانی که متغیری مانند x از نوع مقداری int تعریف می گردد، x نام متغیری است که یک مقدار عددی صحیح را در خود نگهداری می کند. حال چنانچه متغیری به نام y از نوع int داشته باشیم، و متغیر x را به y نسبت دهیم، در این حالت متغیر y همان مقدار x را دارا می باشد. در چنین شرایطی دو کپی از مقدار x در حافظه داریم، یکی در خانه ای به نام x و دیگری در خانه ای به نام y.

به عنوان یک مثال دیگر به شکل زیر دقت نمایید، در این شکل چهار متغیر مقداری تعریف شده است، سپس مقدار متغیر number1 به متغیر number2 نسبت داده شده است.

همانگونه که مشاهده می نمایید، مقدار ۴۲ در دو مکان از حافظه ذخیره شده است، یکی در خانه ای به نام number1 و دیگری در خانه ای به نام number2.



مشابه حالت فوق چنانچه متغیری از نوع مقداری به عنوان پارامتر به یک متد ارسال گردد، مقدار آرگومان در پارامتر متد کپی می شود و در این حالت نیز دو کپی از مقدار وجود دارد. درنتیجه تغیرات انجام شده بر روی مقدار پارامتر در متد، بر روی آرگومان تاثیری نخواهد داشت. زیرا آرگومان و پارامتر دو مکان متفاوت از حافظه بوده و تغیر مقدار هر کدام تاثیری بر دیگری نمی گذارد.

با توجه به اینکه پارامترهای نوع مقداری نیاز به عمل کپی حافظه دارند، توصیه می شود فقط جهت ارسال اندازهٔ کوچکی از حافظه از این نوع پارامترها استفاده گردد. (با توجه به زمانبر بودن عمل کپی)

---

[1] Value Types
[2] Reference Types
[3] Stack
[4] Heap
[5] string

متغیرهای نوع مقداری در حافظه ای به نام پشته یا stack ذخیره می شوند ولی محل ذخیره سازی انواع ارجاعی حافظه ای به نام heap می باشد. در زیر به معرفی مختصر این دو حافظه پرداخته می شود.

پشته (Stack) و هرم (Heap)

حافظه کامپیوتر از نظر منطقی به تعدادی قطعه جداگانه به نام سگمنت تقسیم می شود که دو تا از مهمترین قطعات عبارتند از پشته و هرم. این دو قطعه برای اهداف متفاوتی درنظر گرفته شده اند.

کاربرد پشته

وقتی که متدی فراخوانی می شود، پارامترها و متغیرهای محلی آن احتیاج به حافظه دارند. این حافظه از فضای پشته تامین می گردد. زمانیکه کار متد به پایان رسید، این حافظه اختصاص داده شده بصورت خودکار به پشته برگردانده می شود.

کاربرد هرم

وقتی که به کمک دستور new برای یک نوع ارجاعی حافظه درخواست می شود، این حافظه از هرم تامین می شود. هر گاه کار متد به پایان می رسد، این حافظه بصورت خودکار به هرم برگردانده می شود.

ب- انواع ارجاعی

فصل دهم: فایلها

تاکنون کلیهٔ داده های مورد نیاز برنامه ها در متغیرها (از هر نوعی مانند آرایه، ساختار وغیره) ذخیره می شدند که این داده ها تا زمان پایان یافتن اجرای برنامه، در دسترس بودند و با اتمام اجرای برنامه این داده ها از حافظه سیستم پاک می شدند. علت اصلی این مساله این است که داده های مورد نیاز برنامه که در طول اجرای برنامه به آنها نیاز داشتیم در Ram ذخیره می شوند و با پایان یافتن اجرای برنامه و حداکثر با راه اندازی مجدد برنامه این داده ها از بین می‌روند و نیاز به ورود مجدد داده ها داریم.

با بالا رفتن حجم داده های موردنیاز برنامه، ورود مجدد داده ها در هر بار اجرای برنامه عملی زمانبر خواهد بود به همین دلیل معمولاً داده های مورد نیاز برنامه بر روی حافظه ثانویه مانند هارد ذخیره می شوند تا در فراخوانیهای مجدد برنامه نیاز به ورود آنها توسط کاربر نباشد، بلکه خود برنامه آنها را از حافظه ثانویه بازیابی نماید.

انواع فایلها در C#

فایلهای مورد استفاده در C# به دو بخش تقسیم می شوند که عبارتند از:

الف- فایلهای متنی (text files)

ب- فایلهای دودویی (binary files)

فایلهای متنی فایلهایی هستند که محتویات آنها بصورت کارکتری ذخیره می شود. مثلاً چنانچه عدد ۴۳۷ را در یک فایل متنی ذخیره نماییم بصورت سه کارکتر '3' ، '4' و '7' ذخیر خواهد شد که چون هر کارکتر نیاز به دو بایت حافظه دارد، لذا حافظه مورد نیاز برابر خواهد بود با ۶ بایت.

فایلهایی که برای ذخیرهٔ داده های نوع دار مانند اعداد صحیح، اعداد اعشاری، ساختارها و غیره بکار می روند، فایلهای دودویی نام دارند. در این فایلها داده ها به همان شیوه که در Ram ذخیره می شوند، در آنها نیز ذخیره می شوند. پس یک عدد صحیح در این فایلها همیشه چهار بایت حافظه اشغال می کند. لذا عدد ۴۳۷ نیاز به ۴ بایت حافظه دارد.

نحوه ذخیره سازی داده ها در فایلهای متنی مشابه صفحه نمایش می باشد، در حالیکه برای فایلهای دودویی مشابه حافظه اصلی (Ram) است.

دایرکتوری و فایل

دوکلاس استاتیک که حاوی مجموعه ای از متدهای استاتیک می باشند، جهت انجام عملیات پایه بر روی فایلها و دایرکتوریها وجود دارند، این عملیات عبارتند از:

ایجاد ، حذف، انتقال، حذف و بقیه اعمالی که می توان بر روی فایلها و یا دایرکتوریها انجام داد.

این دو کلاس استاتیک به ترتیب عبارتند از Directory و File.

هر کدام از این دو کلاس دارای مجموعه کاملی از متدها جهت کار با فایلها و دایرکتوریها هستند که در زیر به پاره ای از مهمترین متدهای درون این کلاسها پرداخته می شود.

فصل دهم: فایلها

متدهای کلاس دایرکتوری:

نام متد	عملکرد
CreateDirectory	جهت ایجاد دایرکتوری جدید بکار می‌رود.
Exists	چک می نماید که آیا دایرکتوری با آدرس مشخص شده، وجود دارد یا خیر؟
Delete	جهت حذف یک دایرکتوری بکار می رود.
GetDirectories	لیست کلیهٔ زیردایرکتوریهای درون دایرکتوری داده شده را بر می گرداند.
GetDirectoryRoot	دایرکتوری ریشهٔ متعلق به دایرکتوری داده شده را برمی گرداند.
GetFiles	لیست کلیهٔ فایلهای درون دایرکتوری داده شده را بر می گرداند.

متدهای کلاس فایل:

نام متد	عملکرد
Create	جهت ایجاد فایل جدید بکار می‌رود.
Exists	چک می نماید که آیا فایل با آدرس مشخص شده، وجود دارد یا خیر؟
Delete	جهت حذف یک فایل بکار می رود.
Copy	جهت کپی نمودن فایل از یک مکان به مکان دیگر مورد استفاده واقع می شود.
Move	جهت انتقال فایل از یک مکان به مکان دیگر بکار می رود.

جهت پی بردن به چگونگی عملکرد متدهای فوق به مثالهای زیر توجه نمایید:

مثال۱) مجموعه دستورات زیر چه عملی انجام می دهند؟

```
string path1 = @"c:\nik\C#\file";
string path2 = @"c:\files";
Directory.CreateDirectory(path1);
Directory.CreateDirectory(path2);
```

هر کدام از دستورات اول و دوم فقط دو متغیر path1 و path2 را مقداردهی اولیه می نمایند.

دستور سوم در مسیر یک دایرکتوری خواسته شده را ایجاد می نماید. که برای انجام این کار از ریشهٔ مسیر شروع به جستجو می نماید، اگر nik در c: وجود نداشته باشد، ابتدا آنرا ایجاد می نماید و سپس برای بقیه مسیر یعنی C# و file نیز به همین ترتیب عمل می کند. در انتهای این دستور کل شاخه مورد نظر در درایو C: ایجاد می گردد.

دستور چهارم نیز به همین ترتیب فقط دایرکتوری خواسته شده را در مسیر مشخص شده ایجاد می نماید.

اگر دایرکتوری در مسیر مشخص شده وجود داشته باشد، هیچ مشکلی ایجاد نمی شود و زیردایرکتوریها و فایلهای آن بدون تغییر باقی می مانند.

مثال۲) دستورات فوق را بگونه ای تغییر دهید که قبل از ایجاد دایرکتوری بررسی نماید که آیا دایرکتوری موردنظر وجود دارد یا خیر؟ در صورت عدم وجود آنرا ایجاد نماید.

```
string path1 = @"c:\nik\DotNet";
if (!Directory.Exists(path1))
   {
     Directory.CreateDirectory(path1);
     Console.WriteLine("Directory Created");
   }
else
   Console.WriteLine("Directory is Already Exist");
```

در مجموعه دستورات فوق ابتدا چک می شود که آیا دایرکتوری موردنظر در مسیر مشخص شده وجو دارد یا خیر؟ در صورت عدم وجود دایرکتوری ایجاد می گردد و پیغام ایجاد آن به کاربر داده می شود. اما اگر دایرکتوری از قبل در مسیر مشخص شده موجود باشد، دوباره ایجاد نمی گردد و فقط به کاربر پیغام داده می شود که دایرکتوری از قبل در مسیر موردنظر وجود دارد.

مثال۳) کلیه دایرکتوریهای مسیر داده شده را حذف نمایید.

با توجه به اینکه هر کدام از زیر دایرکتوریها می توانند خالی بوده و یا اینکه خود شامل تعدادی زیردایرکتوری و فایل باشند، پس در استفاده از دستور Delete باید دقت نمود:

```
Directory.Delete( مسیر );
```

این دستور فقط دایرکتوریهای خالی را حذف می نماید و درصورتیکه دایرکتوری خالی نباشد، تولید Exception می نماید.

که می توان پارامتر بولی دومی نیز با آن ارسال نمود تا درصورتیکه مقدار پارامتر بولی true باشد، بتواند دایرکتوریهایی را که خالی نیستند نیز حذف نماید.

```
Directory.Delete( مسیر , مقدار بولی);
```

```
string path1 = @"c:\nik";
foreach (string s in Directory.GetDirectories(path1))
      Directory.Delete(s, true);
```

در این مثال از دستور GetDirectories نیز استفاده شده است که لیست کلیه زیر دایرکتوریها را بصورت آرایه ای از رشته ها به ما برمی گرداند. سپس به کمک دستور Delete کلیهٔ دایرکتوریها حذف شده اند.

دستور GetDirectories را می توان بصورت زیر مورد استفاده قرار داد:

```
string []dir=Directory.GetDirectories( مسیر );
```

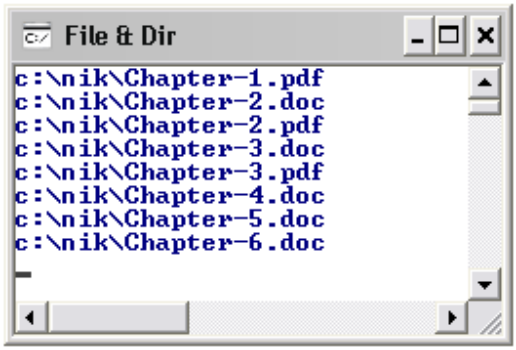
که در این حالت لیست کلیه زیر دایرکتوریهای درون مسیر موردنظر در آرایه dir قرار می گیرند.

مثال۴) کلیه فایلهای درون مسیر داده شده را نمایش دهید.

```
using System;
using System.IO;
namespace StaticMethodFileDir
{
    class Program
    {
        static void Main(string[] args)
        {
            Start();
```

```
            string path1 = @"c:\nik";
            string path2 = @"c:\nikCopy";
            foreach (string s in Directory.GetFiles(path1))
                Console.WriteLine(s);
            Console.ReadLine();
        }
        static void Start()
        {
            Console.BackgroundColor = ConsoleColor.White;
            Console.ForegroundColor = ConsoleColor.DarkBlue;
            Console.Title = "File & Dir";
            Console.Clear();

        }
    }
}
```

خروجی برنامه فوق بصورت زیر خواهد بود:



مثال۵) کلیه فایلهای درون مسیر داده شده (path1) را به مسیر مقصد (path2) کپی نماید.

```
string path1 = @"c:\nik";
string path2 = @"c:\nikCopy";
Directory.CreateDirectory(path2);
foreach (string s in Directory.GetFiles(path1))
     {
          string dest =path2+  s.Substring(s.LastIndexOf("\\"));
          File.Copy(s, dest);
     }
```

در مجموعه دستورات فوق ابتدا دایرکتوری مقصد ایجاد می شود، سپس کلیه فایلهای درون مسیر مبداء بازیابی شده و پس از تنظیم مسیر مقصد به ازای هر فایل (حاوی مسیر داده شده و نام فایل در دایرکتوری مقصد) ، فایل موردنظر در دایرکتوری مقصد کپی می شود.

توجه۱: هنگام کپی نمودن فایل چنانچه فایل از قبل در مقصد وجود داشته باشد، یک پیغام خطا دریافت خواهید کرد. برای رفع مشکل فوق می توان از متد Copy همراه با یک متغیر بولی استفاده نمود که اگر مقدار متغیر بولی true باشد، دراینصورت فایل قبلی با فایل جدید overwrite خواهد شد.

```
File.Copy(مبدا , مقصد , متغیر بولی );
```

به عنوان مثال چنانچه دستور زیر را اجرا نماییم، بدون هیچ اخطار و یا خطایی اگر فایل f1.txt در دایرکتوری nik وجود داشته باشد، بازنویسی خواهد شد.

```
File.Copy ( "c:\f1.txt", "c:\nik\f1.txt", true);
```

فصل دهم: فایلها

چگونگی استفاده از متد Move مشابه متد Copy می باشد.

مثال۶) متدی بنویسید که با دریافت مسیرهای مبداء و مقصد کلیه دایرکتوریها و فایلهای درون مسیر مبداء را در مسیر مقصد کپی نماید.

```
static void Backup(string source, string dest)
  {
    if(!Directory.Exists(dest))
          Directory.CreateDirectory(dest);
    foreach (string file in Directory.GetFiles(source))
         File.Copy(file, dest + file.Substring(file.LastIndexOf("\\")));
    string[] dir = Directory.GetDirectories(source);
    if (dir.Length == 0)
        return;
    for (int i = 0; i < dir.Length; i++)
      {
         string d=dest+dir[i].Substring(dir[i].LastIndexOf("\\"));
         Directory.CreateDirectory(d );
         Backup(dir[i], d);
       }
    }
```

تمرین۱: برنامه ای بنویسید که با دریافت مسیرهای مبدا و مقصد کلیه فایلها و دایرکتوریها را از مبداء به مقصد انتقال دهد.

تمرین۲: برنامه ای بنویسید که با دریافت مسیر پسوند کلیه فایلهای درون دایرکتوری را به CS تغیر دهد.

فایلهای متنی

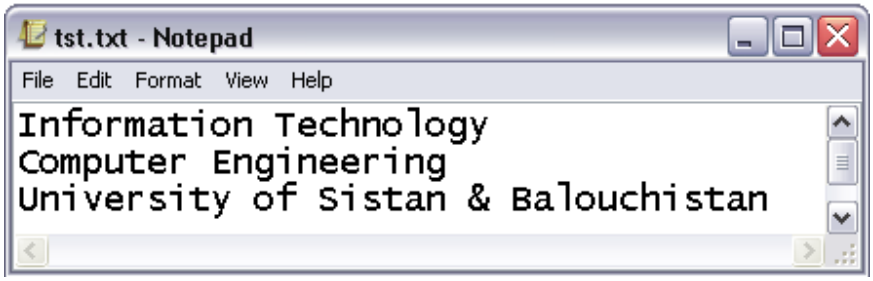
در این بخش به چگونگی کار با فایلهای متنی پرداخته می شود. مسائلی از قبیل اینکه چگونه داده های موردنظر را در فایلهای متنی ذخیره نماییم، همچنین چگونگی بازیابی اطلاعات از فایلهای متنی مورد بحث قرار خواهند گرفت.

برای کار با فایلهای متنی از دو کلاس StreamReader و StreamWriter استفاده می شود. از کلاس اولی جهت خواندن اطلاعات از فایلهای متنی و برای نوشتن در فایلهای متنی از کلاس دوم استفاده می شود.

مثال۷) مجموعه دستورات زیر فایل متنی tst.txt را در مسیر c:\nik ایجاد نموده سپس رشته های InFormation Technology، Software Engineering، University of Sistan & Balouchistan را در فایل ایجاد شده می نویسید.

```
StreamWriter sw=new StreamWriter("c:\\nik\\tst.txt");
sw.WriteLine("Information Technology");
sw.WriteLine("Computer Engineering");
sw.WriteLine("University of Sistan & Balouchistan");
sw.Close();
```

محتویات فایل ایجاد شده بصورت زیر می باشد.



فصل دهم: فایلها

همانگونه که مشاهده می شود از سازندهٔ کلاس StreamWriter جهت ایجاد فایل متنی استفاده می شود. سپس از متد WriteLine جهت اضافه نمودن اطلاعات به فایل ایجاد شده بهره برده ایم. در انتها فایل باز شده را بسته ایم تا اطلاعاتی که هنوز در بافر بوده و به فایل منتقل نشده اند، بطور کامل به فایل منتقل شوند.

۱. با توجه به خروجی دیده می شود که نحوه ذخیره شدن اطلاعات در فایل متنی مشابه صفحه مانیتور می باشد.
۲. جهت نوشتن اطلاعات در فایل متنی علاوه بر متد WriteLine می توان از متد Write نیز استفاده نمود.که در این حالت اطلاعات پشت سرهم در فایل نوشته می شوند.
۳. اگر برنامه فوق را چندین بار اجرا نماییم، سازنده کلاس فایل tst.txt را دوباره ایجاد می نماید و اطلاعات قبلی آن از بین می روند.

اگر بخواهیم اطلاعات جدیدی به انتهای فایل متنی اضافه نماییم، می توانیم از متغیر بولی که برای این منظور درنظر گرفته شده است استفاده نمود و اطلاعات را به انتهای فایل متنی اضافه نمود.

```
StreamWriter sw=new StreamWriter("c:\\nik\\tst.txt", true);
sw.WriteLine("Information Technology");
sw.WriteLine("Computer Engineering");
sw.WriteLine("University of Sistan & Balouchistan");
sw.Close();
```

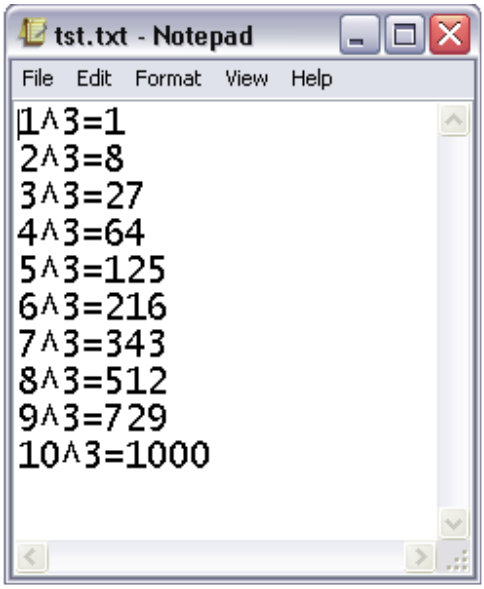
در این حالت اطلاعات با توجه به اینکه مقدار متغیر بولی برابر true می باشد، در نتیجه اطلاعات به انتهای فایل اضافه شده و محتویات قبلی فایل از بین نمی رود.

توجه: می توان اطلاعات را بصورت فرمت بندی شده نیز در فایل ذخیره نمود که نمونه ای از آن در مثال زیر آورده شده است.

مثال۸) چگونگی ذخیره سازی اطلاعات بصورت فرمت بندی شده در فایل متنی.

```
StreamWriter sw=new StreamWriter("c:\\nik\\tst.txt");
for (int i = 1; i <= 10; i++)
     sw.WriteLine("{0}^3={1}", i, i * i * i);
sw.Close();
```

فایل خروجی بصورت زیر خواهد بود:



فصل دهم: فایلها

برای خواندن اطلاعات از فایلهای متنی می توان از متدهای کلاس StreamReader استفاده نمود. در زیر متدها و خاصیت های کلاس StreamReader ذکر شده اند.

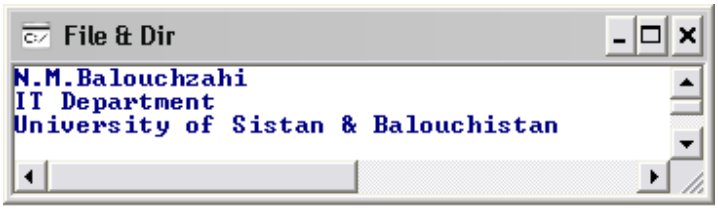
نام متد یا خاصیت	عملکرد
ReadLine	جهت خواندن یک خط از فایل بکار می‌رود.
Read	جهت خواندن یک کارکتر از فایل بکار می‌رود.
ReadToEnd	از مکان جاری تا انتهای فایل را می خواند.
EndOfStream	از این خاصیت جهت تشخیص انتهای فایل سود می بریم.

مثال۹) برنامه ای بنویسید که اطلاعات فایل متنی tst.txt را خوانده و به همان صورت در خروجی چاپ نماید.

```
StreamReader sr = new StreamReader(@"C:\nik\tst.txt");
while (!sr.EndOfStream)
   Console.WriteLine(sr.ReadLine());
sr.Close();
```

با اجرای مجموعه دستورات فوق فایل مورد نظر برای خواندن باز می شود. سپس تا زمانیکه به انتهای فایل نرسیده ایم، بصورت خط به خط از فایل خوانده و هر خط را پس از خواندن در خروجی چاپ می نماییم.

خروجی برنامه فوق در زیر نمایش داده شده است.



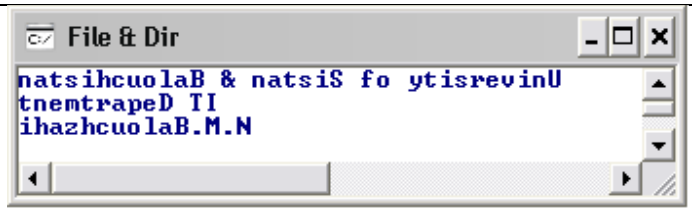
مثال۱۰) متدی بنویسید که با دریافت نام یک فایل، محتویات فایل را از انتها به ابتدا بصورت کارکتر به کارکتر چاپ نماید.

```
static void PrintReverse(string filename)
{
      Stack s = new Stack();
      StreamReader sr = new StreamReader(filename);
      while (!sr.EndOfStream)
            s.Push((char)sr.Read());
       sr.Close();
       while (s.Count > 0)
           Console.Write(s.Pop());
}
```

در این مثال برای معکوس کردن محتوای فایل از پشته استفاده شده است. به این صورت که محتوای فایل از ابتدا تا انتها خوانده شده و کلیه کارکترهای خوانده شده به پشته اضافه می شوند. سپس کل محتوای پشته بر روی صفحه مانیتور چاپ می شود.

نکته: با توجه به اینکه متد Read کد اسکی کارکتر خوانده شده را برمی گرداند، قبل از اضافه نمودن آن به پشته، به کمک type-casting آن را به معادل کارکتری تبدیل نموده و به پشته اضافه می نماییم.

نمونه خروجی مثال فوق در زیر نشان داده شده است:



روشهای تشخیص انتهای فایل در فایلهای متنی:

۱. استفاده از خاصیت `EndOfStream` به همان صورت که در مثالهای فوق از آن استفاده شد.
۲. استفاده از متد `ReadLine` با توجه به اینکه این متد با رسیدن به انتهای فایل مقدار `null` را برمی گرداند.
۳. استفاده از متد `Read` با توجه به اینکه این متد با رسیدن به انتهای فایل مقدار ۱- را برمی گرداند.
۴. استفاده از متد `Peek` که صرفاً به همین منظور درنظر گرفته شده است. که با رسیدن به انتهای فایل مقدار ۱- را برمی گرداند.

به عنوان مثال می توان در مثال قبلی به جای شرط پایان از دستور `sr.Peek()!=-1` استفاده نمود.

تمرین۳:برنامه ای بنویسید که با دریافت یک فایل هر کلمهٔ فایل را با دفعات تکرارش در خروجی چاپ نماید.
تمرین۴: برنامه ای بنویسید که با دریافت یک فایل تعداد کلمات یک حرفی، دوحرفی و ... را در خروجی چاپ نماید.
تمرین۵: برنامه ای بنویسید که با دریافت دو فایل از سطر فرمان میزان تشابه و تفاوت دو فایل را به کاربر گزارش دهد، به اینصورت که تعداد کلمات مشترک، تعداد کلمات مورد اختلاف دو فایل را گزارش دهد.
تمرین ۶: برنامه ای بنویسید که با دریافت یک فایل از ورودی با این فرض که هر سطر فایل حاوی نام خانوادگی و نام یک نفر می باشد، اطلاعات درون فایل را بر اساس نام خانوادگی و نام مرتب نماید.

فایلهای دودویی

نوع دیگر فایلها در `C#` که جهت ذخیره اطلاعات نوع دار از آنها استفاده می شود، عبارتند از فایلهای باینری که در این فایلها اطلاعات به همان شیوه حافظهٔ اصلی در آنها ذخیره می شوند.

نحوه دسترسی به اطلاعات دودویی درون فایل و یا نوشتن اطلاعات در آن با توجه به اینکه نوع اطلاعات مورد استفاده چه باشد، متفاوت است. به عنوان مثال ممکن است نیاز داشته باشیم تا داده هایی با نوع ساده مانند `int`، `char`، `double` و غیره را در فایلها ذخیره نماییم و یا اینکه بخواهیم داده هایی با انواع ترکیبی و پیچیده مانند کلاس و یا ساختار را ذخیره نماییم که در این دو حالت روش کار متفاوت خواهد بود.

ابتدا دو کلاس ساده جهت ذخیره و بازیابی انواع پایه ارائه می شود، سپس به ارائه روشی کلی جهت ذخیره سازی انواع ساده و پیچیده می پردازیم.

برای انواع پایه دو کلاس `BinaryReader` و `BinaryWriter` وجود دارند که به ترتیب از اولی جهت خواندن اطلاعات و از دومی جهت نوشتن در فایلهای دودویی می توان استفاده نمود.

برای خواندن از فایل و یا نوشتن در فایل باید ابتدا فایل را باز نمود و برای هر کدام از حالات خواندن و یا نوشتن نحوهٔ باز کردن فایل متفاوت خواهد بود. برای این منظور از کلاس `FileStream` استفاده می شود که چگونگی عملکرد آن همراه با جزئیات در زیر آمده است.

فصل دهم: فایلها

برای باز کردن فایل یک متغیر از نوع کلاس FileStream ایجاد نموده و در سازندهٔ آن مشخصات فایلی که باید باز شود و نحوهٔ باز شدن فایل را تعیین می نماییم.

```
FileStream fs = new FileStream ( نام و آدرس فایل , مد باز شدن , نحوهٔ دسترسی );
```

قبل از ارائه مثال از چگونگی کارکرد کلاس فوق، به تشریح پارامترهای مد باز شدن و نحوهٔ دسترسی می پردازیم.

مدهای باز کردن فایل(FileMode):

مقدار پارامتر	شرح
Append	اگر فایل وجود داشته باشد، اطلاعات به انتهای آن افزوده می شود، در غیر اینصورت فایل جدیدی ایجاد می گردد و اطلاعات در آن ذخیره می شود.
Create	اگر فایل وجود داشته باشد، حذف شده و از اول ایجاد می گردد.
CreateNew	اگر فایل وجود داشته باشد، تولید خطا می نماید در غیر اینصورت فایل را ایجاد می نماید.
Open	اگر فایل وجود نداشته باشد، تولید خطا می نماید در غیر اینصورت فایل را باز می کند.
OpenOrCreate	اگر فایل وجود داشته باشد آنرا باز می نماید، در غیر اینصورت آنرا ایجاد می نماید.
Truncate	اگر فایل موجود باشد، اطلاعات قبلی آن از بین می رود و اگر وجود نداشته باشد، تولید خطا می نماید.

نحوهٔ دسترسی به فایل(FileAccess):

مقدار پارامتر	شرح
Read	در این حالت فقط می توان از فایل اطلاعات را خواند.
Write	در این حالت فقط می توان اطلاعات را در فایل نوشت.
ReadWrite	در این حالت هم می توان از فایل خواند و هم می توان در فایل نوشت.

مثال۱۱) برنامه ای بنویسید که یک فایل باینری با نام tst.dat در دایرکتوری C:\nik ایجاد نموده و کلیه اعداد فرد دو رقمی را در آن قرار دهد.

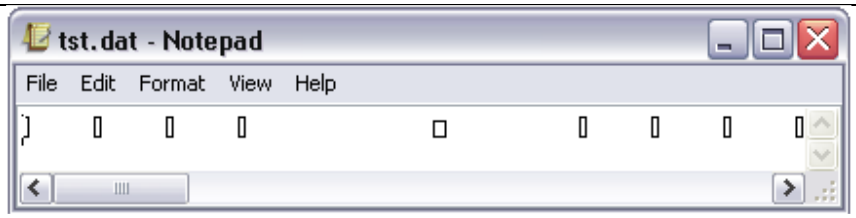
```
FileStream fs = new FileStream(@"c:\nik\tst.dat", FileMode.OpenOrCreate,
FileAccess.Write);
BinaryWriter bw = new BinaryWriter(fs);

for (int i = 1; i <= 99; i += 2)
        bw.Write(i);
bw.Close();
fs.Close();
```

در اینحالت ابتدا چک می شود که اگر فایل وجود دارد آنرا باز نماید، در غیر اینصورت فایل را ایجاد می کند. در هر صورت متغیر fs در واقع مشخصات فایل مورد نظر را دارا می باشد.

پس از آن متغیر bw ایجاد می شود که در واقع این متغیر نیز به کمک fs به فایل باز شده در مرحلهٔ قبل دسترسی دارد.

پس از آن به کمک شیء bw و متد Write اطلاعات موردنظر را در فایل باز شده می نویسیم و در انتها فایل بسته می شود. اطلاعات نوشته شده در فایل به فرمت باینری بوده و قابل فهم نیستند در زیر نمونه اطلاعات فوق که در فایل tst.dat نوشته شده اند، نشان داده شده است.

فصل دهم: فایلها



جهت خواندن اطلاعات از فایل مورد نظر باید از کلاس BinaryReader استفاده نمود که چگونگی استفاده از آن در زیر نشان داده شده است.

مثال۱۲) برنامه ای بنویسید که مجموع اعداد فرد درون فایل فوق را محاسبه نموده و چاپ نماید.

```
FileStream fs = new FileStream(@"c:\nik\tst.dat", FileMode.Open, FileAccess.Read);
BinaryReader br = new BinaryReader(fs);
int sum=0;
while (br.PeekChar() != -1)
      sum += br.ReadInt32();
br.Close();
fs.Close();
Console.WriteLine("Sum = "+ sum);
```

در این حالت فایل را فقط برای خواندن از آن باز کرده ایم، سپس به کمک متد ReadInt32 هر بار یک عدد صحیح از فایل موردنظر خوانده شده است و تا زمانیکه به انتهای فایل نرسیده ایم، اعداد خوانده شده با همدیگر جمع شده اند و در انتها مجموع اعداد در خروجی چاپ شده است.

متد PeekChar فقط کارکتری را که اشاره گر فایل به آن اشاره می کند، چک می نماید که آیا با کارکتر انتهایی فایل یکسان است یا نه؟ که اگر یکسان باشد مقدار منفی یک بر می گرداند. به همین دلیل از این متد جهت بررسی انتهای فایل بهره جسته ایم.

همچنین کلاس BinaryReader دارای متدهای متفاوتی جهت خواندن انواع مختلف داده ها می باشد. به عنوان نمونه متد ReadInt32 جهت خواندن اعداد int و از متد ReadSingle جهت خواندن اعداد float و از هر کدام از بقیه متدها جهت خواندن یک نوع داده خاص استفاده می شود.

مثال ۱۳) برنامه ای بنویسید که ۱۰۰ عدد اعشاری بین ۱۰و۲۰ تولید نموده و در فایلی ذخیره نماید. سپس تعداد اعداد بزرگتر از میانگین را چاپ نماید.

برای این منظور ابتدا ۱۰۰ عدد تصادفی در بازهٔ ۱۰ تا ۲۰ تولید نموده و در فایلی ذخیره می نماییم. بنابراین این اعداد همیشه در فایل وجود دارند و هر زمانی می توان آنها را از فایل بازیابی کرد.

که یک بار کلیه اعداد بازیابی شده اند تا مجموع و سپس میانگین آنها محاسبه گردد، سپس یکبار دیگر فایل باز شده است تا تعداد عناصری که از میانگین بیشتر می باشند شمارش شده و در خروجی چاپ شود.

کل کامل برنامه در زیر آورده شده است.

```
using System;
using System.IO;
class Program
  {
     static void Main(string[] args)
      {
        Random random=new Random();
        FileStream fs;

        fs= new FileStream(@"c:\nik\tst.dat", FileMode.Create, FileAccess.Write);
```

```
  BinaryWriter bw = new BinaryWriter(fs);
  for (int i = 1; i <= 100; i++)
    {
       double rnd = 10+10*random.NextDouble();
        bw.Write(rnd);
    }
  bw.Close();
  fs = new FileStream(@"c:\nik\tst.dat", FileMode.Open, FileAccess.Read);
  BinaryReader br = new BinaryReader(fs);
  double sum = 0;
  int count = 0;
  while (br.PeekChar() != -1)
       {
         sum += br.ReadDouble();
         count++;
        }
  br.Close();
  fs.Close();
  double avg = sum / count;
  fs = new FileStream(@"c:\nik\tst.dat", FileMode.Open, FileAccess.Read);
  br = new BinaryReader(fs);
  count = 0;
  while (br.PeekChar() != -1)
    {
      if( br.ReadDouble()>avg)
           count++;
    }
  br.Close();
  fs.Close();
  Console.WriteLine("Count = "+ count);
  Console.ReadLine();
  }
}
```

در کلیهٔ حالات فوق نحوهٔ دسترسی به اطلاعات درون فایل ترتیبی است، یعنی ابتدا رکورد اول فایل سپس رکورد دوم و به همین ترتیب باقیماندهٔ رکوردها تا انتهای فایل پیمایش می شوند.

به این روش دسترسی به اطلاعات درون فایل **سازمان ترتیبی** گویند.

روش دیگری نیز جهت دسترسی به اطلاعات درون فایل وجود دارد که در آن در هر لحظه می توان هر رکورد را در هر جایی از فایل نوشت و از هر جایی از فایل نیز می توان یک رکورد را بازیابی نمود به این روش دسترسی به اطلاعات درون فایل **سازمان تصادفی** گویند.

جهت ایجاد سازمان تصادفی در فایلهای نوع دار که دارای نوع ساده می باشند، از دو کلاس BinaryReader و BinaryWriter استفاده می گردد. در ادامه به ذکر مثالهایی از چگونگی خواندن و نوشتن تصادفی در فایلهای باینری نوعدار به انواع پایه پرداخته می شود.

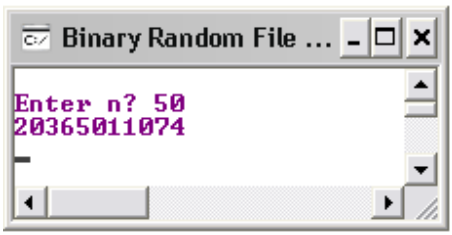
مثال۱۴) برنامه ای بنویسید که ۱۰۰ عدد اول سری فیبوناچی را در یک فایل باینری قرار دهد، سپس با دریافت یک n از کاربر عدد nام را از فایل بازیابی نموده و در خروجی چاپ نماید.

برای انجام این کار ابتدا ۱۰۰ عدد آغازین سری فیبوناچی را تولید نموده و در فایلی قرار می دهیم، سپس با دریافت n از کاربر اشاره گر فایل را با استفاده از خاصیت Position به رکورد nام منتقل می نماییم، سپس رکورد موردنظر را بازیابی نموده و مقدار آنرا چاپ می کنیم.

فصل دهم: فایلها

با توجه به اینکه اندازهٔ اعداد فیبوناچی دارای رشد سریعی می باشد، نوع اعداد مورد استفاده را ulong گرفته ایم . در زیر برنامهٔ لازم برای انجام این عمل و خروجی آن نمایش داده شده است.

```
using System;
using System.IO;
namespace BinaryRandom
{
    class Program
    {
        static void Main(string[] args)
        {
            Start("Binary Random File Access!");
            FileStream fs;
            fs = new FileStream("c:\\2.bin", FileMode.Create, FileAccess.Write);
            BinaryWriter bw = new BinaryWriter(fs);

            ulong f, f1=1, f2=1;
            bw.Write(f1);
            bw.Write(f2);
            for (int i = 2; i < 100; i++)
            {
                f = f1 + f2;
                f1 = f2;
                f2 = f;
                fs.Position = i * sizeof(ulong);
                bw.Write(f);

            }
            bw.Close();
            fs.Close();

            fs = new FileStream("c:\\2.bin", FileMode.Open, FileAccess.Read);
            BinaryReader br = new BinaryReader(fs);
            Console.Write("\nEnter n? ");
            int n = int.Parse(Console.ReadLine());
            fs.Position =n*sizeof(ulong);
            Console.WriteLine(br.ReadUInt64());

            Console.ReadLine();
        }

        static void Start(string title)
        {
            Console.BackgroundColor = ConsoleColor.White;
            Console.ForegroundColor = ConsoleColor.DarkMagenta;
            Console.Title = title;
            Console.Clear();
        }
    }
}
```



چنانچه بخواهیم که باقیماندهٔ اعداد فیبوناچی به عنوان مثال ۱۰۰ عدد بعدی را به انتهای فایل موجود اضافه نماییم، ابتدا اشاره گر فایل را به انتهای فایل منتقل نموده سپس اعداد جدید را در آنجا می نویسیم:

```
FileStream fs;
fs = new FileStream("c:\\2.bin", FileMode.Create, FileAccess.Write);
BinaryWriter bw = new BinaryWriter(fs);

bw.Seek (fs.Length , SeekOrigin.Begin) ;
fs.Positon=fs.Length;
```

دستورات اضافه نمودن اطلاعات

```
bw.Close();
fs.Close();
```

از هر کدام از دستورات فوق می توان جهت انتقال اشاره گر فایل به انتهای فایل استفاده نمود. بعد از آن اگر با متد Write اطلاعات را به فایل اضافه نماییم، این اطلاعات به انتهای فایل اضافه خواهد شد.

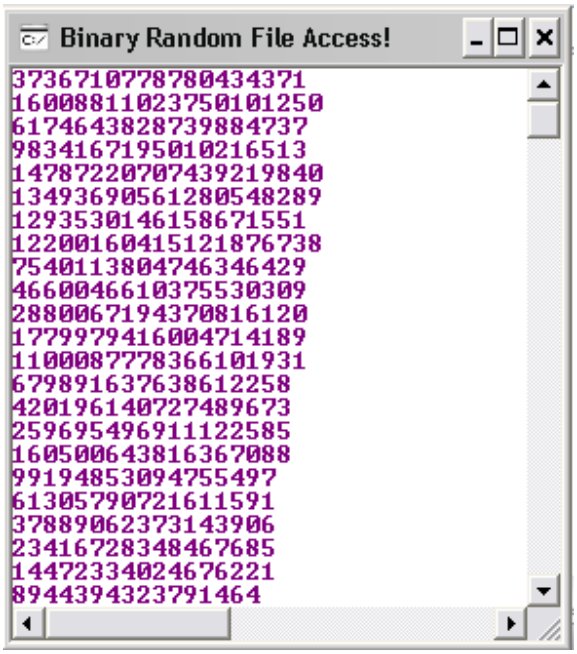
مثال۱۵) اطلاعات درون فایل فوق را از انتها به ابتدا چاپ نمایید.

در این حالت ابتدا اشاره گر را به انتهای فایل منتقل نموده، عدد آخر را می خوانیم سپس هر بار اشاره گر را به عدد قبلی منتقل می نماییم تا ابتدای فایل برسیم.

مجموعه دستورات لازم جهت انجام عمل فوق و خروجی آنها در زیر نشان داده شده است.

```
fs = new FileStream("c:\\2.bin", FileMode.Open, FileAccess.Read);
BinaryReader br = new BinaryReader(fs);
long l = fs.Length - sizeof(ulong);
while (l >= 0)
  {
    fs.Position = l;
    Console.WriteLine(br.ReadUInt64());
    l -= sizeof(ulong);
  }
```

خروجی دستورات فوق در زیر نشان داده شده است.



فصل دهم: فایلها

خواندن و نوشتن ساختارها و اشیاء از/به فایلها

همانند فایلهای باینری با نوع ساده در اینجا نیز دو سازمان متفاوت جهت دسترسی به اطلاعات درون فایل داریم که عبارتند از سازمانهای **ترتیبی** و **تصادفی**.

در هر یک از این دو سازمان مکانیزمهای متفاوتی جهت دسترسی به اطلاعات درون فایل وجود دارد که ابتدا به سازمان ترتیبی و سپس به سازمان تصادفی می پردازیم:

در سازمان ترتیبی جهت خواندن و نوشتن رکوردها از/به فایل از عملیات Serialization و DeSerialization استفاده می شود. که در حالت اول اطلاعات مورد نظر (رکورد و یا شیء) به صورت مجموعه ای از بایتها در می آید و آن مجموعه در فایل نوشته می شود. اما در حالت دوم جریانی از بایتها از فایل خوانده شده و به رکورد و یا شیء مورد نظر تبدیل می شود.

جهت انجام دو عمل فوق معمولا از یکی از دو کلاس زیر استفاده می باشد:

- ✓ BinaryFormatter
- ✓ Soap

حالت اول در فایلهای موجود بر روی کامپیوتر محلی به کار می رود و حالت دوم بیشتر تحت وب کاربرد دارد.

در این بخش فقط به کلاس BinaryFormatter پرداخته می شود.

این کلاس دارای دو متد به نامهای Serialize و DeSerialize می باشد که جهت انجام اعمال فوق بکار می روند.

مثال۱۶) در برنامهٔ زیر چگونگی استفاده از متدها و کلاسهای مورد نیاز جهت انجام این کار نشان داده است:

```
using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
namespace BinaryFileSerialize
{
    class Program
    {
        static void Main()
        {
            FileStream fs ;
            fs= new FileStream("c:\\new2.dat", FileMode.OpenOrCreate,
                     FileAccess.Write);
            BinaryFormatter bf = new BinaryFormatter();
            fs.Seek(0, SeekOrigin.End);
                                انتقال اشاره گر به انتهای فایل
            for (int i = 1; i < 5; i++)
            {
                Complex c = new Complex(2.3*i, 3.2*i);
                bf.Serialize(fs, c);
                                اضافه نمودن اطلاعات به انتهای فایل

            }
            fs.Close();
            fs = new FileStream("c:\\new2.dat", FileMode.Open, FileAccess.Read);
            bf = new BinaryFormatter();
            try
            {
                int k = 0; ;

                while (fs.Position < fs.Length) // تست رسیدن به انتهای فایل
```

```
                {
                    Complex c1 = (Complex)bf.Deserialize(fs);
                       خواندن یک رکورد از فایل و تبدیل آن به نوع Complex
                    c1.Print();
                }

            }
            catch (SerializationException e)
            {
                Console.Write(e.Message);
                Console.ReadLine();
            }

            fs.Close();
            Console.ReadLine();
        }
    }
    [Serializable]
```

این عبارت اضافه می شود تا رکورد ما قابل Serialization و DeSerialization باشد.در غیر اینصورت نمی توانیم عملیات فوق را بر روی رکورد انجام دهیم.

```
    struct Complex
    {
        double real, img;
        public Complex(double real, double img)
        {
            this.real = real;
            this.img = img;
        }
        public void Print()
        {
            Console.Write("\n {0}+{1}i", real, img);
        }
        public double Real
        {
            set { real=value;}
            get { return real; }
        }
        public double Img
        {
            set
            {
                img = value;
            }
            get { return img; }
        }
    }
}
```

روش دیگر دستیابی به اطلاعات درون فایل روش دستیابی تصادفی می باشد که در این روش در هر لحظه اشاره گر فایل را به یک مکان در فایل منتقل نموده سپس اطلاعات را از / به آن مکان خوانده و یا می نویسیم.

برای انجام این کار از همان کلاسهای BinaryReader و BinaryWriter استفاده می شود. اما در این حالت با توجه به اینکه اندازهٔ رکوردها ثابت نیست ابتدا باید اندازهٔ رکورد را ثابت نمود تا اینکه همهٔ رکوردها دارای اندازهٔ ثابتی باشند، سپس رکوردها را در فایل نوشته و یا از فایل می خوانیم.

فصل دهم: فایلها

جهت ثابت نمودن اندازهٔ رکوردها چون اندازهٔ انواع پایه ثابت می باشد، فقط کافی است که اندازهٔ انواعی را که میزان حافظهٔ مصرفی آنها وابسته به مقدار آنها می باشد، ثابت نماییم. در بین انواع فقط کافی است که نوع string را ثابت نماییم که این کار براحتی امکان پذیر می باشد بدینگونه که یک اندازهٔ ماکزیمم برای آن فرض نموده و چنانچه اطلاعات ورودی از آن حجم کمتر باشد، با اضافه نمودن فاصله به انتهای اطلاعات اندازهٔ آنها را ثابت می نماییم اما اگر حجم اطلاعات ورودی بیشتر باشد از بخش اضافی آنها صرفنظر می شود.

در زیر برنامهٔ کاملی از یک رکورد که حاوی اطلاعات دانشجویان می باشد نشان داده شده است که در آن ابتدا طول رکورد را ثابت نموده و سپس اطلاعات با طول ثابت را در فایل می نویسیم، سپس با توجه به اینکه طول رکورد ثابت است با انتقال اشاره گر فایل به یک مکان در فایل، رکورد موجود در آن مکان را بازیابی می نماییم.

در این مثال برای راحتی کار فرض شده است که شماره های افراد از یک شروع می شود به همین دلیل اطلاعات نفر اول در مکان صفر ، نفر دوم در مکان یک و به همین ترتیب اطلاعات در فایل اضافه می گردد.

مثال۱۷) برنامهٔ کامل چگونگی نوشتن رکوردها در فایل و نحوهٔ دستیابی به آنها همرا با مراحل مختلف اجرای برنامه

```
using System;
using System.IO;
namespace BinaryRandom
{
    class Program
    {
        const int RecordSize = 15*2+4+8;
        const int NumOfRecord = 200;
        static void Main(string[] args)
        {
            Start("Binary Random File Access!");
            FileStream fs;
            Console.Write("\n Do you want to create new file (y/n)?");
            char ch = (char)Console.ReadLine()[0];
            if (char.ToLower(ch) == 'y')
            {
                Console.Write("\n Please enter Maximum number of
                  records(<1000):?");
                int num = int.Parse(Console.ReadLine());
                num = num > 1000 ? 1000 : num;
                InitializeFile("C:\\1.bin", num);
            }
            Console.Write("\n Do you want to add recors?");
            ch =(char) Console.ReadLine()[0];
            if (char.ToLower(ch) == 'y')
            {
                Console.Write("\n Please enter number of records:?");
                int num = int.Parse(Console.ReadLine());
                fs = new FileStream("c:\\1.bin", FileMode.OpenOrCreate,
                        FileAccess.Write);
                BinaryWriter bw = new BinaryWriter(fs);
                for (int i = 0; i <num; i++)
                {
                    Student s = GetStudent();
                    fs.Position = (s.ID - 1) * RecordSize;
                    bw.Write(s.Name);
                    bw.Write(s.ID);
                    bw.Write(s.Avg);
                }
                bw.Close();
                fs.Close();
            }
```

```
            fs = new FileStream("c:\\1.bin", FileMode.Open, FileAccess.Read);
            BinaryReader br = new BinaryReader(fs);
            long ID;
            Console.Write("\nEnter student id for search:");
            ID = long.Parse(Console.ReadLine());
            fs.Position = (ID - 1) * RecordSize;
            string name = br.ReadString();
            ID = br.ReadInt64();
            float avg = br.ReadSingle();

            if (ID == 0)
                Console.WriteLine("not found");
            else
                Console.WriteLine("ID={0}\tName={1}\tAvg={2}", ID, name, avg);

            br.Close();
            fs.Close();
            Console.ReadLine();
        }

        static Student GetStudent()
        {
            Student s = new Student();
            Console.Write("\nEnter Name:");
            string name=Console.ReadLine();
            Console.Write("\nEnter ID:");
            long ID =long.Parse( Console.ReadLine());
            Console.Write("\nEnter avg:");
            float avg =float.Parse( Console.ReadLine());
            return new Student(name, ID, avg);

        }
        static void InitializeFile(string fileName,int num)
        {
            Student emptystudent=new Student(" ",0,0.0f);

            FileStream fs;
            fs = new FileStream(fileName, FileMode.Create, FileAccess.Write);
            BinaryWriter bw = new BinaryWriter(fs);

            for (int i = 0; i < num; i++)
            {
                fs.Position = i * RecordSize; // نیازی نمی باشد
                bw.Write(emptystudent.Name);
                bw.Write(emptystudent.ID);
                bw.Write(emptystudent.Avg);

            }
            bw.Close();
            fs.Close();

        }

        static void Start(string title)
        {
            Console.BackgroundColor = ConsoleColor.White;
            Console.ForegroundColor = ConsoleColor.DarkMagenta;
            Console.Title = title;
            Console.Clear();
        }
    }
```

```
    struct Student
    {
        const int MaxName = 15;
        string name;
        long sid;
        float avg;

        public Student(string n, long id, float avg)
        {
            sid = id;
            this.avg = avg;
            if (n.Length > MaxName)
                name = n.Substring(0, MaxName);
            else if (n.Length < MaxName)
                name = n + new string(' ', MaxName - n.Length);
            else
                name = n;
        }

        public string Name
        {
            get { return name; }
            set
            {
                if (value.Length > MaxName)
                    name = value.Substring(0, MaxName);
                else if (value.Length < MaxName)
                    name = value + new string(' ', MaxName - value.Length);
                else
                    name = value;
            }
        }
        public long ID
        {
            get { return sid; }
            set { sid = value; }
        }
        public float Avg
        {
            get { return avg; }
            set { avg = value; }
        }
    }
}
```

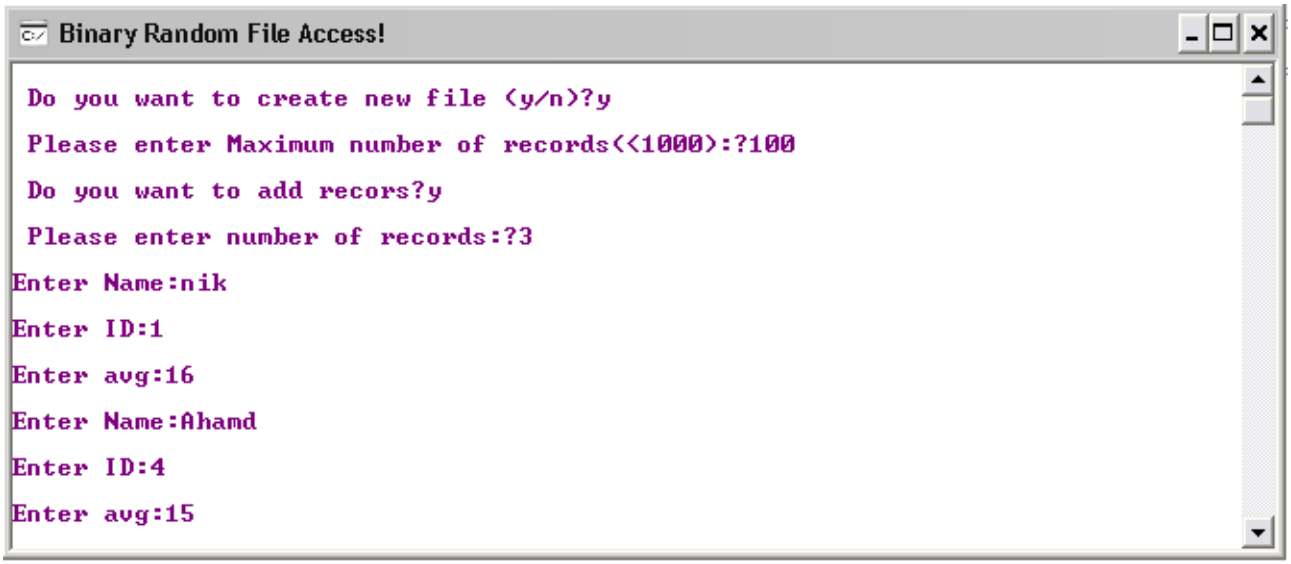
در برنامهٔ فوق چندین متد وجود دارد که هر کدام همراه با جزئیات آن در زیر شرح داده شده است:

ابتدا به ذکر جزئیات مربوط به ساختار Student می پردازیم، این رکورد از سه بخش name، sid و avg تشکیل شده است که عبارتند از نام، شماره و معدل دانشجو. دو فیلد دوم دارای اندازهٔ ثابتی می باشند، اما فیلد اول که حاوی نام دانشجو می باشد با توجه به نام وارد شده توسط کاربر می تواند حجم متغیری از حافظه را اشغال نماید.

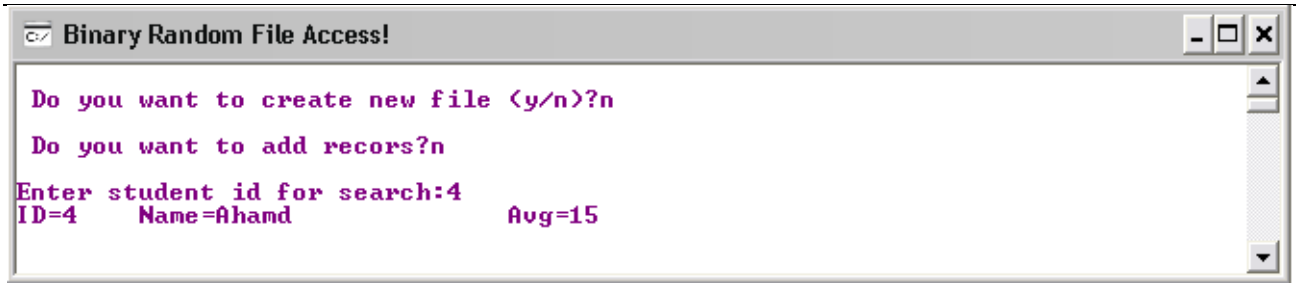
جهت ثابت نمودن اندازهٔ رکورد باید اندازهٔ این فیلد را ثابت نماییم برای این کار فرض می کنیم که هر نام حداکثر حاوی ۱۵ کارکتر باشد و در زمان مقداردهی این فیلد که یا در سازنده و یا در خاصیت انجام می پذیرد، چک می کنیم که اگر طول بیشتر از ۱۵ باشد از باقیماندهٔ اطلاعات صرفنظر می شود و اگر کمتر از آن باشد باقیماندهٔ رشته را با فاصله پر می نماییم.چگونگی انجام این کار در سازنده و خاصیت Name به وضوح مشاهده می گردد.

پس از ثابت نمودن اندازهٔ رکوردها باید آنها را در فایل بنویسیم. جهت نوشتن رکوردها در فایل باید ابتدا مکان رکورد را پیدا نموده و با انتقال اشاره گر به آن مکان رکورد را در آن مکان بنویسیم. مکان هر رکورد در فایل چنانچه فرض نماییم که رکورها از عددی مانند Base شروع می شوند عبارت است از sid-Base که چون فرض نموده ایم که مقدار Base برابر ۱ می باشد، پس مکان هر رکورد یکی کمتر از شمارهٔ آن می باشد.
برای جستجوی یک رکورد خاص پس از انتقال اشاره گر به مکان رکورد موردنظر رکورد را می خوانیم و اطلاعات آنرا چاپ می نماییم اما اگر اطلاعات چنین فردی هنوز وارد نشده باشد، چگونه می توان به آن پی برد؟
برای رفع این مشکل ابتدا کل فایل را با رکوردهای خالی پر نموده، سپس اگر هنگام خواندن یک رکورد فیلد شماره آن برابر مقدار پیش فرض (صفر) بود، در می یابیم که چنین رکوردی هنوز اطلاعاتش وارو نشده است. برای این منظور متدی بصورت `static void InitializeFile(string fileName,int num)` نوشته شده است که با دریافت نام فایل و تعداد رکوردها فایلی با نام داده شده ایجاد می نماید. سپس به تعداد داده شده رکورد خالی در فایل ایجاد شده می نویسد. مقادیر هر رکورد مقادیر پیش فرضی است که در متد مشخص شده است.

جهت نوشتن اطلاعات یک دانشجو در فایل ابتدا اشاره گر را به مکان مورد نظر منتقل نموده و اطلاعات دانشجو را بصورت فیلد به فیلد در فایل می نویسیم و برای خواندن نیز دقیقا به همین ترتیب عمل می شود که ابتدا شمارهٔ دانشجو از کاربر دریافت شده و با انتقال اشاره گر به مکان رکورد موردنظر اطلاعات را بصورت فیلد به فیلد از آن مکان می خوانیم.
نمونه هایی از اجرای برنامه در زیر نشان داده شده است که ابتدا با سوال از کاربر فایل جدیدی که حداکثر گنجایش آن از کاربر دریافت می گردد ایجاد می شود و پس از آن با وارد نمودن اطلاعات توسط کاربر اطلاعات در مکان صحیح خویش قرار می گیرند. همچنین کاربر می تواند با وارد نمودن یک شماره دانشجویی اطلاعات آنرا بازیابی نماید.

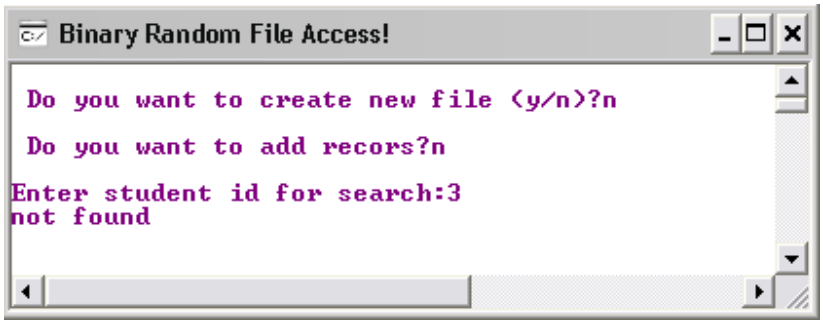


همانگونه که مشاهده می نمایید کاربر از برنامه می خواهد که فایل جدیدی که گنجایش حداکثر ۱۰۰ رکورد را داشته باشد، ایجاد نماید و آنرا با رکوردهای خالی مقداردهی نماید.



نمونهٔ دیگری از خروجی برنامه می باشد که در آن کاربر با وارد کردن شمارهٔ یک فرد اطلاعات آن فرد را مشاهده می نماید.

اگر شمارهٔ فردی وارد شود که اطلاعات وی در فایل اضافه نشده باشد، برنامه با بازیابی اطلاعا ت آن دانشجو چک می نماید، چنانچه شمارهٔ رکورد برابر صفر باشد، چنین فردی در فایل وجود ندارد. نمونهٔ اجرای برنامهٔ فوق در زیر نشان داده شده است:



در این حالت برنامه با نمایش پیغام `not found` به اطلاع کاربر می رساند که اطلاعات چنین فردی هنوز در فایل اضافه نشده است.

تمرینات بخش فایلها

۱. برنامه ای بنویسید که با دریافت نام فایل متنی و یک رشته بعنوان آرگومان سطر فرمان، شمارهٔ سطرهایی از فایل را چاپ نماید که این کلمه در آن خطوط وجود داشته باشد.

۲. برنامه ای بنویسید که با دریافت نام چندین فایل متنی از سطر فرمان تعداد خطوط هر فایل را بصورت جدا چاپ نماید.

۳. برنامه ای بنویسید که با دریافت نام سه فایل متنی از سطر فرمان، اطلاعات فایلهای اول و دوم را بصورت سطر به سطر خوانده و در فایل سوم اضافه نماید. بگونه ای که اطلاعات فایلهای اول و دوم بصورت یک سطر در میان در فایل دوم اضافه گردد. (سطرهای فرد حاوی اطلاعات فایل اول و سطرهای زوج حاوی اطلاعات فایل دوم باشد)

۴. برنامه ای بنویسید که کتابخانه شخصی شما را مکانیزه نماید، اطلاعاتی از کتب که باید در کتابخانه ذخیره شود عبارتند از :

عنوان کتاب، نام نویسنده، نام مترجم، تاریخ خرید، قیمت خرید، موضوع کتاب ، نام ناشر و شماره کتاب .

- ✓ جهت ایجاد یک کتاب از کلاس استفاده نمایید.
- ✓ سازمان فایل را تصادفی درنظر بگیرید.
- ✓ کتاب ممکن است مترجم نداشته باشد.

فصل دهم: فایلها

اعمالی که برنامه باید انجام دهد عبارتند از:

- ✓ چاپ مشخصات کلیه کتابها
- ✓ چاپ مشخصات کلیه کتابها به ترتیب تاریخ خرید
- ✓ چاپ مشخصات کلیه کتابها به ترتیب الفبایی عنوان
- ✓ چاپ مشخصات کلیه کتابها به ترتیب الفبایی موضوع
- ✓ جستجو بر اساس موضوع کتاب
- ✓ جستجو بر اساس نام نویسنده
- ✓ جستجو بر اساس نام مترجم
- ✓ جستجو بر اساس عنوان کتاب
- ✓ چاپ اطلاعات کتبی که در بازهٔ زمانی خاصی خریداری شده اند.
- ✓ محاسبه هزینهٔ انجام شده برای خرید کتاب در یک بازهٔ زمانی خاص
- ✓ بیشترین خرید متعلق به چه موضوعی بوده است؟
- ✓ آماری از لیست کتب موجود در کتابخانه بصورت موضوع کتاب و تعداد کتب موجود از آن موضوع ارائه نمایید

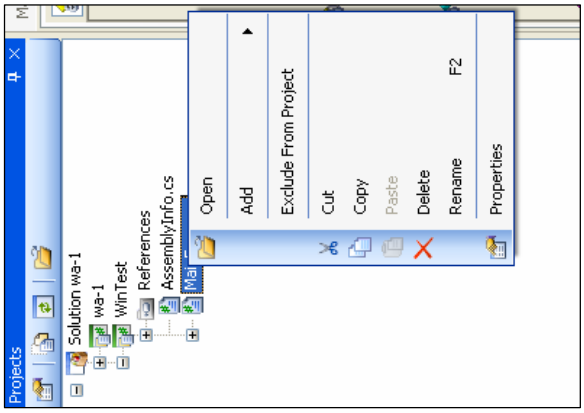
ومن الله التوفیق

بلوچ زهی

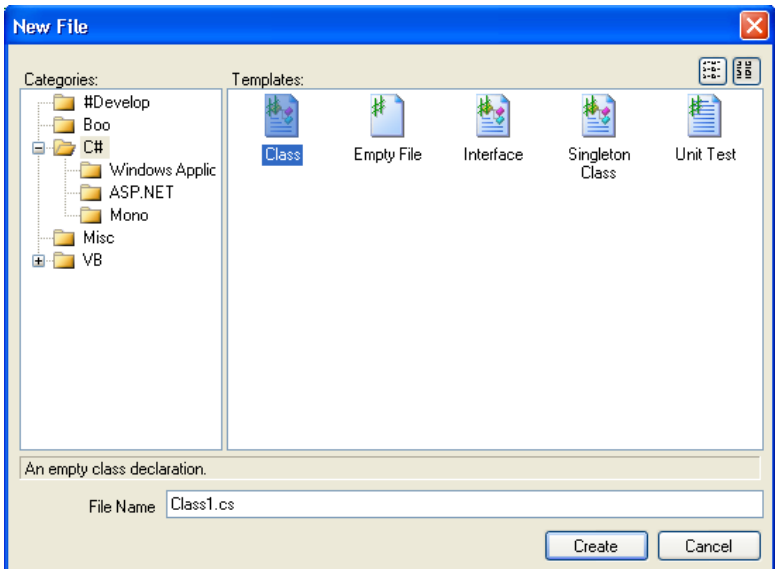
۲۶ / ۹ / ۸۵

در این فصل به چگونگی ایجاد یک فرم ویندوزی در SharpDevelop پرداخته می شود، سپس به تدریج به نحوهٔ اضافه نمودن بقیهٔ کامپونت ها به فرم و کار با Event های آنها پرداخته می شود.

برای این منظور ابتدا یک پروژه از نوع Windows Application ایجاد نمایید (البته می توانید دستورات را درNotepad تایپ نمایید)، سپس بر روی MainForm کلیک راست نموده و آنرا حذف نمایید.



سپس بر روی نام پروژه کلیک راست نموده و گزینهٔ Add/new Item را انتخاب نمایید، سپس از دیالوگ باز شده دو کلاس اضافه نمایید.



نام کلاسها را به ترتیب MyForm.cs و MainForm.cs انتخاب نمایید. کلاس MainForm فقط حاوی متد Main می باشد که از آن به عنوان EntryPoint استفاده می شود. همچنین از کلاس MyForm جهت ایجاد یک فرم و اضافه نمودن کامپونت ها به فرم استفاده می نماییم.

کلاس MyForm را بصورت زیر ایجاد نمایید:

```
using System;
using System.Windows.Forms;
using System.Drawing;
```

```
namespace WinTest
{
	public class MyForm:Form
	{
		public MyForm()
		{
			this.Text="Nik Form";
			this.BackColor=Color.Azure;
		}}}
```

کلاس فوق از کلاس Form ارث می برد. پس به کمک دستور Text می توانیم عنوان فرم را مشخص نمایید.

حال در کلاس MainForm مجموعه دستورات زیر را اضافه نمایید:

```
using System;
using System.Windows.Forms;
namespace WinTest
{
	public class MainForm
	{ static void Main()
		{
			Application.Run(new MyForm());
		}}}
```

حال پروژه را اجرا نمایید، خروجی بصورت زیر خواهد بود:

می توانید دستور Application.Run را با دستورات زیر جایگزین نمایید:

```
	MyForm f=new MyForm();
	f.ShowDialog();
```

چگونه می توانیم یک کامپوننت مثلا یک دکمه و یا یک TextBox را به فرم اضافه نماییم:

برای این منظور یک دکمه را در کلاس MyForm اضافه می نماییم، سپس دکمه را به فرم اضافه می کنیم، مجموعه دستورات زیر چگونگی انجام این کار را نشان می دهند:

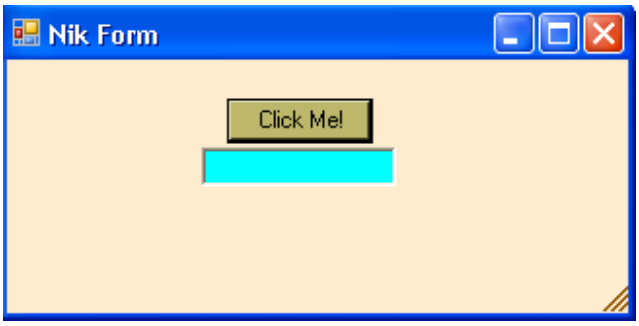
```
using System;
using System.Windows.Forms;
using System.Drawing;
namespace WinTest
{
	public class MyForm:Form
	{
```

```
            Button b; //تعریف دکمه
            public MyForm()
            {
                  this.Text="Nik Form"; //تعیین عنوان فرم
                  this.BackColor=Color.BlanchedAlmond; //تعیین رنگ فرم
                  b=new Button();                  //ایجاد دکمه
                  b.Text="Click Me!"; //تعیین عبارت روی دکمه
                  b.BackColor=Color.DarkKhaki;                  //تعیین رنگ دکمه
                  b.Left=this.Width/2 - b.Width/2; //تعیین مختصات افقی دکمه
                  b.Top=20;                  //تعیین مختصات عمودی دکمه
                  this.Controls.Add(b);    //اضافه نمودن دکمه به فرم
            }
      }
}
```

مشابه حالت فوق می توانید یک TextBox به فرم اضافه نمایید

```
t=new TextBox();
t.Left=this.Width/2 - t.Width/2;
t.Top=b.Top+t.Height+5;
t.BackColor=Color.BlueViolet;
this.Controls.Add(t);
```

نمونه خروجی برنامه در شکل زیر نمایش داده شده است:



حال اگر بر روی دکمه موجود کلیک نماییم، هیچ عملی صورت نمی گیرد. اگر بخواهیم با کلیک بر روی دکمه عملی صورت پذیرد، باید رخداد کلیک دکمه را تعریف نماییم. برای این منظور بصورت زیر عمل می نماییم:

```
b.Click+= delegate { MessageBox.Show("Welcome to Event Handelling!"); };
```

حال اگر بر روی دکمهٔ موردنظر کلیک نماییم مجموعه دستوراتی که در delegate مشخص شده اند، اجرا می شوند. نمونه اجرای برنامه در زیر نشان داده شده است:

علاوه بر روش فوق به کمک EventHandler نیز می توانیم رخداد کلیک موس را کنترل نماییم:

ابتدا درون کلاس متدی بصورت زیر اضافه می نماییم:

```
public void Button_Click(object sender,EventArgs e)
  {
      MessageBox.Show("Welcome to Event Handelling!");
  }
```

سپس در سازنده دستور زیر را اضافه می نماییم:

```
b.Click+= new EventHandler(Button_Click);
```

برنامهٔ کامل بصورت زیرمی باشد:

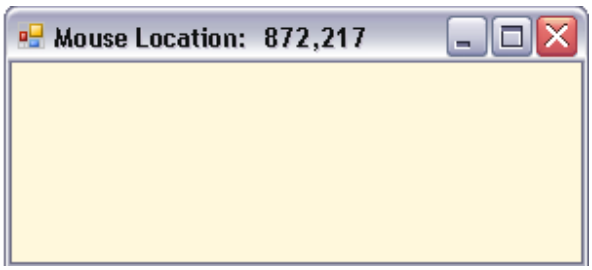
```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace Frm1
{
      public  class MainForm
      {
            public static void Main()
            {
                  Application.Run(new MyForm());
            }
      }
      public class MyForm:Form
      {
            Button b;
            public MyForm()
            {
                  this.Text="Nik Form";
                  this.BackColor=Color.Cornsilk;
                  b=new Button();
                  b.Left=this.Width/2-b.Width/2;
                  b.Top=this.Height/2-b.Height;
                  b.BackColor=Color.DeepSkyBlue;
```

```
                b.Text="Click Me!";
                this.Controls.Add(b);
                b.Click+= new EventHandler(Button_Click);
        }
        public void Button_Click(object sender,EventArgs e)
        {
                MessageBox.Show("Welcome to Event Handelling!");
        }
    }
}
```

در مثال زیر چگونگی handle نمودن رخداد MouseOver فرم نشان داده شده است. در این مثال هر وقت که موس بر روی فرم حرکت نماید، رخداد MouseOver رخ می دهد که در اینصورت تابع Mouse_Handle فراخوانی می گردد و مختصات فعلی موس را در عنوان فرم نمایش می دهد.

```
Using System;
using System.Drawing;
using System.Windows.Forms;
namespace Frm1
{
    public  class MainForm
    {
        public static void Main()
        {
            Application.Run(new MyForm());
        }
    }
    public class MyForm:Form
    {
        public MyForm()
        {
            this.Text="Nik Form";
            this.BackColor=Color.Cornsilk;
            this.MouseMove+= new MouseEventHandler(Mouse_Handle);
        }
        public void Mouse_Handle(object sender,EventArgs e)
        {
            this.Text="Mouse Location:  "+MousePosition.X+","+MousePosition.Y;
        }
    }
}
```

نمونه ای از خروجی برنامهٔ فوق در زیر نشان داده شده است.



کار با کامپوننتهای ویندوز

در این بخش با کامپوننت های موجود در .Net که برای ایجاد برنامه ها از آنها سود می بریم، شرح داده خواهند شد. به ازای هر کامپوننت ،ابتدا خود کامپوننت و کاربرد آن بیان خواهد شد، سپس با ذکر مثال چگونگی استفاده از آن بیان می گردد.

مثال۱) ابتدا به ذکر کامپوننت های پایه می پردازیم:

۱. Label
۲. TextBox
۳. ComboBox
۴. ListBox

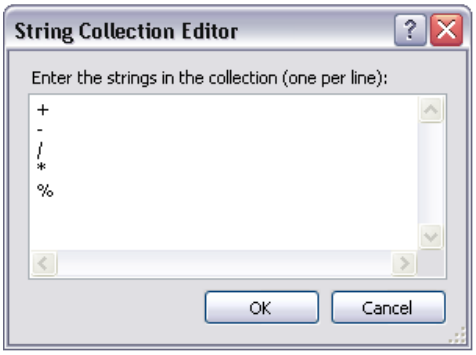
در این بخش جهت تشریح کامپوننت های فوق، به ارائه مثال ماشین حساب ساده ای می پردازیم که حاوی سه عدد TextBox جهت ورود پارامترها و نمایش خروجی و یک ComboBox و یا ListBox جهت نگهداری عملگرها می باشد.

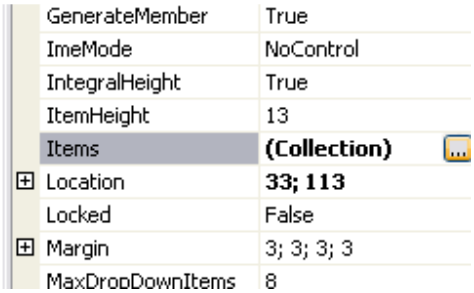
برای ایجاد چنین برنامه ای ابتدا یک برنامهٔ جدید ایجاد نموده و کامپوننت های موردنیاز را بر روی آن می کشیم که پس از جای دادن کامپوننت ها شکل فرم بصورت زیر خواهد بود:

همچنین یک Label برای بخش نتیجه درنظر گرفته شده است. جهت اضافه نمودن عملگرها به ComboBox دو راه وجود دارد که عبارتند از:

الف- اضافه نمودن در زمان طراحی

برای این منظور بر روی Items که جزو خواص ComboBox می باشد، کلیک نموده و در لیست موجود عملگرها را اضافه می نماییم.





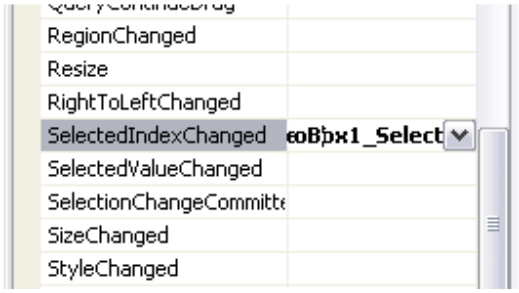
با این عمل عملگرهای فوق به بخش Items که یک Collection می باشد، افزوده می شوند.

ب-اضافه نمودن در زمان اجرا

با توجه به اینکه Items یک Collection می باشد می توانیم با متد Add در زمان اجرا به آن عنصر اضافه نماییم. به عنوان مثال در رخداد Form_Load می توان دستورات لازم جهت اضافه نمودن مقادیر را نوشت.

```
private void Form1_Load(object sender, EventArgs e)
        {
            comboBox1.Items.Add(مقدار عنصر);
        }
```

جهت دسترسی به هر کدام از این عناصر می توانیم از اندیس استفاده نماییم. به عنوان مثال ComboBox1.Item$ inde‡ مقدار عنصر ام بخش Items را به ما می دهد.خاصیت Text ، متنی را که در کامبو باید نشان داده شود، نمایش می دهد. همچنین کامبو دارای Eventی می باشد که هرگاه اندیس مقدار انتخاب شده در کامبو تغییر نماید، آن Event رخ می دهد (SelectedIndexChanged).



با اضافه نمودن Event فوق می توانیم چک نماییم که کدامیک از عملگرها انتخاب شده است، سپس با نوشتن دستورات لازم برای آن عملگر ، خروجی مناسب را در TextBox خروجی نمایش دهیم.

کامپوننت فوق دارای خاصیت SelectedIndex می باشد که اندیس عنصر انتخاب شده را به ما می دهد. می توان مجموعه دستورات زیر را به رخداد فوق اضافه نمود تا درصورت تغییر اندیس عملیات نوشته شده به درستی انجام پذیرد.

```
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    double op1, op2, result;
    op1 =double.Parse( textBox1.Text);// عدد اول
    op2 = double.Parse(textBox2.Text);// عدد دوم
    string op=comboBox1.Items[comboBox1.SelectedIndex ].ToString();// عملگر
    switch (op)
    {
        case "+":
            result=op1+op2;
            break;
        case "-":
            result = op1 - op2;
             break;
    }
    textBox3.Text = result.ToString(); //  نمایش نتیجه در خروجی
}
```

به همین ترتیب می توان کدهای لازم برای بقیهٔ عملگرها را نیز به برنامهٔ فوق اضافه نمود. با اضافه نمودن بقیه عملگرها مشکلی که وجود دارد این است که اگر اعداد درون TextBoxهای ورودی را تغییر دهیم، تا زمانیکه عملگر جدیدی انتخاب نشده باشد، خروجی تغییری نخواهد کرد. اگر بخواهیم با تغییر مقدار ورودی ها مقدار خروجی نیز تغییر نماید، باید در رخداد TextChanged هر کدام از TextBoxهای ورودی کدهای مرحلهٔ قبل را بازنویسی نماییم.

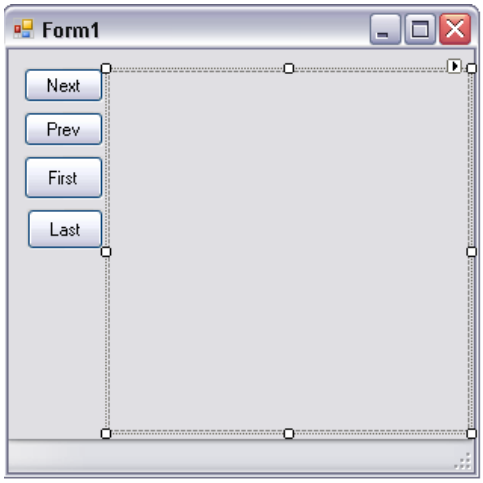
تمرین۱: دستورات لازم برای کلیهٔ رخدادهای موردنیاز را بنویسید.

تمرین۲: به جای ComboBox از ListBox استفاده نموده و برنامهٔ فوق را بازنویسی نمایید.

همچنین می توان از ToolTrip استفاده نمود تا در هنگام فوکس گرفتن یک کامپوننت ، عبارت مناسبی برای کاربر نمایش داده شود. با اضافه کردن کامپونت ToolTip خاصیت ToolTip on ToolTip1 به هر کدام از کامپونت های روی فرم افزوده می شود. عبارتی که در این خاصیت نوشته می شود، با فوکس گرفتن کامپونت برای کاربر نمایش داده می شود. نمونه ای از چگونگی عملکرد ToolTip در زیر نشان داده شده است.

مثال۲) در این مثال از کامپونت های PictureBox، Button و StatusBar جهت ایجاد یک آلبوم استفاده می نماییم.

نمایی از شکل اولیهٔ آلبوم در زیر نشان داده شده است:



از چهار دکمه جهت گردش در آلبوم و از PictureBox جهت نمایش عکس جاری و از StatusBar جهت نمایش ابعاد و نام عکس بهره می بریم.

برای این منظور در ابتدای برنامه لیست کامل عکسها را در آرایه ای از رشته ها قرار می دهیم و به کمک دکمه ها عکسها را پیمایش می نماییم.

ابتدا آرایهٔ موردنیاز و اندیس عکس جاری را در کلاس تعریف می کنیم، سپس در رخداد Form_Load دستورات لازم جهت نمایش عکس اول را در PictureBox می نویسیم. مجموعه دستورات زیر برای این منظور نوشته شده اند:

```
public partial class Form1 : Form
 {
    string[] album; // آرایه
    int index = 0;// اندیس

    private void Form1_Load(object sender, EventArgs e)
        {
            album = Directory.GetFiles("mypics");  // مقداردهی آرایه با دادن مسیر عکسها
            pictureBox1.Image = new Bitmap(album[0]);  // نمایش عکس اول در آلبوم
            pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage; // تنظیم نحوهٔ نمایش عکس
        }
}
```

درکد فوق ابتدا از متد GetFiles کلاس Directory جهت قرار دادن آدرس کلیّهٔ عکسها در آرایه استفاده شده است. سپس به کمک کلاس Bitmap از روی آدرس عکس، خود عکس را به خاصیت Image بخش PictureBox نسبت داده ایم تا این عکس در آنجا نمایش داده شود. حال باید کدهای لازم جهت پیمایش آلبوم را در دکمه های متناظر قرار می دهیم. در زیر کدهای لازم جهت قرار دادن در رخدادهای Click دکمه های مختلف گنجانده شده است.

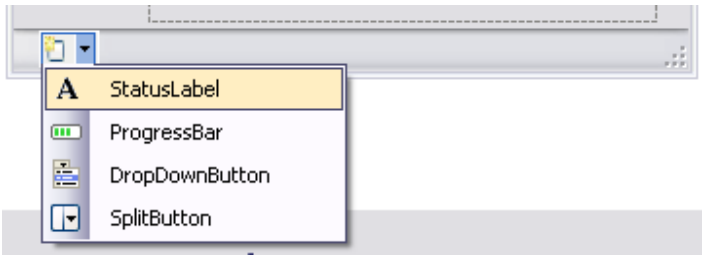
```
private void Next_Click(object sender, EventArgs e)
        {
            if (index < album.Length - 1)
              ++index;
            else
                  index=0;
            pictureBox1.Image = new Bitmap(album[index]);
        }
```

در مجموعه دستورات فوق، ابتدا چک می شود چنانچه به انتهای آرایه رسیده باشیم، اندیس دوباره به ابتدای آرایه برمی گردد ولی اگر به انتهای آرایه نرسیده باشیم،فقط اندیس یک واحد اضافه می گردد.

برای دکمه های Prev ، First و Last به همین ترتیب مجموعه دستورات زیر را می توان اضافه نمود:

```
      private void Prev_Click(object sender, EventArgs e)
        {
            if (index > 0)
                index--;
            else
                index =album.Length-1;
            pictureBox1.Image = new Bitmap(album[index]);
        }

        private void First_Click(object sender, EventArgs e)
        {
            index = 0;
            pictureBox1.Image = new Bitmap(album[index]);
        }

        private void Last_Click(object sender, EventArgs e)
        {
            index = album.Length - 1;
            pictureBox1.Image = new Bitmap(album[index]);
        }
```

حال برای نمایش اندازه و نام عکس در StatusBar باید سه Label به StatusBar اضافه نماییم. چگونگی انجام این عمل در زیر نمایش داده شده است:



همانگونه که مشاهده می شود، چهار شیء متفاوت را می توان به نوار وضعیت اضافه نمود. که اولین آنها Label می باشد.

```
private void GetPicStatus()
 {
  toolStripStatusLabel1.Text = "Width:" + pictureBox1.Image.Size.Width.ToString();
  toolStripStatusLabel2.Text = "Height:" + pictureBox1.Image.Size.Height.ToString();
  toolStripStatusLabel3.Text = "Name:" +
            album[index].Substring(album[index].LastIndexOf("\\") + 1);         }
```

با فراخوانی متد فوق در کلیهٔ رخدادها، مشخصات عکسها پس از نمایش عکس، در نوار وضعیت نمایان خواهد شد.

خروجی برنامه در زیر نشان داده شده است.

در نوار وضعیت علاوه بر مشخصات عکس، ساعت فعلی سیستم نیز گنجانده شده است. برای اخذ ساعت و تاریخ جاری می توانیم از کلاس DateTime استفاده نماییم. این کلاس همچنین دارای متدهای لازم جهت نمایش تاریخ و ساعت به فرمتهای مختلف می باشد. اما با توجه به اینکه ساعت سیستم بصورت لحظه ای تغییر می کند، باید مقدار ساعت را نیز بصورت لحظه ای در نوار وضعیت Refresh نماییم. برای انجام این کار عمل بازنویسی ساعت باید هر ثانیه یکبار صورت پذیرد که برای انجام این کار می توان از Timer استفاده نماییم. هر Timer دارای رخداد Tick می باشد که با توجه به مقدار قرار داده شده در خاصیت Interval(چنانچه برابر k باشد) ، هر K میلی ثانیه، یکبار این رخداد اجرا خواهد شد.

در رخداد Tick مشخص می نماییم که با هر بار اتفاق افتادن این رخداد مقدار ساعت جاری سیستم در Label موجود بر روی نوار وضعیت قرار داده شود.

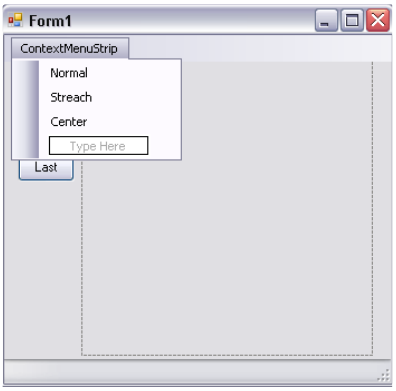
```
private void timer1_Tick(object sender, EventArgs e)
        {
            toolStripStatusLabel4.Text = DateTime.Now.ToLongTimeString();
        }
```

اما نباید فراموش کرد که مقدار Interval باید مشخص گردد و همچنین Timer نیز باید روشن شود.با نوشتن دستورات زیر در رخداد Form_Load می توان اعمال فوق را انجام داد:

```
timer1.Interval = 1000;
timer1.Start();
```

اضافه نمودن Context Menu به یک شی

منوهای فوق، منوهایی هستند که هنگام کلیک راست بر روی یک شی ظاهر می شوند، برای اضافه نمودن اینگونه منوها باید یک کامپوننت ContextMenuStrip را بر روی فرم بکشیم، سپس با کلیک بر روی آن گزینه های منو را انتخاب نماییم. در شکل زیر چگونگی انجام این کار دیده می شود.



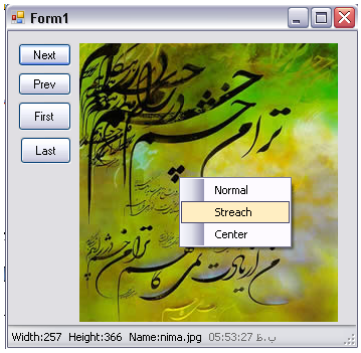
حال برای هر کدام از گزینه های اضافه شده باید کدهای لازم نوشته شود. کد لازم برای گزینه های فوق در زیر نشان داده شده است:

```
private void normalToolStripMenuItem_Click(object sender, EventArgs e)
  {
      pictureBox1.SizeMode = PictureBoxSizeMode.Normal;
  }

  private void streachToolStripMenuItem_Click(object sender, EventArgs e)
  {
      pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
  }

  private void centerToolStripMenuItem_Click(object sender, EventArgs e)
  {
      pictureBox1.SizeMode = PictureBoxSizeMode.CenterImage;
  }
```

حال باید مشخص نماییم که با کلیک راست بر روی چه عناصری، منوی ایجاد شده باید ظاهر گردد. برای این منظور باید خاصیت ContextMenuStrip عنصر موردنظر را برابر با منوی ایجاد شده قرار داد.

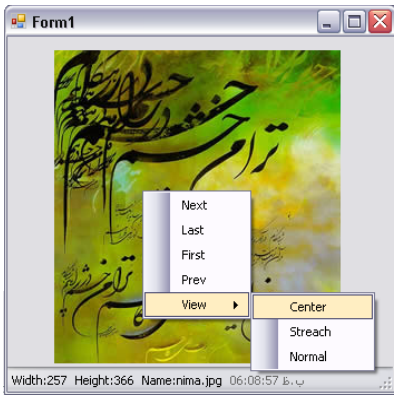
حال عنصر PictureBox را انتخاب نموده و خاصیت ContextMenuStrip آنرا برابر با ContextMenuStrip1 (منوی ایجاد شده) قرار دهید.

با انجام اعمال فوق می توان نحوهٔ نمایش عکس در PictureBox را در زمان اجرا با راست کلیک بر روی عکس تغییر داد. خروجی برنامه در شکل زیر دیده می شود.



تمرین۳: در منوی ظاهر شده یک ستون در سمت چپ قرار داده شده است که می توان آیکون مناسبی را برای هر کدام از گزینه های کلیک راست مشخص نمود. چگونه می توان این کار را انجام داد ؟

تمرین۴: دکمه های سمت چپ فرم فوق را حذف نموده و بگونه ای عمل نمایید که بتوان با کلیک راست بر روی عکس و انتخاب گزینه های بعدی، قبلی، اول و آخر آلبوم را پیمایش نمود.



تمرین۵: گزینه هایی را به منوی فوق اضافه نمایید تا بتوان عکس را با زاویه های متفاوت دوران داد.(از متد `RotateFlip` کلاس `Image` استفاده نمایید.)

گزینه ها بگونه ای باشند تا بتوان عکس را با زوایای ۹۰، ۱۸۰ و ۲۷۰ دوران داد.

استفاده از MenuStrip

در این بخش به چگونگی استفاده از منوهای استاندارد که در اغلب برنامه ها از آنها استفاده می شود، پرداخته می شود. مثال مورد استفاده در این قسمت همان آلبوم عکس مرحلهٔ قبل می باشد که می خواهیم آنرا بگونه ای دیگر تکمیل نماییم.

مثال۳) آلبوم عکس مرحلهٔ قبل را بگونه ای تکمیل نمایید تا دارای قابلیتهای نشان داده شده در زیر باشد.

اجزاء مختلف فرم فوق در زیر شرح داده شده اند:

Create: در این بخش می توان با انتخاب گزینهٔ Add Folder کلیهٔ فایلهای عکس درون آن پوشه را به آلبوم اضافه نمود.

View: با انتخاب این گزینه می توان آلبوم را پیمایش نموده و نحوهٔ نمایش عکسها را تعیین نمود.

Rotate: در این بخش می توانیم عکسها را با زاویه های متفاوت دوران داد.

About: با کلیک بر روی این گزینه فقط یک فرم جدید نمایش داده خواهد شد که در آن مشخصات نرم افزار آمده است.

در ادامه به تشریح چگونگی ایجاد اجزاء مختلف پرداخته می شود:

برای ایجاد بخش About ابتدا یک فرم از نوع AboutBox به پروژه اضافه می نماییم، سپس در رخداد کلیک گزینهٔ About دستورات لازم جهت نمایش فرم ایجاد شده را می نویسیم.

```
AboutBox1 a = new AboutBox1(); // ایجاد یک شی از نوع فرم اضافه شده
a.Show();// نمایش فرم
```

Create: در این بخش ابتدا یک شی از نوع FolderBrowserDialog ایجاد می نماییم، دستورات لازم جهت نمایش دیالوگ را می نویسیم. پس از آن با توجه به دایرکتوری انتخاب شده توسط کاربر کلیهٔ فایلهای از نوع عکس آن دایرکتوری را به آلبوم اضافه می نماییم.

```
private void addFolderToolStripMenuItem_Click(object sender, EventArgs e)
   {
        FolderBrowserDialog fd = new FolderBrowserDialog();
                    //ابتدا یک شی از نوع FolderBrowserDialog تعریف می کنیم
        fd.RootFolder = Environment.SpecialFolder.Desktop;
```

```
                                        تعیین نمودن پوشه پیش فرض//
            fd.ShowNewFolderButton = false;
                                        حذف گزینه ایجاد پوشه جدید //
            if (fd.ShowDialog() == DialogResult.OK)//چک نمودن کلید انتخاب شده توسط کاربر
            {
                string[] NewFiles = Directory.GetFiles(fd.SelectedPath,"*.jpg");
                 //انتخاب فایلهای تصویری درون پوشه انتخاب شده
                foreach (string f in NewFiles)
                    al.Add(f);// اضافه نمودن فایلهای فایلها به آلبوم
            }
        }
```

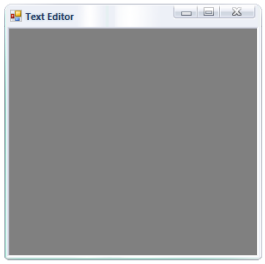
تمرین ۶: دو بخش دیگر را بگونه ای تکمیل نمایید تا امکان دوران عکس و پیمایش عکسها قابل انجام باشد.

آشنایی با دیالوگ ها و فرمهای MDI

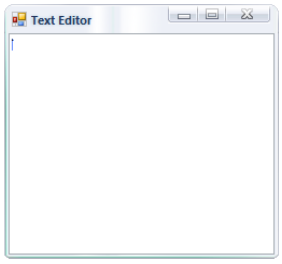
در این قسمت با دیالوگهای موجود در .Net و فرمهای MDI آشنا می شوید.

مثال۴) یک برنامه Notepad ساده ایجاد نمایید که قابلیت ایجاد، ذخیره، کپی نمودن و ... فایلهای متنی را داشته باشد.

برای انجام این مثال ابتدا باید یک فرم از نوع MDI ایجاد نماییم. برای انجام این کار کافی است یک فرم ساده به پروژه اضافه نموده، سپس خاصیت IsMdiContainer را برابر با true قرار دهید. پس از آن نمای فرم بصورت زیر در خواهد آمد.



فرم MDI

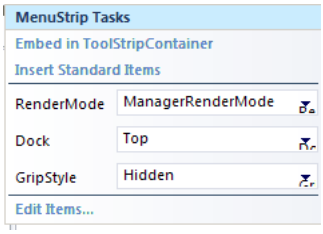


فرم MDI با TextBox

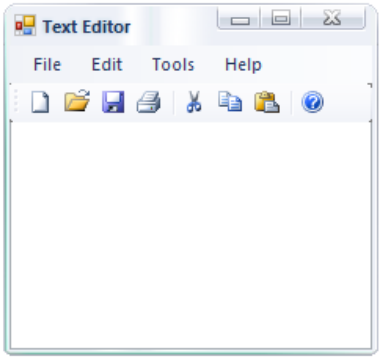
حال یک TextBox به فرم اضافه نمایید و خاصیت Multiline آنرا برابر با true قرار دهید. همچنین خاصیت Dock آنرا برابر با Fill قرار دهید تا کل فضای فرم را پر نماید. که شکل فرم پس از اضافه نمودن TextBox و تنظیمات فوق بصورت بالا در خواهد آمد.

حال می خواهیم گزینه های استاندارد نرم افزارهای خانواده Office را به منوی بالای آن اضافه نماییم، برای این منظور مراحل زیر را انجام می دهیم.

ابتدا یک MenuStrip به فرم اضافه می شود. بعد از آن با زدن علامت فلش کوچک که در سمت راست بالای آن ظاهر می گردد، شکل زیر نمایان می شود. در این بخش با کلیک بر روی عبارت Insert Standard Item منوی استاندارد به بالای فرم افزوده می شود.



همچنین می توانیم با انتخاب ToolStrip و انجام مراحل فوق دکمه های استاندارد را به بالای فرم و زیر منوهای استاندارد اضافه نماییم که پس از افزودن عناصر دو مرحله قبل، فرم بصورت زیر در می آید.



حال با توجه با کارکرد برنامه می توانیم دکمه ها و منوهای اضافی را حذف نموده و یا اینکه گزینه جدیدی اضافه نماییم.کدهای لازم برای دکمه های استاندارد New، Open، Save، Copy، Cut و Paste را می نویسیم.

دکمه Open

با زدن دکمه Open و یا با انتخاب Open از منوی فایل، باید بتوانیم یک فایل متنی را بازنموده و محتویات آنرا در TextBox نمایش دهیم.

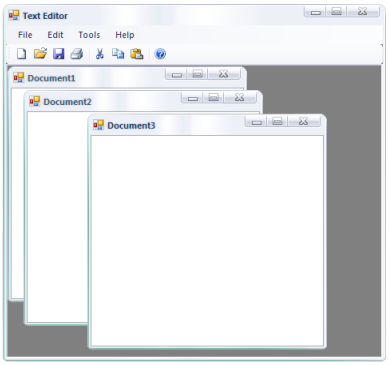
برای این منظور از OpenFileDialog استفاده می نماییم بگونه ای که با کلیک بر روی دکمه Open فایل باز شده و محتوای آن خوانده شده و در TextBox قرار بگیرد.

دستورات زیر را به رخداد کلیک Open اضافه نمایید.

```
OpenFileDialog od = new OpenFileDialog();
od.Filter = "Text files (*.txt)|*.txt"; // چه فایلهایی نمایش داده شوند
od.InitialDirectory = "c:\\";    // کدام دایرکتوری نمایش داده شود؟
if (od.ShowDialog() == DialogResult.OK) // نمایش دیالوگ
  {
     StreamReader sr = new StreamReader(od.FileName); // باز کردن فایل انتخاب شده بصورت متنی
     this.textBox1.Text = sr.ReadToEnd();// خواندن کل فایل و نمایش آن
     sr.Close();// بستن فایل
  }
```

دکمه New

با انتخاب این دکمه یک فرم باید بجای TextBox ظاهر گردد. برای این منظور ابتدا TextBox را حذف نموده و دستورات زیر را در رخداد کلیک گزینه New اضافه نمایید. با هر بار کلیک نمودن گزینه New باید یک فرم جدید اضافه شود. نمونه ای از خروجی برنامه در شکل زیر نمایش داده شده است.



به ابتدای کلاس متغیرهای زیر را اضافه نمایید:

int count;// از این متغیر جهت تعیین شماره فرم باز شده استفاده می شود.

Form mdiChild;//فرمی که درون فرم موجود باز می شود.

TextBx textedit; //مشابه حالت قبل برای ورود و نمایش اطلاعات در فرم بکار می رود.

مقدار متغیر Count را در سازنده کلاس برابر با ۱ قرار دهید و با هر بار ایجاد یک فرم جدید به آن یک واحد اضافه نمایید.

```
private void newToolStripMenuItem_Click(object sender, EventArgs e)
        {
            mdiChild = new Form(); // ایجاد شی فرم
            mdiChild.Text = "Document" + count.ToString();// تعیین عبارت بالای فرم
            mdiChild.MdiParent = this; // تعیین فرم ایجاد شده بعنوان فرزند فرم اصلی
            edittxt = new TextBox();
            edittxt.Multiline = true;
            edittxt.Dock = DockStyle.Fill;
            mdiChild.Controls.Add(edittxt);
            mdiChild.Show();
            count++;// افزایش شمارنده
        }
```

دکمه Save

در این قسمت با کلیک دکمه Save ، محتوای فرم فرزند فعال باید در یک فایل متنی بر روی حافظه جانبی ذخیره گردد. کدهای لازم برای انجام این کار در زیر آمده است:

```
        private void saveToolStripMenuItem_Click(object sender, EventArgs e)
        {
            SaveFileDialog sd = new SaveFileDialog();
            sd.Title = "Save File!";
```

```
        sd.Filter = "Text files (*.txt)|*.txt";
        sd.InitialDirectory = "c:\\";
        if (sd.ShowDialog() == DialogResult.OK)
        {
            StreamWriter sw = new StreamWriter(sd.FileName);
            Form activeChildForm = this.ActiveMdiChild;//تعیین فرم فعال
            if (activeChildForm != null)
            {
               TextBox activetextbox=(TextBox)activeChildForm.ActiveControl;
               if (activetextbox != null)
                   sw.Write(activetextbox.Text);
           }
           sw.Close();
        }
    }
```

دکمه Open

در این قسمت با کلیک دکمه Open، باید محتوای فایل انتخاب شده از روی حافظه جانبی را در فرم فعال نمایش دهیم. کدهای لازم برای انجام این کار در زیر آمده است:

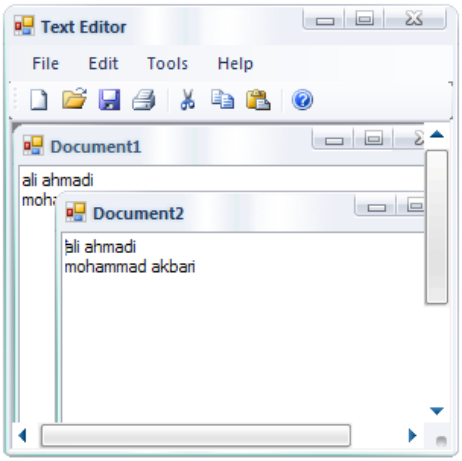
```
    private void openToolStripMenuItem_Click(object sender, EventArgs e)
    {
        OpenFileDialog od = new OpenFileDialog();
        od.Filter = "Text files (*.txt)|*.txt";
        od.InitialDirectory = "c:\\";
        if (od.ShowDialog() == DialogResult.OK)
        {
            StreamReader sr = new StreamReader(od.FileName);
            Form activeChildForm = this.ActiveMdiChild;
            if (activeChildForm != null)
            {
                TextBox activetextbox=(TextBox)activeChildForm.ActiveControl;
                if (activetextbox!=null)
                    activetextbox.Text=sr.ReadToEnd();
            }
            sr.Close();
        }
    }
```

دکمه های Copy و Paste

برای انجام دو عمل فوق، ابتدا یک متغیر از نوع رشته ای در ابتدای کلاس تعریف می نماییم، سپس هنگام کپی اطلاعات TextBox را در آن متغیر ذخیره نموده و سپس هنگام Paste نمودن محتوای متغیر را در TextBox قرار می دهیم.

```
private void copyToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form activeChildForm = this.ActiveMdiChild;
    if (activeChildForm != null)
    {
        TextBox activetextbox = (TextBox)activeChildForm.ActiveControl;
        if (activetextbox != null)
            s = activetextbox.Text;
    }
}

private void pasteToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form activeChildForm = this.ActiveMdiChild;
    if (activeChildForm != null)
    {
        TextBox activetextbox = (TextBox)activeChildForm.ActiveControl;
        if (activetextbox != null)
            activetextbox.Text = s;
    }
}
```

خروجی برنامه در شکل زیر نشان داده شده است:



تمرین۷: کدهای لازم برای دکمه های Cut، Select All، Save As، Undo، Redo و About را بنویسید.

تمرین۸: دکمه Help را بصورت مناسبی برای نرم افزار بازنویسی نمایید.

برای انجام این کار ابتدا راهنمای برنامه را در یک فایل HTML بنویسید، سپس کاری انجام دهید که با کلیک دکمه Help فایل HTML نمایش داده شود.

تمرین۸: برنامه ای بنویسید که مشابه شکل زیر دارای دو Split Container باشد که هر کدام حاوی یک Tab Control بوده و در یکی از این دو Split Container مشخصات رشته ها و در دیگری مشخصات گرایش های رشته ها وجود داشته باشد. با انتخاب هر رشته کلیه گرایشهای رشته در Split Container دیگر نمایش داده شود.

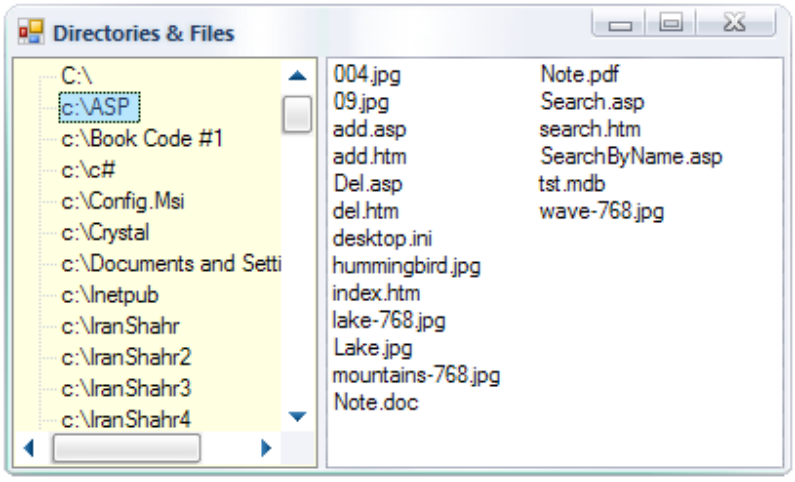
تمرین۹: تمرین شماره ۸ را بگونه ای توسعه دهید تا در لیست گرایشها این امکان وجود داشته باشد تا با کلیک هر گرایش، یک فرم دیگر باز شده و مشخصات دقیق تری از گرایش را نمایش دهد. اطلاعات دقیق تر عبارتند از:

- تعداد واحد
- هدف از ایجاد گرایش
- توانایی های فارغ التحصیلین
- موقعیت شغلی در ایران

مثال۵) کار با TreeView و ListView

در این مثال از یک TreeView و از یک ListView استفاده می نماییم. از TreeView برای نمایش لیست دایرکتوریها و از ListView برای برای نمایش فایلهای یک دایرکتوری استفاده می کنیم.

یک نمونه از خروجی برنامه در زیر نشان داده شده است:



برای انجام مثال فوق، از یک SplitContainer استفاده می نماییم که سمت چپ وی یک TreeView و سمت راست آن حاوی یک ListView می باشد. پس از اضافه نمودن کامپوننت های فوق، کدهای زیر را نیز اضافه می نماییم.

```
private void Form1_Load(object sender, EventArgs e)
    {

        treeView1.Nodes.Add("C:\\");
        TreeRefresh("c:\\", treeView1.Nodes[0]);

    }
```

در کد فوق ابتدا c: بعنوان ریشه TreeView اضافه می گردد. سپس متد نوشته شده در زیر فراخوانی می گردد تا لیست کلیه زیردایرکتوریها را به ساختار درختی اضافه نماید.

```
    private void TreeRefresh(string str, TreeNode tn)
    {
        string[] dir = Directory.GetDirectories(str);
        foreach (string d in dir)
        {
            TreeNode treen = new TreeNode(d);
            treeView1.Nodes.Add(treen);
        }
    }
```

دستورات زیر را در رخداد TreeView_AfterSelect نوشته ایم تا با انتخاب یکی از دایرکتوریها از سمت چپ SplitContainer ساختار سمت چپ دوباره Refresh گردد، سپس متد نوشته شده بعدی نیز فراخوانی می گردد تا لیست فایلهای دایرکتوری انتخاب شده را در سمت راست نمایش دهد.

```
    private void treeView1_AfterSelect_1(object sender, TreeViewEventArgs e)
    {
        TreeRefresh(e.Node.Text, e.Node);
        ListRefresh(e.Node.Text);
    }


    private void ListRefresh(string str)
    {
        listView1.Items.Clear();
        listView1.View = View.List;
        FileInfo[] files = new DirectoryInfo(str).GetFiles();
        foreach (FileInfo fi in files)
        {
            ListViewItem f = listView1.Items.Add(fi.Name);
            f.ImageIndex = f.Index;
        }
     }
```

تمرین۱۰: مثال فوق را بگونه ای بازنویسی نمایید تا برنامه دارای قابلیتی باشد تا بتوانیم هر دایرکتوری دلخواهی را انتخاب نموده و لیست دایرکتوریها و فایلهای آنرا نمایش دهیم.(دایرکتوریها در ساختار درختی و فایلها در ListView).

تمرین ۱۱: در تمرین زیر از یک PictureBox، Button و GroupBox استفاده شده است که در آن با انتخاب هر فایلی به کمک دکمه Browse در GroupBox مشخصات عکس و در PictureBox خود عکس نمایش داده شود.(Scrollها باید فعال باشند)

وماتوفیقی الا بالله

بلوچ زهی

۸۵/ ۱۰ / ۲۰

گروه مهندسی فناوری اطلاعات

کوئیز شماره یک برنامه سازی پیشرفته گروه یک

نام و نام خانوادگی: **شماره دانشجویی:**

---

۱) خروجی متد بازگشتی زیر را به ازای مقادیر داده شده محاسبه نمایید.

```
static long tst(int p, int q)
{
  if (q <= 0)
      return 1;
  long r = tst(p, q / 2);
  r *= r;
  if (q % 2 == 0)
    return r;
  else
     return r * p;
 }
     a. tst( 2 , 3 )= ?                b. tst ( 3,5 ) =?
```

۲)عملکرد متد بازگشتی زیر را تعیین نمایید. سپس مقدار چاپ شده توسط متد را با توجه به ورودی های داده شده محاسبه نمایید.

```
static void Reverse(string s, int i)
{
  if (i < s.Length-1)
     Reverse(s, i + 1);
   Console.Write(s[i]);
 }

Reverse(“ABCD”, 0 )= ?          Reverse ( “ABCD”, 2) = ?
```

۳)خروجی متد بازگشتی زیر را به ازای tst (72 , 2) محاسبه نمایید. سپس معادل غیر بازگشتی آنرا بنویسید.

```
static void tst(int n, int k)
  {
  if (n == 1) return;
  else
   {
     int c = 0;
     while (n % k == 0)
       {
         c++;
         n/=k;
        }
        if (c > 0)
           Console.WriteLine("{0}, {1}", k, c);
    }
    tst(n, k + 1);
   }
```

۴)متدی بازگشتی بنویسید که با دریافت عددی از ورودی مجموع ارقام آنرا چاپ نماید.

گروه مهندسی فناوری اطلاعات

کوئیز شماره سه برنامه سازی پیشرفته گروه یک

نام و نام خانوادگی: **شماره دانشجویی:**

۱) برنامه ای بنویسید که با دریافت نام فایل متنی از سطر فرمان، به ازای هر خط از فایل شماره خط و تعداد کلمات غیر تکراری آن خط را چاپ نماید.

۲)کلاسی تعریف نمایید که از آن بتوان جهت نمایش چند جمله ای یک متغیره استفاده نمود. سپس عملگرهای داده شده زیر را بگونه ای سربارگذاری نمایید که هر کدام بتوانند اعمال خواسته شده را انجام دهند.

+ : جمع دو چند جمله ای

-: تفاضل دو چند جمله ای

-: محاسبه قرینهٔ یک چند جمله ای، فقط ضرایب قرینه می شوند.

==: چک تساوی دو چند جمله ای

>: جهت مقایسهٔ دو چند جمله ای

نکته: جهت نمایش چند جمله ای یک متغیره می توان از Sorted List استفاده نمود، بگونه ای که توان در بخش Key و ضریب در بخش Value قرار گیرد.

موفق و موید باشید
بلوچ زهی

گروه مهندسی فناوری اطلاعات

کوئیز شماره سه برنامه سازی پیشرفته گروه دو

نام و نام خانوادگی: **شماره دانشجویی:**

۱) برنامه ای بنویسید که با دریافت نام دو فایل متنی از سطر فرمان، چک نماید که آیا دو فایل عکس همدیگر هستند یانه؟ دو فایل عکس همدیگر هستند اگر کارکتر اول یکی با کارکتر آخر دیگری و کارکتر دوم با ماقبل آخر و ... برابر باشند. درصورت برابر بودن برنامه باید True و درغیر اینصورت False را چاپ نماید.

۲)کلاسی تعریف نمایید که از آن بتوان جهت نمایش اعدادصحیح بزرگ بدون علامت استفاده نمود. سپس عملگرهای داده شده زیر را بگونه ای سربارگذاری نمایید که هر کدام بتوانند اعمال خواسته شده را انجام دهند. **+** : جمع دو عدد بزرگ، **==**: چک تساوی دو عدد بزرگ، **<**: چک می نماید که عدد اول از دومی کوچکتر است یا خیر؟

راهنمایی: جهت نمایش اعداد بزرگ می توان از ArrayList استفاده نمود، بگونه ای که هر رقم عدد در یک خانه از ArrayList قرار بگیرد.

از پشت صفحه به عنوان پاسخنامه استفاده نمایید.

موفق و موید باشید

بلوچ زهی

۸۵ / ۹ / ۲۹

University of Canberra
Division of Business, Law and Information Sciences
School of Information Sciences and Engineering

Programming Graphical User Interfaces PG

Drawing Basics

1. The **Graphics** class: provides pens, brushes and drawing methods. The **Graphics** class is included in the **System.Drawing** namespace. This namespace is implemented on top of **Graphics Device Interface Plus (GDI+)**. **GDI+** is a Win32DLL (gdiplus.dll).

2. The **try-finally** and **using** blocks:

   The code for drawing on the Windows form is of the form

```
Graphics g = this.CreateGraphics();
try
{
      // code for drawing
}
finally
      g.Dispose();
```

   The **try-finally** block is used to release an underlying resource hold by the graphics object when we finish drawing. The resource is managed by Windows when the graphics object is created

   The **try-finally** block can be replaced by a **using** block as follows

```
using (Graphics g = this.CreateGraphics())
{
      // code for drawing
}
```

   The **using** block always calls the **Dispose()** method at the end of the block for objects created in the **using** block

3. Drawing on a control: most of control objects have their own `CreateGraphics` method
   - Draw on Windows form: `Graphics g = this.CreateGraphics();`
   - Draw on button1: `Graphics g = this.button1.CreateGraphics();`

4. If drawing methods are implemented in the **Paint** event handler, use the **Graphics** object in the paint event argument **e**

```
Graphics g = e.Graphics;
```

5. If drawing methods are **not** implemented in the **Paint** event handler, use the **CreateGraphics()** method as shown above and the method **Refresh()** to call the Paint event and update the form. For example

```
private void MainForm_MouseMove(object sender,
      System.Windows.Forms.MouseEventArgs e)
{
      using (Graphics g = this.CreateGraphics())
      {
            g.DrawRectangle(Pens.Black, 20, 20, 250, 120);
            this.Refresh();
      }
}
```

   The **Refresh()** method can be regarded as the combination of the **Invalidate()** method (to ask Windows for a Paint event for the Windows form) and the **Update()** method (to force the Paint event to happen)

6. If drawings on the form are changed when **resizing** the Windows form, add the following code to the constructor (after the **InitializeComponent()** method)

```
this.SetStyle(ControlStyles.ResizeRedraw, true);
```

7. To reduce **flicker** when drawing with mouse, add the following to the constructor

```
this.SetStyle(ControlStyles.DoubleBuffer, true);
this.SetStyle(ControlStyles.UserPaint, true);
this.SetStyle(ControlStyles.AllPaintingInWmPaint, true);
```

   Using **ControlStyles.DoubleBuffer** and **true**, drawing is performed in a buffer, and after it completes, the result is output to the screen. Double-buffering prevents flicker caused by the redrawing of the control. To fully enable double-buffering, you must also set the **UserPaint** and **AllPaintingInWmPaint** style bits to **true**.

University of Canberra
Division of Business, Law and Information Sciences
School of Information Sciences and Engineering

Programming Graphical User Interfaces PG

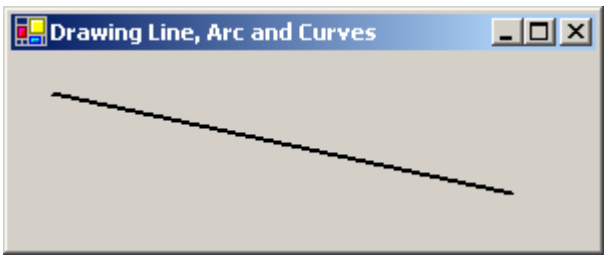
Drawing Methods (including Drawing Text)

The following examples show how to use all Drawing methods available in the Drawing and Drawing 2D namespaces. Code is added to the **Paint** event handler. The form name is **MainForm**.

1. **Drawing Line**:

Syntax: `DrawLine(Pen pen, Point pt1, Point pt2);`

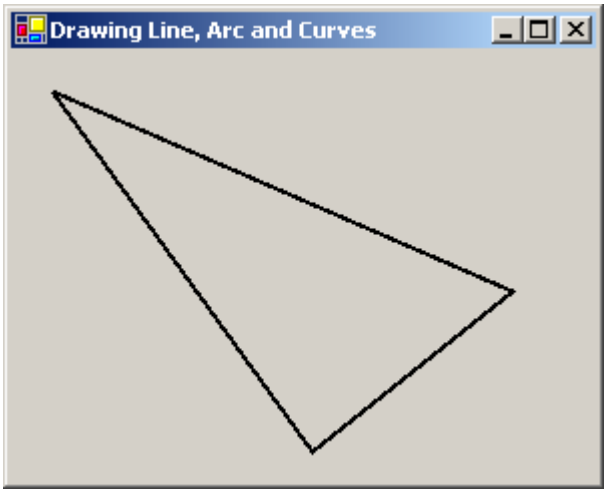
```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      Pen myPen = new Pen(Color.Black, 2);
      Point pt1 = new Point(20, 20);
      Point pt2 = new Point(250, 70);

      using (Graphics g = e.Graphics)
            g.DrawLine(myPen, pt1, pt2);
}
```



2. **Drawing Lines**:

Syntax: `DrawLines(Pen pen, Point [] points);`

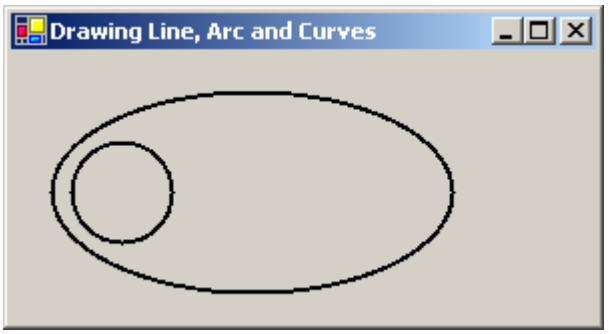
```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      Pen myPen = new Pen(Color.Black, 2);
      PointF[] points =
      {
            new PointF(20.0F, 20.0F),
            new PointF(250.0F, 120.0F),
            new PointF(150.0F, 200.0F),
            new PointF(20.0F, 20.0F)
      };

      using (Graphics g = e.Graphics)
            g.DrawLines(myPen, points);
}
```



3. **Drawing Ellipse and Circle**:

Syntax: `DrawEllipse(Pen pen, int x, int y, int width, int height);`

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      Pen myPen = new Pen(Color.Black, 2);
      using (Graphics g = e.Graphics)
      {
            g.DrawEllipse(myPen, 20, 20, 200, 100);
            g.DrawEllipse(myPen, 30, 45, 50, 50);
      }
}
```

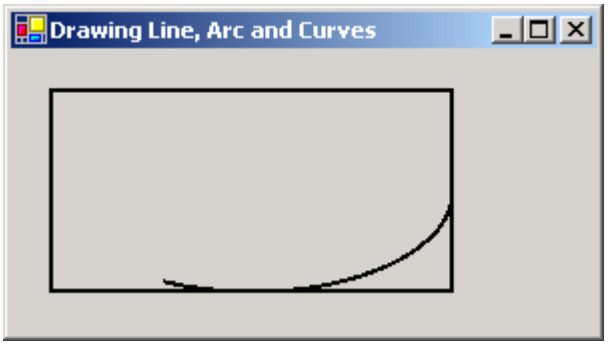


4. **Drawing Arc**:

Syntax: `DrawArc(Pen pen, Rectangle rect, float startAngle, float sweepAngle);`

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {
            Pen myPen = new Pen(Color.Black, 2);
            Rectangle rect = new Rectangle(20, 20, 200, 200);
            g.DrawRectangle(myPen, 20, 20, 200, 200);
```

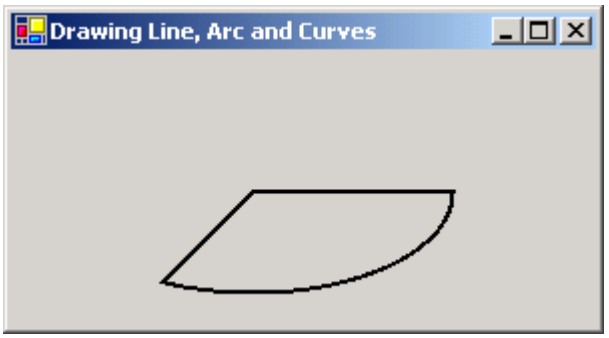
```
            g.DrawArc(myPen, rect, 0.0f, 135.0f);
        }
}
```



5. **Drawing Pie**:

Syntax: `DrawPie(Pen pen, Rectangle rect, float startAngle, float sweepAngle);`

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {
            Pen myPen = new Pen(Color.Black, 2);
            Rectangle rect = new Rectangle(20, 20, 200, 200);
            g.DrawPie(myPen, rect, 0.0f, 135.0f);
      }
}
```

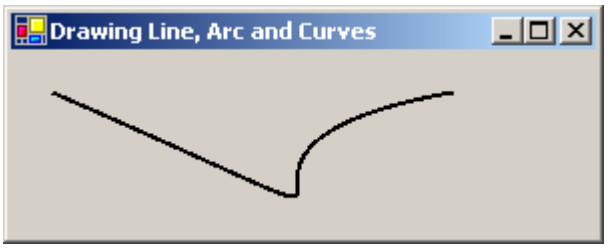


6. **Drawing Bezier**: A Bézier spline is a curve specified by four points: two end points and two control points. The curve begins at *pt1* and ends at *pt2*. The curve doesn't pass through the control points, but the control points act as magnets, pulling the curve in certain directions and influencing the way the curve bends

Syntax: `DrawBezier(Pen pen, Point pt1, Point pt2, Point pt3, Point pt4);`

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {
```

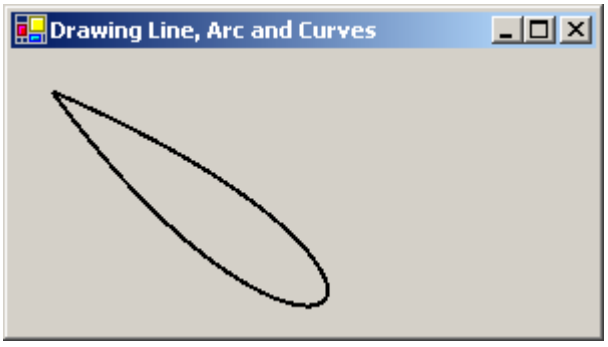
```
            Pen myPen = new Pen(Color.Black, 2);
            g.DrawBezier(myPen,
                  new PointF(20.0F, 20.0F),
                  new PointF(250.0F, 120.0F),
                  new PointF(50.0F, 50.0F),
                  new PointF(220.0F, 20.0F) );
      }
}
```



7. **Drawing Beziers**:

Syntax: `DrawBeziers(Pen pen, Point [] points);`

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      Pen myPen = new Pen(Color.Black, 2);
      PointF[] points =
      {
            new PointF(20.0F, 20.0F),
            new PointF(250.0F, 120.0F),
            new PointF(150.0F, 200.0F),
            new PointF(20.0F, 20.0F)
      };
      using (Graphics g = e.Graphics)
            g.DrawBeziers(myPen, points);
}
```

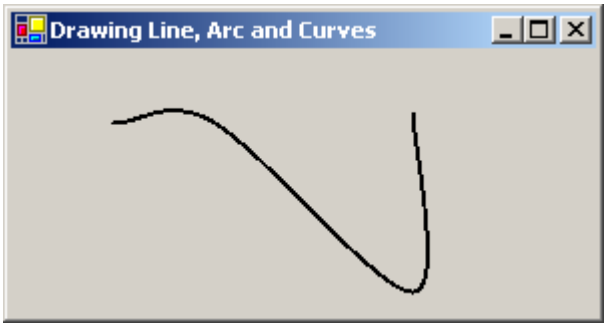


8. **Drawing Curve**:

Syntax: `DrawCurve(Pen pen, Point [] points, int offset, int numberOfSegments, float tension);`

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      Pen myPen = new Pen(Color.Black, 2);
```

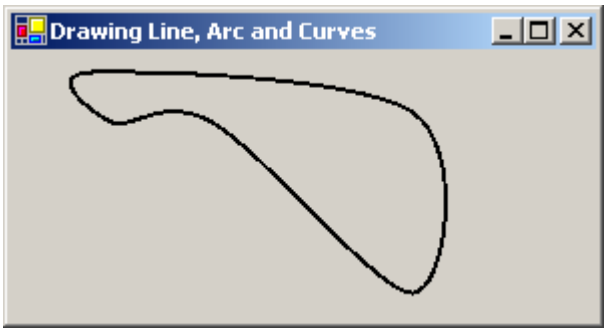
```
        float tension = 0.5F;
        PointF[] points =
        {
                new PointF(40.0F, 10.0F),
                new PointF(50.0F, 35.0F),
                new PointF(100.0F, 35.0F),
                new PointF(200.0F, 120.0F),
                new PointF(200.0F, 30.0F)
        };
        int offset = 1;
        int segments = 3;

        using (Graphics g = e.Graphics)
                g.DrawCurve(myPen, points,
                        offset, segments, tension);
}
```



9. **Drawing Closed Curve**: requires the **Drawing2D** namespace

Syntax: `DrawClosedCurve(Pen` ***pen***`, Point [] ***points***`, float` ***tension***`, FillMode` ***mode***`);`

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
        Pen myPen = new Pen(Color.Black, 2);
        float tension = 0.5F;
        PointF[] points =
        {
                new PointF(40.0F, 10.0F),
                new PointF(50.0F, 35.0F),
                new PointF(100.0F, 35.0F),
                new PointF(200.0F, 120.0F),
                new PointF(200.0F, 30.0F)
        };

        using (Graphics g = e.Graphics)
                g.DrawClosedCurve(myPen, points,
                        tension, FillMode.Alternate);
}
```



10. **Drawing Rectangle**:

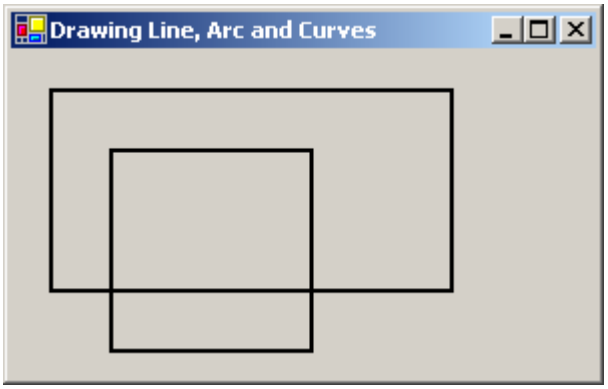
Syntax: `DrawRectangle(Pen pen, int x, int y, int width, int height);`

Example: *refer to example 8*

11. **Drawing Rectangles**:

Syntax: `DrawRectangles(Pen pen, Rectangle [] rects);`

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      Rectangle rect1 = new Rectangle(20, 20, 200, 100);
      Rectangle rect2 = new Rectangle(50, 50, 100, 100);

      Rectangle [] rects = {rect1, rect2};

      using (Graphics g = e.Graphics)
            g.DrawRectangles(myPen, rects);
}
```

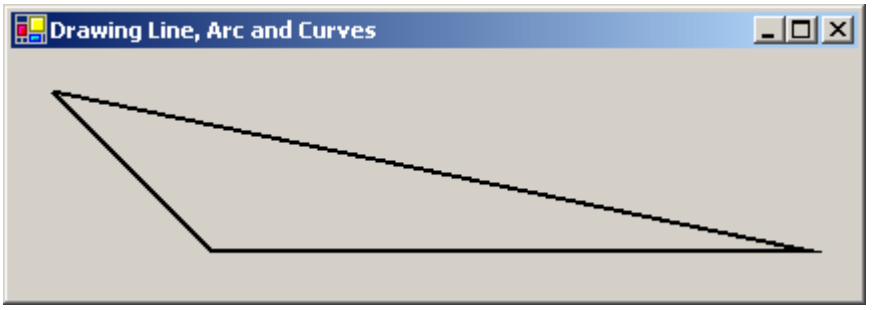


12. **Drawing Path**: requires the **Drawing2D** namespace

Syntax: `DrawPath(Pen pen, GraphicsPath path);`

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      GraphicsPath path = new GraphicsPath();
      path.AddLine(20, 20, 100, 100);
```

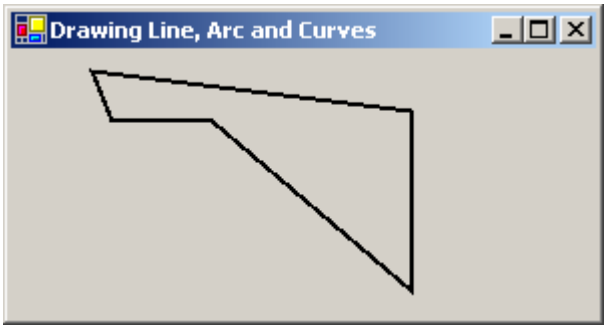
```
        path.AddLine(100, 100, 400, 100);
        path.AddLine(400, 100, 20, 20);
        using (Graphics g = e.Graphics)
              g.DrawPath(myPen, path);
}
```



13. **Drawing Polygon**:

Syntax: `DrawPolygon(Pen pen, Point [] points);`

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
        Graphics myGraphics = e.Graphics;
        PointF[] points =
        {
              new PointF(40.0F, 10.0F),
              new PointF(50.0F, 35.0F),
              new PointF(100.0F, 35.0F),
              new PointF(200.0F, 120.0F),
              new PointF(200.0F, 30.0F)
        };
        using (Graphics g = e.Graphics)
              g.DrawPolygon(myPen, points);
}
```

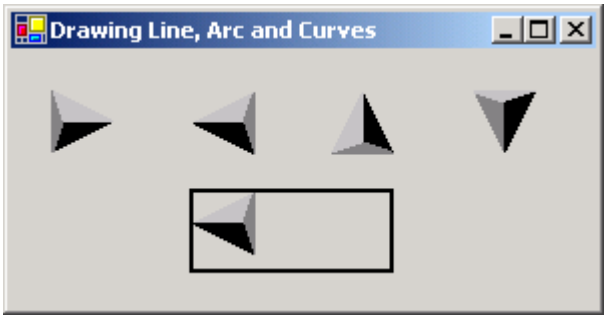


14. **Drawing Icon**:

Syntax: `DrawIcon(Icon icon, Rectangle targetRect);`

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
        Icon icon1 = new Icon("ARW03RT.ICO");
        Icon icon2 = new Icon("ARW03LT.ICO");
        Icon icon3 = new Icon("ARW03UP.ICO");
```

```
        Icon icon4 = new Icon("ARW03DN.ICO");

        using (Graphics g = e.Graphics)
        {
              g.DrawIcon(icon1, 20, 20);
              g.DrawIcon(icon2, 90, 20);
              g.DrawIcon(icon3, 160, 20);
              g.DrawIcon(icon4, 230, 20);
              Rectangle rect = new Rectangle(90, 70, 100, 40);
              g.DrawRectangle(myPen, rect);
              g.DrawIconUnstretched(icon2, rect);
        }
}
```



DrawIconUnstretched method: draws the image represented by the specified Icon object without scaling the image

15. **Drawing Image**:

Syntax:

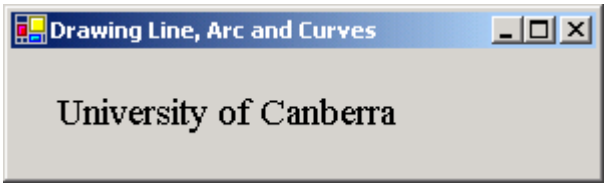
```
DrawImage(Image image, Point point);
DrawImageUnscaled(Image image, Point point);
```

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
        Image image =
              Image.FromFile("summer2004_synchro_swim.gif");
        Point displayPoint = new Point(20, 20);
        using (Graphics g = e.Graphics)
              g.DrawImage(image, displayPoint);
}
```

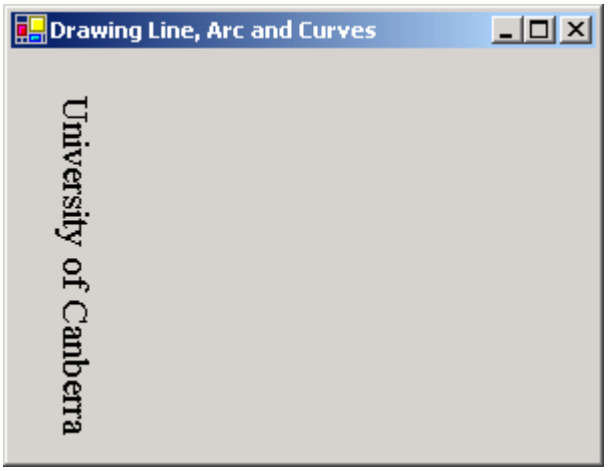
16. **Drawing Text**:

Syntax: `DrawString(String s, Font f, Brush b, Rectangle r, StringFormat format);`

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      SolidBrush myBrush = new SolidBrush(Color.Black);
      Rectangle myRect = new Rectangle(20, 20, 200, 40);
      Font myFont = new Font("Times New Roman", 14);

      using (Graphics g = e.Graphics)
            g.DrawString("University of Canberra",
                  myFont, myBrush, myRect);
}
```




```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      SolidBrush myBrush = new SolidBrush(Color.Black);
      Rectangle myRect = new Rectangle(20, 20, 200, 40);
      Font myFont = new Font("Times New Roman", 14);

      StringFormat format = new StringFormat();
      format.FormatFlags =
            StringFormatFlags.DirectionVertical;

      using (Graphics g = e.Graphics)
            g.DrawString("University of Canberra",
                  myFont, myBrush, myRect, format);
}
```



University of Canberra
Division of Business, Law and Information Sciences
School of Information Sciences and Engineering

Programming Graphical User Interfaces PG

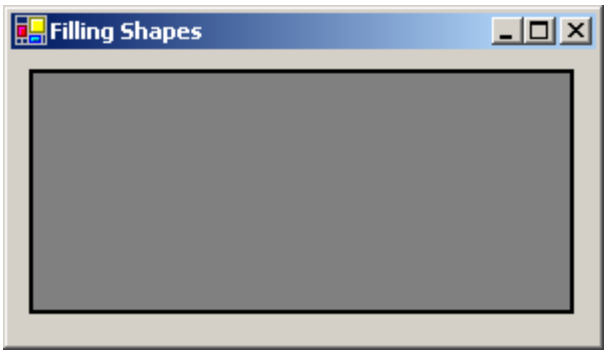
Filling Shapes

- Any of the shapes in the `Graphics` library that contain an *interior* can also be filled. Simple shapes such as rectangles and ellipses have obvious interiors. If a filling operation is applied to an *open* curve, `Graphics` will add an extra line segment to close the curve.
- Drawing a shape requires a `Pen` and filling a shape requires a `Brush`.
- Note that filling and drawing are two separate operations. For example, `FillRectangle()` produces a rectangle containing only the filled interior (without any outline of the edges) of the rectangle. To produce a filled interior and a drawn outline, we need to call `FillRectangle()` and *then* `DrawRectangle()` *in this order*.
- Five brushes available for filling shapes are SolidBrush, HatchBrush, TextureBrush, LinearGradientBrush and PathGradientBrush.
- Two filling modes available for filling a more complex shape (e.g. a polygon) are `FillPolygon()` and `FillPath()`.

The following examples can be placed in the Paint event handler of a Windows Application project using C#. The project name is FillingShapes and the Form1 is replaced with MainForm.

1. Filling a rectangle or an ellipse with the SolidBrush

```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using(Graphics g = e.Graphics)
      {
            Pen pen = new Pen(Color.Black, 2);
            SolidBrush brush = new SolidBrush(Color.Gray);
            g.FillRectangle(brush, 10, 10, 270, 120);
            g.DrawRectangle(pen, 10, 10, 270, 120);
      }
}
```

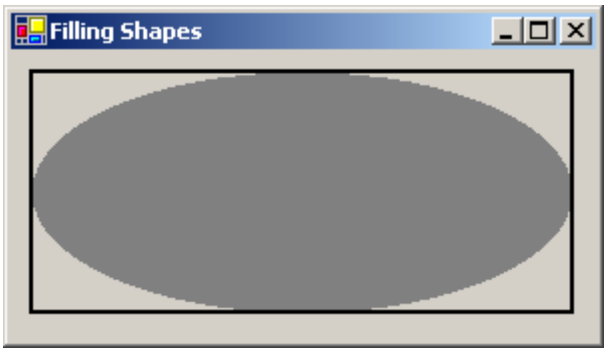


We can create a Rectangle object before calling filling and drawing operations as follows

```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using(Graphics g = e.Graphics)
      {
            Pen pen = new Pen(Color.Black, 2);
            SolidBrush brush = new SolidBrush(Color.Gray);
            Rectangle rect = new Rectangle(10, 10, 270, 120);
            g.FillRectangle(brush, rect);
            g.DrawRectangle(pen, rect);
      }
}
```

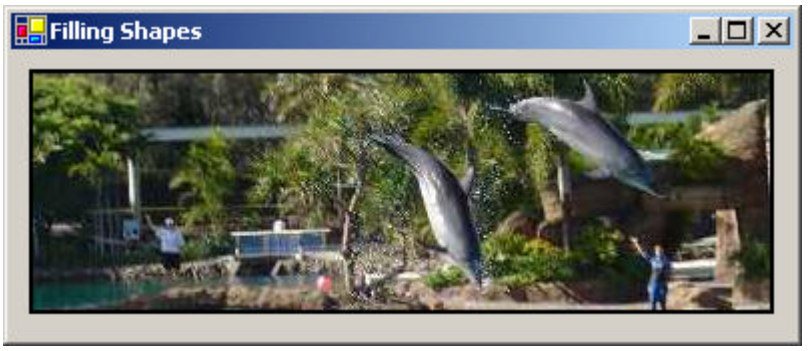
The Rectangle object can be used in the `FillEllipse()` as follows

```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using(Graphics g = e.Graphics)
      {
            Pen pen = new Pen(Color.Black, 2);
            SolidBrush brush = new SolidBrush(Color.Gray);
            Rectangle rect = new Rectangle(10, 10, 270, 120);
            g.FillEllipse(brush, rect);
            g.DrawRectangle(pen, rect);
      }
}
```

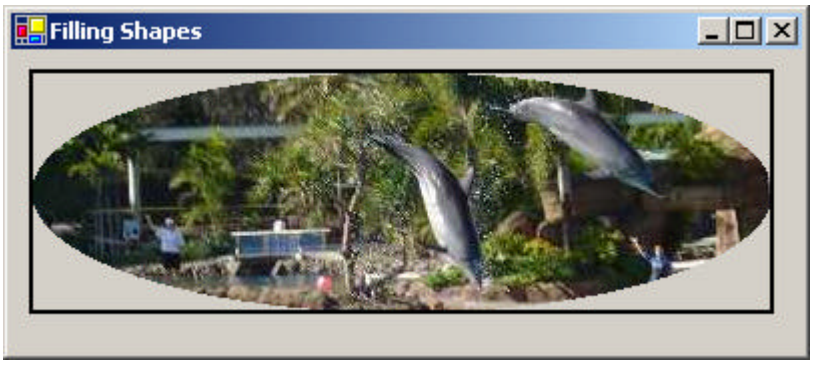


2. Filling a rectangle or an ellipse with the TextureBrush

```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using(Graphics g = e.Graphics)
      {
            Pen pen = new Pen(Color.Black, 2);
            Image image = Image.FromFile("dolphins.jpg");
            TextureBrush brush = new TextureBrush(image);
            Rectangle rect = new Rectangle(10, 10, 270, 120);
            g.FillRectangle(brush, rect);
            g.DrawRectangle(pen, rect);
      }
}
```

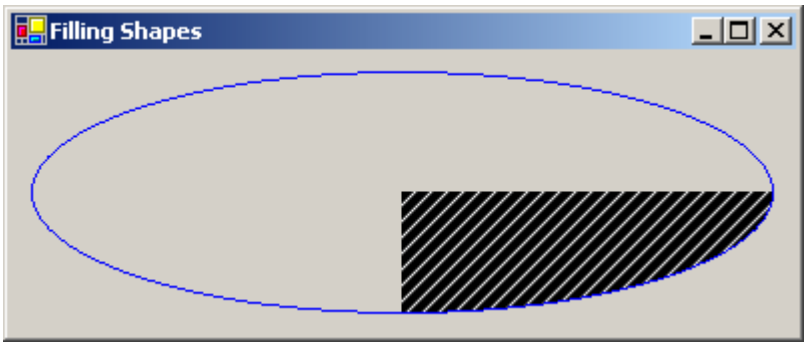



```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using(Graphics g = e.Graphics)
      {
            Pen pen = new Pen(Color.Black, 2);
            Image image = Image.FromFile("dolphins.jpg");
            TextureBrush brush = new TextureBrush(image);
            Rectangle rect = new Rectangle(10, 10, 270, 120);
            g.FillEllipse(brush, rect);
            g.DrawRectangle(pen, rect);
      }
}
```



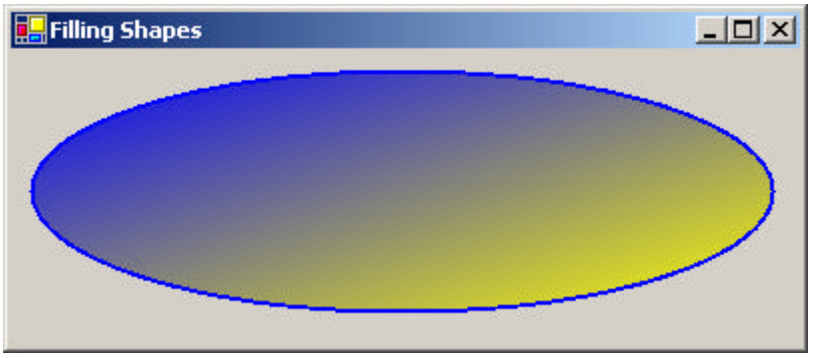
3. Filling a pie with the HatchBrush: note that the `System.Drawing.`**`Drawing2D`** namespace is required

```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using(Graphics g = e.Graphics)
      {
            Pen pen = new Pen(Color.Blue, 1);
            HatchBrush brush = new HatchBrush(
                  HatchStyle.BackwardDiagonal,
                  Color.White, Color.Black);
            Rectangle rect = new Rectangle(10, 10, 370, 120);
            g.FillPie(brush, rect, 0, 90);
            g.DrawEllipse(pen, rect);
      }
}
```



4. Filling a rectangle or an ellipse with the LinearGradientBrush: note that the `System.Drawing.`**`Drawing2D`** namespace is required

```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using(Graphics g = e.Graphics)
      {
            Pen pen = new Pen(Color.Blue, 2);
            Rectangle rect = new Rectangle(10, 10, 370, 120);
            LinearGradientBrush brush = new
                  LinearGradientBrush(
                        rect, Color.Blue,
                        Color.Yellow,
                        LinearGradientMode.ForwardDiagonal);
            g.FillEllipse(brush, rect);
            g.DrawEllipse(pen, rect);
      }
}
```

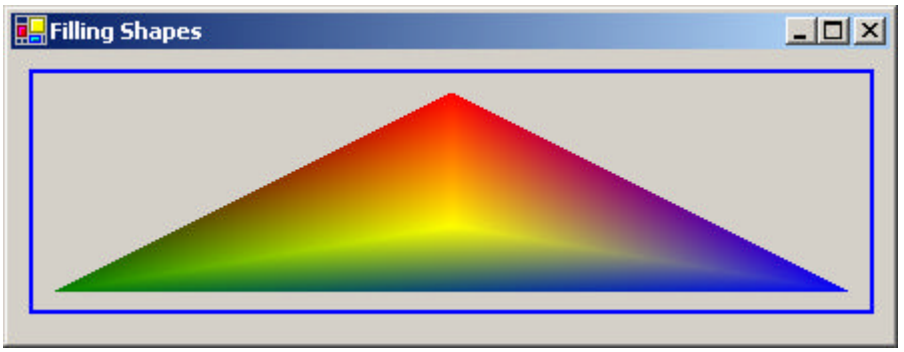



```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using(Graphics g = e.Graphics)
      {
            Pen pen = new Pen(Color.Blue, 2);
            Rectangle rect = new Rectangle(10, 10, 370, 120);
            LinearGradientBrush brush = new
                  LinearGradientBrush(
                        new Point(10,10), new Point(50,50),
                        Color.BlueViolet, Color.AliceBlue);
            g.FillRectangle(brush, rect);
            g.DrawRectangle(pen, rect);
      }
}
```

5. Filling a rectangle or an ellipse with the PathGradientBrush: note that the `System.Drawing.`**`Drawing2D`** namespace is required

The following examples are equivalent

```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using(Graphics g = e.Graphics)
      {
            Pen pen = new Pen(Color.Blue, 2);
            Point [] points = {new Point(220, 20),
                  new Point(20, 120), new Point(420, 120)};
            GraphicsPath path = new GraphicsPath();
            path.AddLines(points);
            PathGradientBrush brush = new
                  PathGradientBrush(path);
            brush.CenterColor = Color.Yellow;
            Color [] surround =
                  {Color.Red, Color.Green, Color.Blue};
            brush.SurroundColors = surround;
            Rectangle rect = new Rectangle(10, 10, 420, 120);
            g.FillRectangle(brush, rect);
            g.DrawRectangle(pen, rect);
      }
}

private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using(Graphics g = e.Graphics)
      {
            Pen pen = new Pen(Color.Blue, 2);
            Point [] points = {new Point(220, 20),
                  new Point(20, 120), new Point(420, 120)};
            PathGradientBrush brush =
                  new PathGradientBrush(points);
            brush.CenterColor = Color.Yellow;
            Color [] surround =
                  {Color.Red, Color.Green, Color.Blue};
            brush.SurroundColors = surround;
            Rectangle rect = new Rectangle(10, 10, 420, 120);
            g.FillRectangle(brush, rect);
            g.DrawRectangle(pen, rect);
      }
}
```

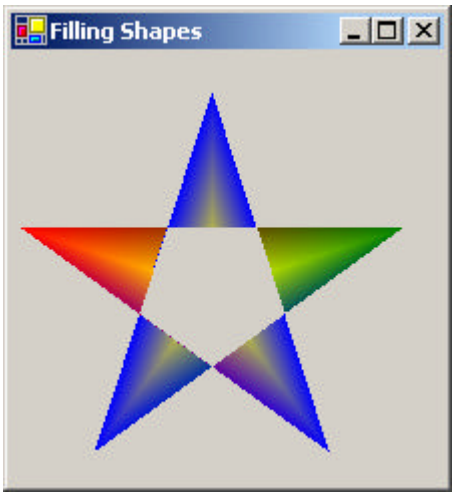


6. Filling a polygon with the PathGradientBrush: note that the `System.Drawing.`**`Drawing2D`** namespace is required

The first example uses **`FillPolygon()`** and the second one uses **`FillPath()`**. The **`FillMode.Alternate`** is used. Both produce the same result

```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using(Graphics g = e.Graphics)
      {
            Pen pen = new Pen(Color.Blue, 2);
            Point [] points = {new Point(5, 88),
                  new Point(195, 88), new Point(41, 200),
                  new Point(100, 19), new Point(159, 200)};
            GraphicsPath path = new GraphicsPath();
            path.AddLines(points);
            PathGradientBrush brush =
                  new PathGradientBrush(path);
            brush.CenterColor = Color.Yellow;
            Color [] surround =
                  {Color.Red, Color.Green, Color.Blue};
            brush.SurroundColors = surround;
            Rectangle rect = new Rectangle(10, 10, 420, 120);
            g.FillPolygon(brush, points, FillMode.Alternate);
      }
}

private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using(Graphics g = e.Graphics)
      {
            Pen pen = new Pen(Color.Blue, 2);
            Point [] points = {new Point(5, 88),
                  new Point(195, 88), new Point(41, 200),
                  new Point(100, 19), new Point(159, 200)};
            GraphicsPath path =
                  new GraphicsPath(FillMode.Alternate);
            path.AddLines(points);
            PathGradientBrush brush =
                  new PathGradientBrush(path);
            brush.CenterColor = Color.Yellow;
            Color [] surround =
                  {Color.Red, Color.Green, Color.Blue};
            brush.SurroundColors = surround;
            Rectangle rect = new Rectangle(10, 10, 420, 120);
```

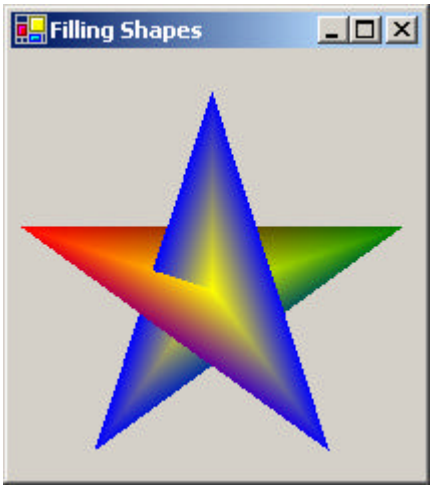
```
            g.FillPath(brush, path);
      }
}
```



The **`FillMode.Winding`** is used below. The first example uses **`FillPolygon()`** and the second one uses **`FillPath()`**. Both produce the same result

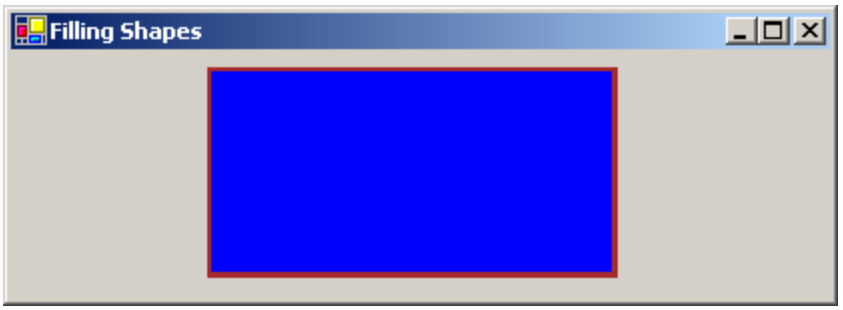
```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using(Graphics g = e.Graphics)
      {
            Pen pen = new Pen(Color.Blue, 2);
            Point [] points = {new Point(5, 88),
                  new Point(195, 88), new Point(41, 200),
                  new Point(100, 19), new Point(159, 200)};
            GraphicsPath path = new GraphicsPath();
            path.AddLines(points);
            PathGradientBrush brush =
                  new PathGradientBrush(path);
            brush.CenterColor = Color.Yellow;
            Color [] surround =
                  {Color.Red, Color.Green, Color.Blue};
            brush.SurroundColors = surround;
            Rectangle rect = new Rectangle(10, 10, 420, 120);
            g.FillPolygon(brush, points, FillMode.Winding);
      }
}

private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using(Graphics g = e.Graphics)
      {
            Pen pen = new Pen(Color.Blue, 2);
            Point [] points = {new Point(5, 88),
                  new Point(195, 88), new Point(41, 200),
                  new Point(100, 19), new Point(159, 200)};
            GraphicsPath path =
                  new GraphicsPath(FillMode.Winding);
            path.AddLines(points);
            PathGradientBrush brush =
                  new PathGradientBrush(path);
            brush.CenterColor = Color.Yellow;
```

```
            Color [] surround =
                  {Color.Red, Color.Green, Color.Blue};
            brush.SurroundColors = surround;
            Rectangle rect = new Rectangle(10, 10, 420, 120);
            g.FillPath(brush, path);
      }
}
```



7. Filling a region: a region describes a set of pixels, a pixel is considered either fully inside, or fully outside the region. Consequently, **FillRegion** does not antialias the edges of the region.

```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using(Graphics g = e.Graphics)
      {
            Pen pen = new Pen(Color.Brown, 5);
            SolidBrush blueBrush = new SolidBrush(Color.Blue);
            Rectangle rect = new Rectangle(100, 10, 200, 100);
            Region region = new Region(rect);
            g.DrawRectangle(pen, rect);
            g.FillRegion(blueBrush, region);
      }
}
```



University of Canberra
Division of Business, Law and Information Sciences
School of Information Sciences and Engineering

Programming Graphical User Interfaces PG

Using Pens

- A **Pen** object is a Microsoft Windows Graphics Device Interface Plus (GDI+) object used to draw lines and curves.
- A **Pen** object has attributes that can be set to adjust the width, colour, and style used to draw lines and curves.

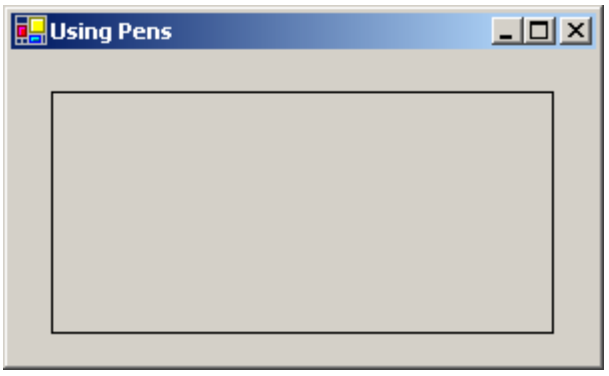
The following examples show how to use pens. Code is added to the **Paint** event handler. The form name is **MainForm**.

Syntax:

```
Pens.selectedColor (with a width of 1)
Pen(System.Drawing.Color color, float width)
Pen(System.Drawing.Brush brush, float width)
```

Example 1: `Pens.selectedColor (with a width of 1)`

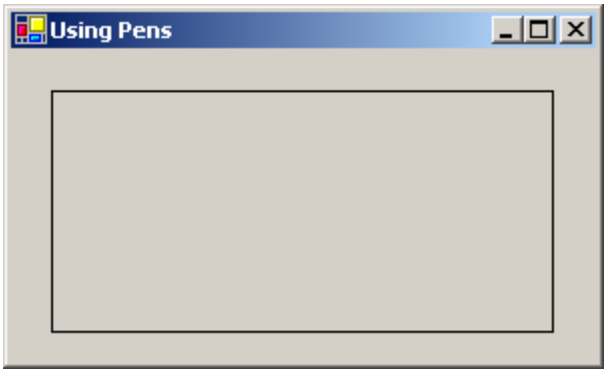
```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {
            Pen myPen = Pens.Black;
            g.DrawRectangle(myPen, 20, 20, 250, 120);
      }
}
```



Example 2: `Pen(System.Drawing.Color color, float width)`

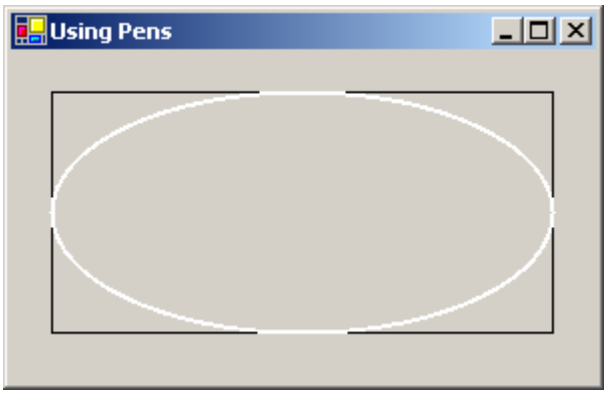
```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {
```

```
            Pen myPen = new Pen(Color.Black, 1);

            g.DrawRectangle(myPen, 20, 20, 250, 120);
        }
}
```



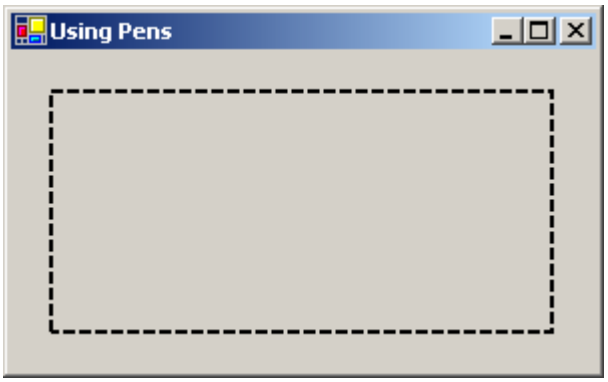
We can reset the **Width** and **Color** attributes after creating the pen

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {

            Pen myPen = new Pen(Color.Black, 1);
            g.DrawRectangle(myPen, 20, 20, 250, 120);

            myPen.Width = 2;
            myPen.Color = Color.White;

            g.DrawEllipse(myPen, 20, 20, 250, 120);
      }
}
```



We can also set the **DashStyle** attribute after creating the pen

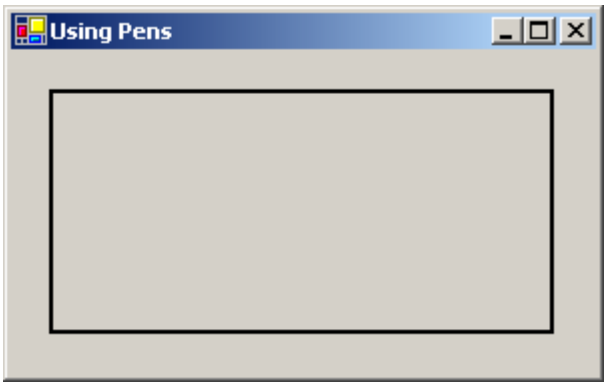
```
      myPen.DashStyle = DashStyle.Dash;
```



Other attributes such as **DashDot**, **DashDotDot** and **Dot** are available.

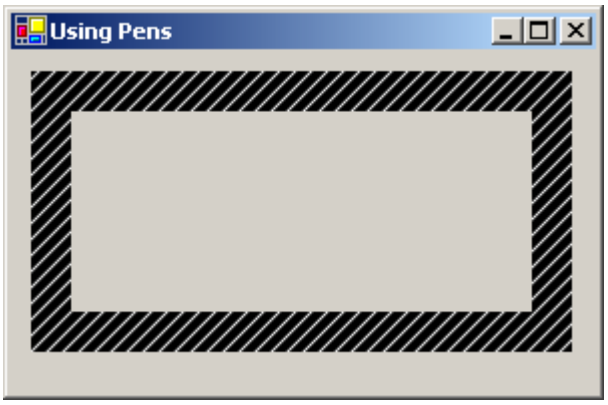
Example 3: **Pen(System.Drawing.Brush *brush*, float *width*)**

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {
            SolidBrush myBrush = new SolidBrush(Color.Black);
            Pen myPen = new Pen(myBrush, 2);

            g.DrawRectangle(myPen, 20, 20, 250, 120);
      }
}
```



Example 4: using **HatchBrush**

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {
            HatchBrush myBrush = new HatchBrush(
                  HatchStyle.BackwardDiagonal,
                  Color.White, Color.Black);
            Pen myPen = new Pen(myBrush, 20);
            g.DrawRectangle(myPen, 20, 20, 250, 120);
      }
}
```

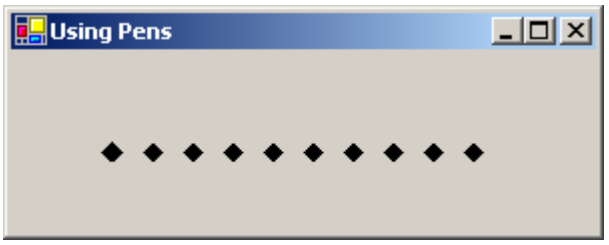


Example 5: using **LineCap**

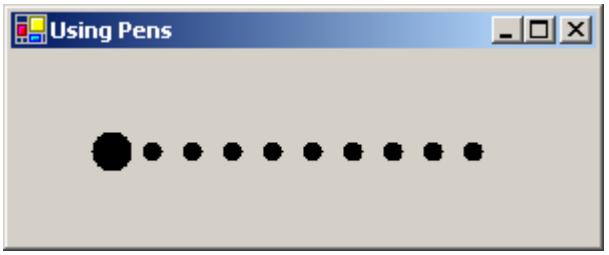
The LineCap enumeration specifies the type of graphic shape to use on the end of a line drawn with a pen. The cap can be a square, circle, triangle, arrowhead, custom, or masked (hidden). End caps can also "anchor" the line by centering the cap at the end of the line

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {
            Pen myPen = new Pen(Color.Black, 10);
            myPen.SetLineCap(LineCap.DiamondAnchor,
                  LineCap.ArrowAnchor, DashCap.Flat);
            Point pt1 = new Point(50, 50);
            Point pt2 = new Point(250, 50);
            g.DrawLine(myPen, pt1, pt2);
      }
}
```


```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {
            Pen myPen = new Pen(Color.Black, 10);
            myPen.StartCap = LineCap.Triangle;
            myPen.EndCap = LineCap.Triangle;
            myPen.DashCap = DashCap.Triangle;
            myPen.DashStyle = DashStyle.Dot;
            Point pt1 = new Point(50, 50);
            Point pt2 = new Point(250, 50);
            g.DrawLine(myPen, pt1, pt2);
      }
}
```




```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {
            Pen myPen = new Pen(Color.Black, 10);
            myPen.DashStyle = DashStyle.Dot;
            myPen.SetLineCap(LineCap.RoundAnchor,
                  LineCap.Square, DashCap.Round);
            Point pt1 = new Point(50, 50);
            Point pt2 = new Point(250, 50);
            g.DrawLine(myPen, pt1, pt2);
      }
}
```



University of Canberra
Division of Business, Law and Information Sciences
School of Information Sciences and Engineering

Programming Graphical User Interfaces PG

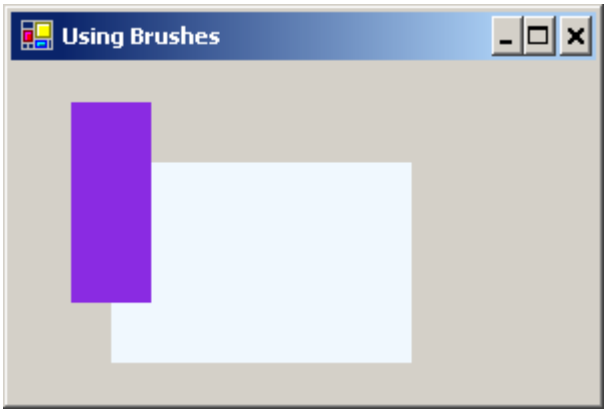
Using Brushes

- A brush is an instance of any class that derives from the **MustInherit** (**abstract**) **Brush** class, and can be used to fill shapes or paint text.
- Brushes are objects that are used with a **Graphics** object to create solid shapes and to render text. There are several different types of brushes: **SolidBrush**, **HatchBrush**, **TextureBrush**, **LinearGradientBrush** and **PathGradientBrush**.

The following examples show how to use brushes. Code is added to the **Paint** event handler. The form name is **MainForm**.

1. **Using system-defined brushes (for standard colours)**: these brushes are included in the Brushes class

```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {
            g.FillRectangle(Brushes.AliceBlue, 50,50,150,100);
            g.FillRectangle(Brushes.BlueViolet, 30,20,40,100);
      }
}
```



2. **SolidBrush**: The SolidBrush class defines a solid colour Brush object used to fill in shapes similar to the way a paint brush can paint the inside of a shape.

Syntax: `SolidBrush(Color color)`

```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
```

```
        {
            SolidBrush myBrush = new SolidBrush(Color.Black);
            g.FillRectangle(myBrush, ClientRectangle);
        }
    }
```

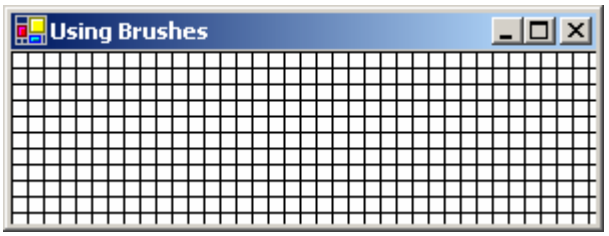
The form is black. The `ClientRectangle` is to get the rectangle that represents the client area of the Windows form

3. **HatchBrush**: the `System.Drawing.`**`Drawing2D`** namespace is required

```
HatchBrush(HatchStyle hatchStyle, Color foreColor, Color backColor)
```

The **HatchStyle** enumeration specifies the hatch pattern consisting of a solid background colour and lines drawn over the background

```
using System.Drawing.Drawing2D;

. . .

private void MainForm_Paint(object sender,
        System.Windows.Forms.PaintEventArgs e)
{
        using (Graphics g = e.Graphics)
        {
            HatchBrush myBrush = new
                HatchBrush(HatchStyle.Cross,
                Color.Black, Color.White);
            g.FillRectangle(myBrush, ClientRectangle);
        }
}
```



`HatchStyle.`**`Cross`**

`HatchStyle.`**`BackwardDiagonal`**

`HatchStyle.`**`ZigZag`**

.

4. **TextureBrush**: The **TextureBrush** class defines a Brush object that contains an Image object that is used to fill the interior of a shape

Syntax: `TextureBrush(Image` *`image`*`)`

```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {
            Image myImage = new
                  Bitmap("summer2004_archery.gif");
            TextureBrush myBrush = new TextureBrush(myImage);
            g.FillRectangle(myBrush, ClientRectangle);
      }
}
```

We can use a wrap mode to specify how a texture or gradient is tiled when it is larger than the area being filled

```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {
            Image myImage = new
                  Bitmap("summer2004_archery.gif");
            TextureBrush myBrush = new TextureBrush(myImage);
            myBrush.WrapMode = WrapMode.TileFlipX;
            g.FillRectangle(myBrush, ClientRectangle);
      }
}
```

5. **LinearGradientBrush**: The LinearGradientBrush class defines a brush that paints a color gradient in which the color changes gradually across the stroke from the starting boundary line to the ending boundary line. The color gradient has one color at the starting boundary line and another color at the ending boundary.

   Syntax:

   ```
   LinearGradientBrush(Rectangle rect, Color color1, Color color2,
   float angle, bool isAngleScalable)
   ```

   or

   ```
   LinearGradientBrush(Rectangle rect, Color color1, Color color2,
   LinearGradientMode mode);
   ```

   or

   ```
   LinearGradientBrush(Point pt1, Point pt2, Color color1, Color
   color2);
   ```

   Examples for those constructor methods are as follows

   Example 1:

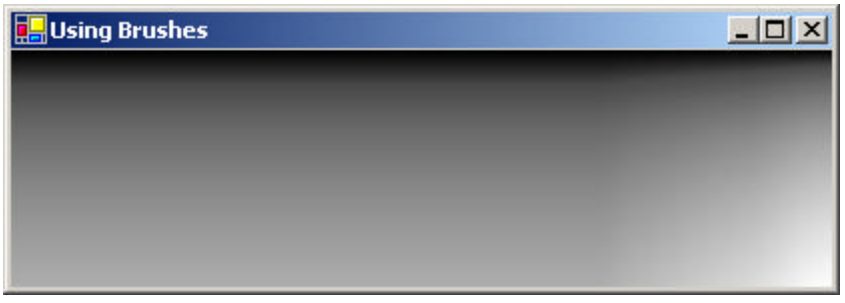
   ```
   private void MainForm_Paint(object sender,
         System.Windows.Forms.PaintEventArgs e)
   {
         using (Graphics g = e.Graphics)
         {
               LinearGradientBrush myBrush = new
                     LinearGradientBrush(
                     ClientRectangle, Color.Black,
                     Color.White, (float) Math.PI/4, true);
               g.FillRectangle(myBrush, ClientRectangle);
         }
   }
   ```

Example 2:

```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {
            LinearGradientBrush myBrush = new
                  LinearGradientBrush(
                  ClientRectangle, Color.Black,
                  Color.White, LinearGradientMode.Vertical);
            g.FillRectangle(myBrush, ClientRectangle);
      }
}
```

If the gamma correction is set to true:

```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {
            LinearGradientBrush myBrush = new
                  LinearGradientBrush(
                  ClientRectangle, Color.Black,
                  Color.White, LinearGradientMode.Vertical);
            myBrush.GammaCorrection = true;
            g.FillRectangle(myBrush, ClientRectangle);
      }
}
```



We can use the **`LinearColors`** method to set the starting and ending colors of the gradient as follows

```
private void MainForm_Paint(object sender,
      System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
```

```
        {
                LinearGradientBrush myBrush = new
                        LinearGradientBrush(
                        ClientRectangle, Color.Black,
                        Color.White, LinearGradientMode.Vertical);
                Color [] linearColors = {Color.White, Color.Gray};
                myBrush.LinearColors = linearColors;
                g.FillRectangle(myBrush, ClientRectangle);
        }
}
```

Example 3:

```
private void MainForm_Paint(object sender,
        System.Windows.Forms.PaintEventArgs e)
{
        using (Graphics g = e.Graphics)
        {
                LinearGradientBrush myBrush = new
                        LinearGradientBrush(
                        new Point(0,0), new Point(100,100),
                        Color.Black, Color.White);
                g.FillRectangle(myBrush, ClientRectangle);
        }
}
```

6. **PathGradientBrush**: A **PathGradientBrush** object stores the attributes of a color gradient that you can use to fill the interior of a path with a gradually changing color.

Syntax: `PathGradientBrush(const GraphicsPath path);`

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
        using (Graphics g = e.Graphics)
        {
```



```
            GraphicsPath path = new GraphicsPath();
            path.AddLine(20, 20, 100, 100);
            path.AddLine(100, 100, 400, 100);
            path.AddLine(400, 100, 20, 20);
            PathGradientBrush myBrush = new
                  PathGradientBrush(path);
            g.FillPath(myBrush, path);
      }
}
```

We can fill colours using CenterColor and SurroundColors as follows

```
private void MainForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
      using (Graphics g = e.Graphics)
      {
            GraphicsPath path = new GraphicsPath();
            path.AddLine(110, 40, 20, 200);
            path.AddLine(20, 200, 200, 200);
            path.AddLine(200, 200, 110, 40);
            PathGradientBrush myBrush = new
                  PathGradientBrush(path);
            myBrush.CenterColor = Color.Aqua;
            Color [] surround = {Color.Red, Color.Green,
                  Color.Blue, Color.Brown};
            myBrush.SurroundColors = surround;
            g.FillPath(myBrush, path);
      }
}
```