آموزش گام به گام برنامه‌نوسی C#

(تحت Form Application)

تالیف:

دکتر رمضان عباس‌نژادورزی
دکتر فاطمه عبدی سقاواز
مهندس محمد نادعلی زاده‌چاری

فن‌آوری نوین

سرشناسه	: عباس‌نژاد ورزی، رمضان، ۱۳۴۸ -
عنوان و نام پدیدآور	: آموزش گام به گام برنامه نویسی C# / (تحت form Application) رمضان عباس‌نژادورزی، فاطمه عبدی سقاواز، محمد نادل علیزاده چاری
مشخصات نشر	: بابل: فناوری نوین، ۱۳۹۷
مشخصات ظاهری	: ۵۴۴ ص.
شابک	: ۵۷۰۰۰۰ ریال: ۳- ۲۱ - ۷۲۷۲-۶۰۰-۹۷۸
وضعیت فهرست نویسی	: فیپا
موضوع	: سی شارپ (زبان برنامه‌نویسی کامپیوتر)
موضوع	: C # (Computer program language)
موضوع	برنامه‌نویسی شی‌گرا
موضوع	Object-oriented programming (Computer science)
شناسه افزوده	عبدی‌سقاواز، فاطمه، ۱۳۵۶ -
شناسه افزوده	نادعلی‌زاده چاری، محمد، ۱۳۶۴ -
رده بندی کنگره	: ۷۶/۷۳ QA / س۱۳۹۷۹۵ ۲ع
رده بندی دیویی	: ۰۰۵/۱۳۳
شماره کتابشناسی ملی	: ۵۲۹۲۰۵۰

فن‌آوری نوین — www.fanavarinovin.net — بابل، کدپستی ۴۷۱۶۷-۷۳۴۴۸ — تلفن: ۰۱۱-۳۲۲۵۶۶۸۷

آموزش گام به گام برنامه نویسی C# (تحت form Apolication)

تألیف: دکتر رمضان عباس‌نژادورزی، دکتر فاطمه عبدی سقاواز، مهندس محمد نادعلیزاده چاری

ناشر: فن‌آوری نوین

چاپ اول: تابستان ۱۳۹۷

تیراژ: ۱۰۰۰ جلد

شابک: ۳ - ۲۱ - ۷۲۷۲ - ۶۰۰ - ۹۷۸

حروفچینی و صفحه‌آرایی: فن‌آوری نوین — **قیمت: ۵۷۰۰۰ تومان**

تهران، خ اردیبهشت، نبش وحید نظری، پلاک ۱۴۲ — **تلفکس: ۶۶۴۰۰۱۴۴-۶۶۴۰۰۲۲۰**

فهرست مطالب

مقدمه

زبان C# (سی شارپ) یکی از محبوب‌ترین زبان‌های برنامه نویسی شی‌گرا در ایران است. این زبان در دانشگاه‌ها نیز به عنوان یکی از مهم‌ترین زبان‌های رشته‌های کامپیوتر، فناوری اطلاعات، ICT علوم کامپیوتر و رشته‌های دیگر مهندسی تدریس می‌شود. علاوه بر این، اکثر برنامه‌های تجاری در ایران با زبان C# پیاده‌سازی‌شده و توسعه می یابند. به همین دلیل، یادگیری این زبان برای دانشجویان و توسعه دهندگان برنامه‌های کامپیوتری یک امر ضروری است. یکی از روش‌های موفق یادگیری هر زبان برنامه‌نویسی، حل مسائل متنوع و آشنایی با الگوریتم‌های مختلف است.

در این کتاب سعی شده است با ارائه مثال‌های ساده، روان، هدف‌دار و متنوع، مفاهیم اساسی برنامه نویسی C# را به صورت گام به گام آموزش دهد. به علاوه کتاب شامل مسائل متنوع و حل آن‌ها می‌باشد که در پروژه‌های بزرگ و پیچیده می‌توانید از آن‌ها استفاده کنید.

در زبان C# مباحثی مختلفی را می‌توان آموزش داد. این کتاب فقط بحث Form Application را آموزش می‌دهد. چنانچه بخواهید Console Application را بی‌آموزید می‌توانید به کتاب‌های سیستم‌های شیء گرا با زبان C#، برنامه‌نویسی پیشرفته با زبان C# و حل ۶۰۰ مسئله C# از همین انتشارات مراجعه کنید. اما، چنانچه بخواهید با بانک اطلاعاتی(تکنولوژی ADO.NET) کار کنید می‌توانید به کتاب آموزش گام به گام بانک اطلاعاتی با زبان C# تالیف رمضان عباس نژادورزی مراجعه نمایید. همچنین اگر نیاز دارید تکنولوژی LINQ را بی‌آموزید، می‌توانید به کتاب آموزش گام به گام LINQ با زبان C# تالیف رمضان عباس نژادورزی مراجعه کنید.

این کتاب شامل یک پیوست می‌باشد که به صورت الکترونیکی در کتاب الکترونیکی ارائه شده است. در این پیوست پروژه‌های متعددی پیاده‌سازی گردیده است. برخی از این پروژه‌ها عبارت‌اند از:

۱. ارسال پست الکترونیک از طریق سرویس دهنده SMTP
۲. ارسال پیامک از طریق مودم GSM و تلفن همراه
۳. پیاده سازی دفترچه تلفن از طریق فایل
۴. نقاشی و حرکت ادمک از طریق کلیدهای جهت نما
۵. پیاده‌سازی تقویم و زمان بندی پروژه‌ها

امیدوارم این کتاب مورد استقبال اساتید و دانشجویان رشته‌های مختلف که زبان برنامه‌نویسی C# را مطالعه می‌کنند، قرار گیرد.

کتاب حاوی برنامه‌های است که کد آنها را می‌توانید به صورت رایگان از سایت انتشارات فناوری نوین به آدرس www.fanavarienovin.net بگیرید.

مولفین
fanavarienovin@gmail.com

فصل ۱ آشنایی با زبان C#

پیشرفت‌های زبان‌های برنامه‌نویسی نظیر C++ و جاوا، موجب ایجاد مشکلات و نیازمندی‌های جدیدی گردید. ایجاد یکپارچگی اجزا نرم‌افزاری از زبان‌های مختلف برنامه‌نویسی مشکل بود و در نصب این ابزار مشکلات مشترکی وجود داشت. به همین دلیل بود که نسخه جدید قطعات[1] با نرم‌افزارهای قدیمی سازگار نبود. از طرف دیگر، نیاز به برنامه‌های تحت وب، موجب گردید تا NET. و زبان برنامه‌نویسی C# ایجاد شود.

C# زبانی است که برنامه نویسان را قادر می‌سازد، به‌راحتی بتوانند از زبان‌های مختلف در پروژه‌شان استفاده کنند. زیرا، C# ریشه در C، C++ و جاوا دارد و ابزارهای زیادی از آن‌ها را در خود جمع کرده، علاوه بر این، ابزارهای جدیدی به آن‌ها اضافه نموده است و قوانین شیءگرایی به‌طور کامل در آن پیاده شده است. در ضمن این زبان باقابلیت برنامه‌نویسی ویژوال، امکان ایجاد برنامه‌هایی با استفاده از **محیط توسعه مجتمع** (IDE)[2] را تأمین کرده است. با استفاده از IDE، برنامه‌نویس می‌تواند به‌راحتی برنامه را ایجاد، اجرا، آزمایش (تست) و رفع اشکال (خطایابی) نماید. پس، در زمان برنامه‌نویسی صرفه‌جویی زیادی خواهد شد. روند ایجاد سریع برنامه‌ها با استفاده از IDE، به **توسعه سریع برنامه** (RAD)[3] معروف است.

مزیت دیگر C# استفاده از قطعات تولیدشده در زبان‌های برنامه‌نویسی مختلف در آن است.

۱ – ۱. فرآیند برنامه‌نویسی در دات‌نت

در طراحی یک برنامه، اولین گام تعیین نوع برنامه‌ای است که می‌خواهید آن را ایجاد کنید. در دات‌نت برنامه‌های متعددی از قبیل برنامه تحت کنسول، برنامه‌های تحت ویندوز، برنامه‌های تحت وب، وب‌سرویس یا انواع دیگری را می‌توان ایجاد کرد. در این کتاب روش ایجاد برنامه‌های تحت کنسول و تحت ویندوز را می‌آموزیم. گام بعدی انتخاب زبان برنامه‌نویسی است. این مرحله از اهمیت ویژه‌ای برخوردار است. چون، زبان‌های غیردات‌نت امکانات مختلفی را در اختیارتان می‌گذارند. اما، در دات‌نت زبان‌های مختلف به یکدیگر شبیه شده‌اند و امکانات یکسانی را در اختیارتان قرار می‌دهند. چون این زبان‌ها در هنگام اجرا از زبان مشترک زمان اجرا (CLR)[4] استفاده می کنند. بنابراین، در زمان اجرا این مورد که در طراحی برنامه از چه زبانی استفاده‌شده است، تفاوتی ندارد. ازآنجایی‌که زبان‌های مختلف گرامرهای متفاوتی دارند، بنابراین باید زبانی

---

1 .Components
2. IDE(Integrated Development Environment)
3.RAD(Rapid Application Development)
4. Common Language Runtime

انتخاب شود تا با گرامر آن آشنا باشید. چون، گرامر زبان #C شبیه زبان C و ++C است به همین دلیل، زبان برنامه‌نویسی و طراحی را #C انتخاب نمودیم.

ویژوال استودیودات‌نت از کامپایلرها و زبان‌های مختلفی تشکیل‌شده است که عبارت‌اند از:

۱. ویژوال‌بیسیک ۲. ویژوال #C

۳. ویژوال ++C ۴. ویژوال #F

علاوه بر مایکروسافت شرکت‌های دیگر نیز برای زبان‌های خود کامپایلرهایی را عرضه کرده‌اند که CLR را به‌عنوان محیط زمان اجرای نهایی مورداستفاده قرار داده‌اند. برخی از این زبان‌ها عبارت انداز: APL، Cobol، Fortran، ML، Mercury، Perl، Python، RPG، Smalltalk و غیره.

گام سوم، نوشتن کد موردنیاز برنامه است. البته برنامه‌های مختلف کدهای متعددی خواهند داشت. در ادامه با انواع برنامه و کدهای پیاده‌سازی آن‌ها آشنا خواهید شد. گام چهارم، کامپایل نمودن برنامه می‌باشد. کامپایل موجب می‌شود تا خطاهای برنامه (نحوی و گرامری) رفع شده، برنامه به کد میانی CLR ترجمه شود.

۲ – ۱. مجموعه کتابخانه کلاس دات‌نت Framework

علاوه بر CLR، در مجموعه دات‌نت Framework، بخش دیگری به نام کتابخانه کلاس چارچوب (FCL)[1] وجود دارد. این بخش شامل هزاران کلاس می‌باشد که هرکدام وظیفه خاصی دارند. مجموعه‌های FCL و CLR به طراحان اجازه می‌دهند که چندین مدل برنامه را طراحی کنند که عبارت‌اند از:

۱. برنامه‌های تحت کنسول در ویندوز، برنامه‌هایی ایجاد می‌کند که نیاز به رابط گرافیکی کاربر ندارند. این برنامه‌ها از رابط خط فرمان استفاده می‌کنند. این نوع برنامه‌ها معمولاً برای نوشتن ابزارهایی نظیر کامپایلر و بعضی از برنامه‌های کاربردی به کار می‌روند (در فصل‌های ۱ تا ۶ این نوع برنامه‌ها را می‌آموزیم).

۲. برنامه‌های تحت ویندوز، برنامه‌هایی هستند که نیاز به رابط گرافیکی کاربر دارند. برنامه‌هایی تحت ویندوز برنامه‌های دسک‌تاپ نیز نامیده می‌شوند. زمانی که به برنامه‌های تحت وب نیاز نباشد، می‌توان از این برنامه‌ها استفاده نمود (در فصل‌های ۷ تا ۹ روش ایجاد برنامه‌های تحت ویندوز را می‌آموزیم).

۳. برنامه‌های تحت وب، برنامه‌هایی ایجاد می‌کنند که مبتنی بر صفحات HTML[2] هستند. این نوع برنامه‌ها، از طریق سرویس‌دهنده بانک اطلاعاتی یا چندین وب‌سرویس، اطلاعات موردنیازشان را دریافت کرده، پردازش‌های موردنیاز را بر روی آن انجام داده، صفحات مبتنی بر HTML ایجاد می‌نمایند تا این صفحات از طریق مرورگرهای وب[3] در کامپیوتر سرویس‌گیرنده[4] قابل‌نمایش باشند.

---

1. Framework Class Library 2. Hyper Text Markup Language 3. Browser 4. Client

۴. سرویس‌های ویندوز، سرویس‌هایی را می‌توان در دات‌نت ایجاد کرد که توسط مدیر کنترل سرویس ویندوز (SCM)[1] و نیز دات‌نت Framework قابل‌کنترل هستند. معمولاً این سرویس‌ها برای تبادل اطلاعات بین برنامه‌های مختلف استفاده می‌شوند.

۵. وب‌سرویس‌ها، سرویس‌ها یا توابعی هستند که به‌راحتی از طریق شبکه وب قابل‌دسترسی و فراخوانی هستند.

۶. قطعات و کتابخانه کلاس، در دات‌نت Framework می‌توان قطعات[2] و کتابخانه‌هایی با کلاس‌های جدید ایجاد نمود. این قطعات و کتابخانه‌های کلاس‌های جدید را به‌راحتی می‌توان در برنامه‌های دیگر (حتی به زبان‌های دیگر) استفاده نمود.

۷. و غیره

۳ – ۱. فضای نام

همان‌طور که بیان گردید، در FCL هزاران کلاس وجود دارند. برای دسته‌بندی کلاس‌ها، تمام کلاس‌های مرتبط به هم در یک فضای نام[3] قرار می‌گیرند. اصلی‌ترین فضای نام، فضای نام System است. این فضای نام، شامل کلاس object و تعدادی کلاس پایه دیگر است. کلاس object یک کلاس پایه است که تمام کلاس‌های FCL از این کلاس مشتق می‌شوند (در فصل ۶ با مفهوم کلاس‌های پایه و مشتق آشنا خواهید شد). کلاس‌هایی که در FCL وجود دارند، **کلاس‌های آماده** نام دارند. علاوه بر این کلاس‌ها، برنامه‌نویس می‌تواند کلاس‌های جدیدی را ایجاد کرده و از آن‌ها استفاده کند. چون، ممکن است کلاس موجود در FCL همه نیازهای برنامه‌نویس را برطرف نکند. چگونگی ایجاد این کلاس‌ها را در فصل ۵ و ۶ می‌آموزیم. در این بخش می‌خواهیم به کلاس‌های موجود در FCL بپردازیم. کلاس‌های موجود در FCL با توجه به کاربردشان در فضاهای نام مختلف قرار می‌گیرند. این عمل دو مزیت زیر را برای برنامه‌نویس در پی دارد:

۱. موجب دسته‌بندی کلاس‌ها می‌شود. یعنی کلاس‌هایی که به هم مرتبط هستند، در یک فضای نام قرار می‌گیرند تا اولاً بتوان به‌راحتی آن‌ها را به پروژه اضافه نمود و ثانیاً فضاهای نامی که در پروژه به آن‌ها نیازی نیست، به پروژه اضافه نگردند.

۲. علاوه بر این می‌توان در کلاس‌های مختلف از نام‌های تکراری استفاده نمود. فضای نام و کلاس‌ها موجب می‌شوند تا نام‌های تکراری از یکدیگر تفکیک شوند.

وقتی برنامه جدیدی ایجاد می‌کنید فضاهای نام جدول ۱ – ۱ به پروژه اضافه می‌شود (البته وقتی‌که برنامه‌ای از نوع Console Application ایجاد می‌نمایید. اگر برنامه‌هایی با نوع‌های دیگر به برنامه اضافه کنید، ممکن است فضاهای نام دیگر به پروژه‌تان اضافه شود). علاوه بر فضاهای نامی که به‌طور خودکار به برنامه

---

[1] .Windows Service Control Manager [2] .Components [3].Namespace

اضافه می‌شوند، می‌توانید فضاهای نام دیگر را نیز به پروژه‌تان اضافه کنید. برای این منظور می‌توانید از دستور using به‌صورت زیر استفاده نمایید:

```
using نام فضای نام;
```

به‌عنوان‌مثال، دستورات زیر را ببینید:

```
using System.Convert;
using System.Io;
```

دستور اول، فضای نام System.Convert را به برنامه اضافه می‌کند تا بتوانید از متدهایی که برای تبدیل انواع داده‌های مختلف به کار می‌روند، استفاده نمایید (این متدها را در ادامه می‌آموزیم) و دستور دوم، فضای نام System.Io را به پروژه اضافه می‌کند تا بتوانید از کلاس‌هایی که جهت ورودی – خروجی داده‌ها مانند فایل‌ها به کار می‌روند، استفاده نمایید.

چنانچه در ابتدای برنامه با دستور using فضای نام را اضافه نکنید در کلیه مکان‌هایی که می‌خواهید از کلاس‌های موجود در آن فضای نام استفاده نمایید باید مسیر کامل فضای نام را ذکر کنید (به‌صورت زیر):

```
System.Convert.ToInt32
```

این دستور از متد ToInt32() کلاس Convert موجود در فضای نام System استفاده می‌کند.

هر پروژه جدیدی که ایجاد می‌شود، یک فضای نام جدید همنام با پروژه نیز ایجاد می‌گردد.

۴ – ۱. آموزش زبان‌های برنامه‌نویسی

آموزش زبان‌های برنامه‌نویسی مانند زبان‌های طبیعی زنده دنیا است. یعنی برای آموزش زبان‌های برنامه‌نویسی باید مراحل زیر را انجام داد:

۱. مانند هر زبان طبیعی ابتدا باید علائم تشکیل‌دهنده زبان را آموخت. به‌عنوان‌مثال، زبان فارسی از علائم الف تا ی، ارقام ۰ تا ۹ و علائم خاص مانند ! ، : ، ؟ و غیره تشکیل‌شده است. هرکدام از این علائم (نمادها) مفهوم خاصی را دارند. زبان #C، نیز از علائم a تا z ، A تا Z ، ۰ تا ۹ ، علائم ویژه نظیر ; ، : ،] ، [، / و غیره تشکیل‌شده است. ابتدا باید مفاهیم هریک از این علائم را در زبان #C آموخت.

۲. همان‌طور که می‌دانید از ترکیب علائم هر زبان کلمات به وجود می‌آیند. برخی از کلمات دارای معنی و مفهوم هستند و برخی دیگر معنی و مفهوم خاصی ندارند. به‌عنوان‌مثال، کلمات **بابا، آب، داد،** در زبان فارسی مفهوم خاصی دارند. ولی کلمات **تپتانم و بکیاپ** مفهوم خاصی ندارند. به کلماتی که در زبان دارای مفهوم خاص هستند، **کلمات کلیدی** می‌گویند. در زبان #C کلمات کلیدی نظیر for، if، else، while و int وجود دارند. در آموزش این زبان ابتدا باید کلمات کلیدی را شناخت. معنی و کاربرد هرکدام از آن‌ها را باید آموخت.

جدول ۱ – ۱ برخی فضاهای نام.	
فضای نام	هدف
System	شامل کلاس‌های پایه دات‌نت و انواع داده از قبیل double، int، char و غیره است.
System.Collection.Generic	شامل کلاس‌هایی اصلی کلکسیون در دات‌نت می‌باشد.
System.LINQ	از کلاس‌هایی تشکیل‌شده است که برای کار با LINQ [۱] به کار می‌روند.
System.Text	از کلاس‌هایی تشکیل‌شده است که برای کار کردن بر روی متن از قبیل رمزگذاری، رمزگشایی، کلاس StringBuilder و غیره به کار می‌روند.

۳. در هر زبان طبیعی از ترکیب کلمات کلیدی با یک قواعد خاص، جمله ایجاد می‌شود (مانند بابا آب داد). همان‌طور که می‌دانید در زبان فارسی ابتدا فاعل، سپس مفعول و در پایان فعل قرار می‌گیرد. در زبان C# نیز برای ایجاد جملات (دستورات) قواعد خاصی وجود دارد. به‌عنوان‌مثال، int برای تعریف داده‌های نوع صحیح به کار می‌رود و به‌صورت زیر استفاده می‌گردد:

int متغیر۱ , متغیر۲ , ... , متغیرn ;

۴. همان‌طور که می‌دانید، در زبان‌های طبیعی از کنار هم قرار گرفتن جملات مرتبط به هم پاراگراف ایجاد می‌شود. در زبان‌های برنامه‌نویسی نیز با کنار هم قرار دادن دستورات مرتبط به هم، بلاک ایجاد می‌شود. در زبان C#، هر بلاک با { شروع و با } خاتمه می‌یابد.

۵. چند پاراگراف صفحات و فصول را ایجاد خواهند کرد و این روند ادامه می‌یابد تا یک کتاب نوشته شود. در زبان‌های برنامه‌سازی نیز نوشتن برنامه‌ها هم همین روند را دارد. تعدادی بلاک، فایل، و چند فایل مرتبط به هم، برنامه را ایجاد می‌کند.

در ادامه کتاب به آموزش زبان C# با این شیوه می‌پردازیم.

۵ – ۱. شناسه‌ها

شناسه‌ها[۲]، نام‌هایی هستند که برنامه‌نویس به عناصر C# از قبیل کلاس‌ها[۳]، فضاهای نام[۴]، متدها[۵]، فیلدها[۶]، خواص[۷] و غیره انتخاب می‌کند. به‌عنوان‌مثال، (شکل ۱ – ۱) را مشاهده کنید. در این شکل هر کلمه‌ای که در داخل مستطیل قرار دارد، شناسه است. قبل از استفاده از شناسه‌ها باید آن‌ها را نام‌گذاری نمود. قوانین نام‌گذاری شناسه‌ها در زیر آمده‌اند (شکل ۲ – ۱). شناسه‌ها در C# با رنگ سبز مشخص می‌گردند.

- کاراکترهای الفبایی (A تا Z، a تا z) و خط ربط (_) می‌توانند هر مکان نام شناسه قرار گیرند.

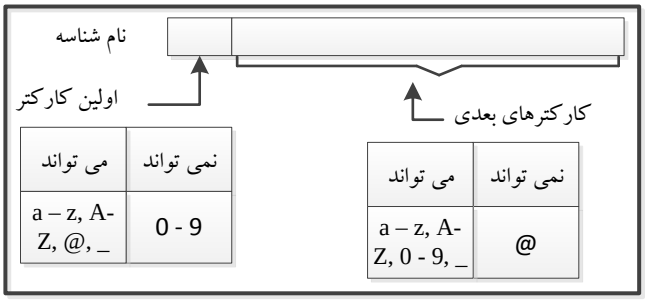
---

[۱]. برای کسب اطلاعات بیشتر درزمینهٔ LINQ به کتاب آموزش گام‌به‌گام LINQ تألیف رمضان عباس نژاد ورزی انتشارات فناوری نوین مراجعه فرمایید.

[2].Identifiers [3].Classes [4].Namespaces [5].Methods [6]. Fields [7].Properties

```
using System;

namespace ProjectNamespace {

    class Program {

        static void Main(string[] args) {

            // perform a calculation
            int x = 10 * 10;

            // print out the result of the calculation
            Console.WriteLine("Result: {0}", x);
        }
    }
}
```

شکل ۱ – ۱ برخی از شناسه‌های در یک برنامه.



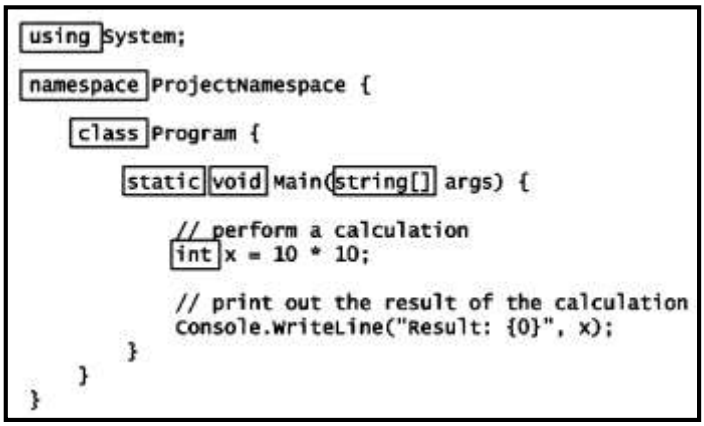
شکل ۲ – ۱ روش نام‌گذاری شناسه‌ها.

- ارقام صفر تا ۹ نمی‌توانند در اولین مکان نام شناسه قرار گیرند، ولی می‌توانند در مکان‌های دیگر نام شناسه مورداستفاده قرار گیرند.
- کاراکتر @ می‌تواند در اولین مکان نام شناسه قرار بگیرد، اما، نمی‌تواند در مکان‌های دیگر نام شناسه قرار گیرد.
- نام شناسه نسبت به حروف بزرگ و کوچک حساس است. یعنی، شناسه‌های myVar و MyVar دو نام مختلف برای دو شناسه در نظر گرفته می‌شوند.

۶ – ۱. کلمات کلیدی

کلماتی که در زبان شناخته‌شده‌اند و مفهوم خاصی در آن زبان‌دارند، **کلمات کلیدی**[1] نامیده می‌شوند. برخی از کلمات کلیدی را در (شکل ۳ – ۱) می‌بینید. این کلمات در داخل مستطیل قرار دارند. C# از کلمات کلیدی زیادی تشکیل می‌شود که برخی از آن‌ها را در جدول ۲ – ۱ می‌بینید (کلمات کلیدی در برنامه C# با رنگ آبی مشخص می‌شوند).

[1].Keywords



```
using System;

namespace ProjectNamespace {

    class Program {

        static void Main(string[] args) {

            // perform a calculation
            int x = 10 * 10;

            // print out the result of the calculation
            Console.WriteLine("Result: {0}", x);
        }
    }
}
```

شکل ۳ – ۱ برخی از کلمات کلیدی در برنامه #C.

جدول ۲ – ۱ کلمات کلیدی #C.

abstract	const	extern	out	Int	short	typeof
as	continue	false	override	interface	sizeof	uint
base	decimal	finally	params	internal	stackalloc	ulong
bool	default	fixed	private	is	static	uchecked
break	delegate	float	protected	lock	string	unsafe
byte	do	for	public	long	struct	unshort
case	double	foreach	readonly	namespace	switch	using
catch	else	goto	ref	new	this	virtual
char	enum	if	return	null	throw	void
checked	event	implicit	sbyte	object	true	volatile
class	explicit	in	sealed	operator	try	while
کلمات کلیدی مختص زبان #C						
ascending	by	descending	equals	From	get	Group
into	join	let	on	Orderby	partial	Select
set	value	where	yield			

۷ – ۱. فضای سفید

فضای سفید (whitespace)، کاراکترهایی هستند که قابلیت چاپ ندارند. این کاراکترها توسط کامپایلر نادیده گرفته می‌شوند، اما برنامه‌نویس برای افزایش خوانایی برنامه از این کاراکترها در برنامه‌اش استفاده می‌کند. برخی از این کاراکترها عبارت‌اند از : ۱. کاراکتر فضای خالی (space) ۲. کاراکتر Tab ۳. خط جدید (New Line) و ۴. کلید Enter(carriage return).

۸ – ۱. لیترال‌ها

لیترال‌ها[1]، داده‌هایی هستند که به‌صورت ثابت در کد برنامه‌تان وارد می‌کنید. لیترال‌ها می‌توانند مقادیر عددی، رشته‌ای (که در بین جفت کتیشن قرار می‌گیرند) یا منطقی (True یا False) باشند. در (شکل ۴ – ۱) برخی از لیترال‌ها را می‌بینید. در این شکل لیترال‌ها در داخل مستطیل قرار دارند.

[1].Literals

```
using System;

namespace ProjectNamespace {

    class Program {

        static void Main(string[] args) {

            // perform a calculation
            int x = 10 * 10

            // print out the result of the calculation
            Console.WriteLine("Result: {0}", x);
        }
    }
}
```

شکل ۴ – ۱ برخی از لیترال‌ها در C#.

۹ – ۱. توضیحات

توضیحات[1] توسط کامپایلر نادیده گرفته می‌شوند و موجب افزایش خوانایی برنامه می‌گردند(شکل ۵ – ۱). در این شکل توضیحات در داخل مستطیل قرار دارند. توضیحات در برنامه با رنگ سبز پررنگ نمایش داده می‌شوند. به دو روش می‌توان توضیحات را به برنامه اضافه کرد:

```
using System;

namespace ProjectNamespace {

    class Program {

        static void Main(string[] args) {

            // perform a calculation
            int x = 10 * 10;

            // print out the result of the calculation
            Console.WriteLine("Result: {0}", x);
        }
    }
}
```

شکل ۵ – ۱ موجب افزایش خوانایی برنامه.

۱. **کاراکترهای //**، تمام کلمات بعد از // **توسط** کامپایلر نادیده گرفته می‌شوند. یعنی، این کلمات توضیحات در نظر گرفته می‌شوند. به‌عنوان‌مثال، دستور زیر را در نظر بگیرید:

```
int i = 0;    //Define   I  and initial .
```

این دستور i را تعریف کرده، مقدار صفر را در آن قرار می‌دهد و در ادامه دستور، توضیح برای آن آمده است.

---

[1].Comments

۲. کاراکترهای *\ و /*، توضیحات می‌توانند با کاراکترهای *\ شروع می‌شوند و با کاراکترهای /* خاتمه یابند. یعنی، تمام کلماتی که بین *\ و /* قرار می‌گیرند، توضیحات در نظر گرفته می‌شوند. با این روش می‌توان توضیحات چند سطری(خطی) نیز ایجاد نمود. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
/*
This text is ignored by the compiler.
Unlike single-line comments, delimited comments
like this one can span multiple lines.
*/
```

این دستورات توضیحات چند سطری را ایجاد می‌کنند.

در C# نمی‌توانید توضیحات تودرتو تعریف کنید. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
/*This is an attempt at a nested comment.
? /*Ignored because it's inside a commentInner comment
? */Closes the comment because it's the first end delimiter
      encountered
? */Syntax error because it has no opening delimiter*/
```

این دستورات موجب تولید خطا توسط کامپایلر خواهند شد. چون در C# نمی‌توان توضیحات تودرتو ایجاد کرد. اکنون دستورات زیر را ببینید:

```
//Single-line comment      /* Nested comment?
? /*Incorrect because it has no opening delimiter*/
```

این دستورات نیز موجب تولید خطا توسط کامپایلر خواهند شد.

۱۰ – ۱. کاراکترهای ویژه

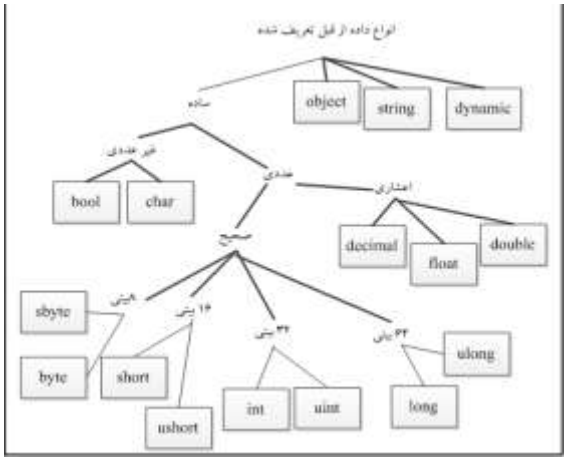
این کاراکترها برای نگهداری گروهی از اجزا یا جداکننده‌ها به کار می‌روند. برخی از این کاراکترها را در (شکل ۶ – ۱) می‌بینید. در این شکل، کاراکتر ; انتهای دستورات C# را مشخص می‌کند. کاراکتر }، بلاک C# را باز می‌کند و کاراکتر {، بلاک بازشده C# را می‌بندد. هر بلاک در C# با کاراکتر } شروع و با کاراکتر { خاتمه می‌یابد. در C# کاراکترهای } و { می‌توانند به‌صورت تودرتو به کار روند(شکل ۵ – ۱ را ببینید).

```
using System;

namespace ProjectNamespace {

    class Program {

        static void Main(string[] args) {

            // perform a calculation
            int x = 10 * 10;

            // print out the result of the calculation
            Console.WriteLine("Result: {0}", x);
        }
    }
}
```

شکل ۶ – ۱ نمایش کاراکترهای خاص.

۱۱ – ۱. انواع داده

در #C، دو نوع داده مقدار و ارجاع وجود دارند. انواع داده‌های #C را در (شکل ۷ – ۱) می‌بینید. خلاصه این داده‌ها در جدول ۳ – ۱ آمده‌اند.



شکل۷ – ۱ انواع داده‌های #C.

۱۲ – ۱. انواع مقدار

متغیر نوع مقدار (value-type)، به‌طور مستقیم داده‌اش را نگه‌داری می‌کند. یعنی، محتوی متغیر نوع مقدار، مقدارش است. به‌عنوان‌مثال، دستور زیر مقدار ۲۵ را به متغیری به نام i نسبت می‌دهد:

```
int   i = 25;
```

این عمل به شکل مقابل انجام می‌شود:

نوع	نام	مقدار
Int	i	25

وقتی متغیر نوع مقدار ذخیره می‌شود، #C نوشته‌ای از نوع، نام شناسه و مقدار را نگهداری می‌کند و هنگامی‌که متغیر نوع مقدار را کپی می‌کنید، متغیر دوم جداگانه‌ای با همان نوع و مقدار ایجاد می‌شود. به‌عنوان‌مثال، دستور زیر کپی i را در j نشان می‌دهد:

```
int   j = i;
```

این عمل به شکل مقابل انجام می‌گردد:

نوع	نام	مقدار
int	i	25
int	j	25

این متغیرها ارتباطی باهم ندارند. اگر مقدار متغیری عوض شود، متغیر دیگر مقدار قبلی‌اش را نگهداری می‌کند. به‌عنوان‌مثال، دستور مقابل را ببینید:

```
i = 50;
```

این دستور، مقدار i را به ۵۰ تغییر می‌دهد، اما، مقدار j همان ۲۵ باقی خواهد ماند (شکل مقابل را مشاهده کنید):

نوع	نام	مقدار
int	i	50
int	j	25

تمام انواع تعریف‌شده در جدول۴ – ۱ (به‌جز نوع object و string) متغیرهای از نوع مقدار را تعریف می‌کنند.

جدول ۳ - ۱ انواع داده‌های C#

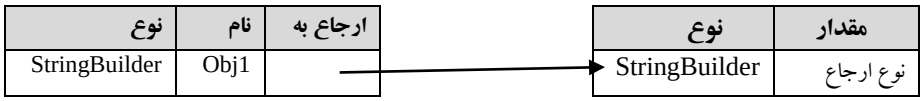
نام	هدف	محدوده	معادل .NET	مقدار پیش‌فرض
sbyte	عدد صحیح ۸ بیتی با علامت	-128….127	System.SByte	**0**
byte	عدد صحیح ۸ بیتی بدون علامت	….2550	System.Byte	**0**
short	عدد صحیح ۱۶ بیتی با علامت	-32768…32767	System.Int16	**0**
ushort	عدد صحیح ۱۶ بیتی بدون علامت	0….65535	System.UInt16	**0**
Int	عدد صحیح ۳۲ بیتی با علامت	-2147483648… 2147483647	System.Int32	**0**
Uint	عدد صحیح ۳۲بیتی بدون علامت	0….4294964295	System.UInt32	**0**
Long	عدد صحیح ۶۴ بیتی با علامت	-922337203685477 5808… 922337203685477 807	System.Int64	**0**
Ulong	عدد صحیح ۶۴ بیتی بدون علامت	0… 18446744073 709551615	System.UInt64	**0**
Float	عدد اعشاری با دقت معمولی	105×10^{-45} ... 3.4×10^{38}	System.Single	0.0f
Double	عدد اعشاری با دقت مضاعف	5×10^{-324} ... 1.7×10^{-308}	System.Double	0.0d
Bool	نوع منطقی	True, False	System.Boolean	False
Char	کاراکتر یونیکد،	U+0000 …. U+ffff	System.Char	\x0000
Decimal	عدد ده‌دهی با ۲۵ رقم اعشار	$\pm 1.0\times 10^{28}$ ... $\pm 7.9\times 10^{28}$	System.Decimal	0m
Object	کلاس پایه‌ای که همه انواع دیگر از آن مشتق می‌شوند	-----	System.object	Null
String	مجموعه‌ای از کاراکترهای یونیکد	------	System.string	Null

۱۳ – ۱. انواع ارجاع

انواع ارجاع (Reference Types) به دو شکل یک شیء و ارجاع به آن شیء می‌باشند. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
StringBuilder  obj1 = new StringBuilder ("نوع ارجاع");
```

نوع StringBuilder برای تعریف رشته‌ای از کاراکترها به کار می‌رود. این دستور به شکل زیر عمل می‌کند:

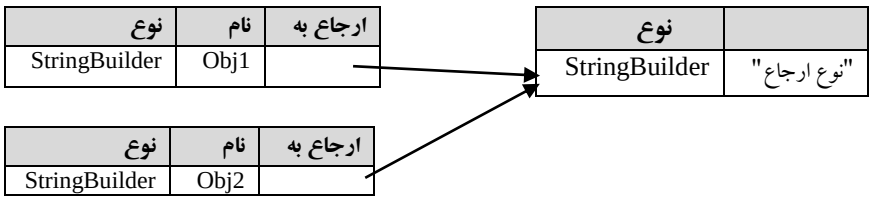


همان‌طور که در این شکل می‌بینید، نمی‌توان به‌طور مستقیم به شیء و مقدار آن دستیابی داشت. برای انجام این کار باید از طریق ارجاع استفاده نمایید. در این شکل متغیر ارجاع و شیء را مشاهده می‌کنید. ارجاع، شامل نوع متغیر، نام آن و پیوند که آن را به شیء ارجاع می‌دهد و شیء شامل مقدار است.

وقتی‌که متغیری از نوع ارجاع را کپی می‌کنید، کپی جدیدی از ارجاع ایجاد خواهد شد، اما، یک شیء جدید نیست. به‌عنوان‌مثال، دستور زیر را مشاهده کنید:

```
StringBuilder obj2 = obj1;
```

این دستور، بیان می‌کند که obj2 و obj1 هر دو شی به مقدار "نوع ارجاع" اشاره می‌کنند(شکل زیر را ببینید):



همان‌طور که در این شکل می‌بینید، فقط یک شیء از نوع StringBuilder داریم، اما دو ارجاع به نام‌های obj2 و obj1 داریم که به آن اشاره می‌کنند و برخلاف نوع مقدار، مقدار هر دو ارجاع یکی است(یعنی، مقدار "نوع ارجاع" را دارند). با تغییر مقدار یکی، مقدار دیگری نیز تغییر خواهد کرد. C# دو نوع ارجاع اولیه به نام‌های String و object دارد.

دو ارجاع به یک شیء می‌توانند انواع مختلف داشته باشند.

متغیرهای نوع ارجاع را می‌توان طوری تعریف کرد که به‌جایی اشاره نکنند. به‌عنوان‌مثال، دستور زیر را در نظر بگیرید:

```
StringBuilder    obj = null;
```

نوع	نام	ارجاع به
StringBuilder	obj	Null

انواع تهی پذیر

انواع تهی پذیر[1]، اجازه می‌دهند تا متغیری از نوع مقدار ایجاد کنید که می‌تواند یک داده معتبر و غیر معتبر را بپذیرد. بنابراین می‌توانید قبل از استفاده از متغیر به معتبر بودن داده آن مطمئن شوید. یک نوع تهی پذیر همیشه بر پایه نوع دیگر است. ایجاد نوع تهی پذیر به‌صورت زیر است:

; نام متغیر ؟ نوع پایه

به‌عنوان‌مثال، دستور زیر را در نظر بگیرید:

```
int?    i = 20;
```

این دستور متغیر i را از نوع تهی پذیر تعریف کرده مقدار ۲۰ را در آن قرار می‌دهد.

اکنون با عملگر =! می‌توان مقدار i را با تهی مقایسه کرد. دستور زیر را در نظر بگیرید:

```
bool      b = i != null;
```

این دستور b را از نوع bool تعریف می‌کند. اگر i برابر تهی باشد، True در b، وگرنه False در b قرار می‌گیرد.

به سه طریق می‌توان به متغیر از نوع تهی پذیر مقدار داد که عبارت‌اند از:

۱. مقداردهی در هنگام تعریف

۲. مقداردهی از طریق یک متغیر تهی پذیر دیگر

۳. مقدار null

به‌عنوان‌مثال، دستور زیر را مشاهده کنید:

```
int? myI1, myI2, myI3;
myI1 = 28;          // Value of underlying type
myI2 = myI1;       // Value of nullable type
myI3 = null;      // Null
```

دستور اول، سه متغیر به نام‌های myI1 , myI2 , myI3 از نوع تهی پذیر تعریف می‌کند، دستور دوم، مقدار ۲۸ را در myI1 قرار می‌دهد، دستور سوم، مقدار متغیر تهی پذیر myI1 را در متغیر تهی پذیر myI2 قرار می‌دهد و دستور چهارم، مقدار (null) را در متغیر myI3 قرار می‌دهد.

دستورات

دستورات در زبان #C دو نوع‌اند:

۱. **دستورات ساده**، هر دستور که با یک ; خاتمه می‌یابد، **دستور ساده** (Simple statement) نام دارد.

به‌عنوان‌مثال، دستور زیر را مشاهده کنید:

```
int    i = 5 ;
```

---

[1].Nullable Types

این دستور، متغیر i را از نوع int تعریف کرده، مقدار اولیه ۵ را در آن قرار می‌دهد.

۲. **بلاک**، مجموعه‌ای از صفر یا چند دستور مرتبط به هم که در داخل بلاک باز ({) و بلاک بسته (}) قرار می‌گیرند، را **بلاک** گویند.

به‌عنوان‌مثال، دستورات مقابل را ببینید:

```
{
  int  i = 5;
  int  j = 10;
   …
}
```

این دستورات تشکیل بلاک را می‌دهند.

ساختار برنامه #C

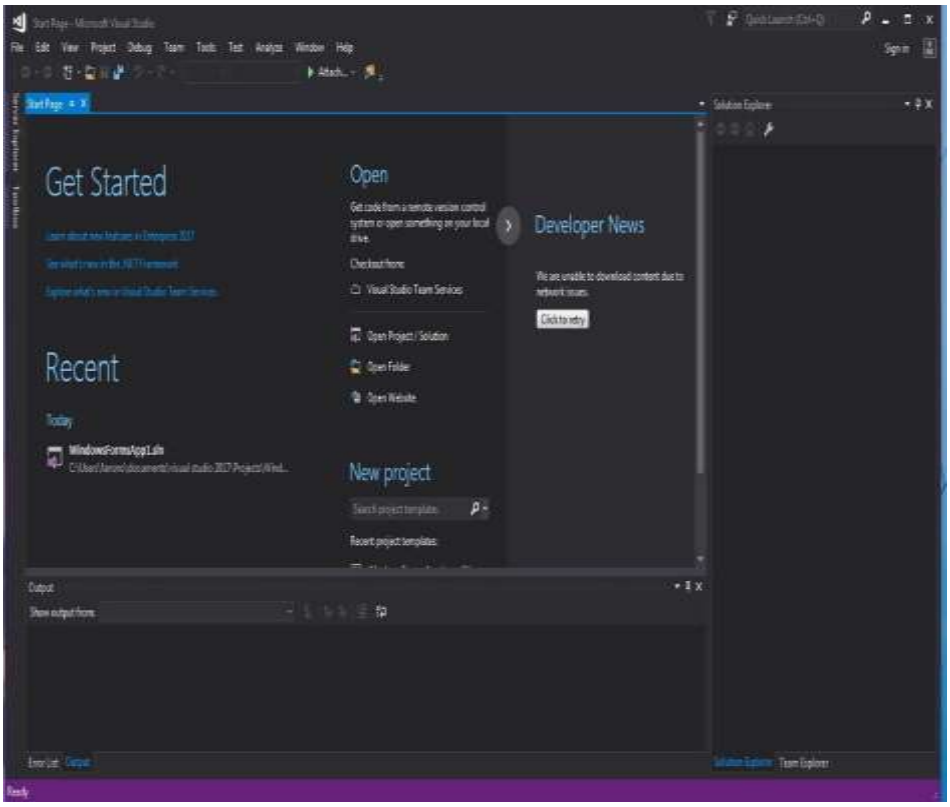
برای این‌که با ساختار برنامه #C آشنا شویم، یک برنامه جدید ایجاد می‌کنیم و از روی برنامه جدید ساختار برنامه را می‌آموزیم. برای ایجاد برنامه جدید، مثال ۱-۱ را ببینید.

مثال ۱- ۱. برنامه‌ای که مراحل ایجاد و اجرای یک برنامه در #C را نشان می‌دهد (هدف این برنامه آشنایی با ایجاد و اجرای برنامه در #C است).

مراحل طراحی و اجرا

۱. نرم‌افزار #C را اجرا کنید. برای این منظور گزینه زیر را اجرا نمایید:

۲. اکنون صفحه اول ویژوال استودیو ظاهر می‌شود(شکل۸ - ۱).

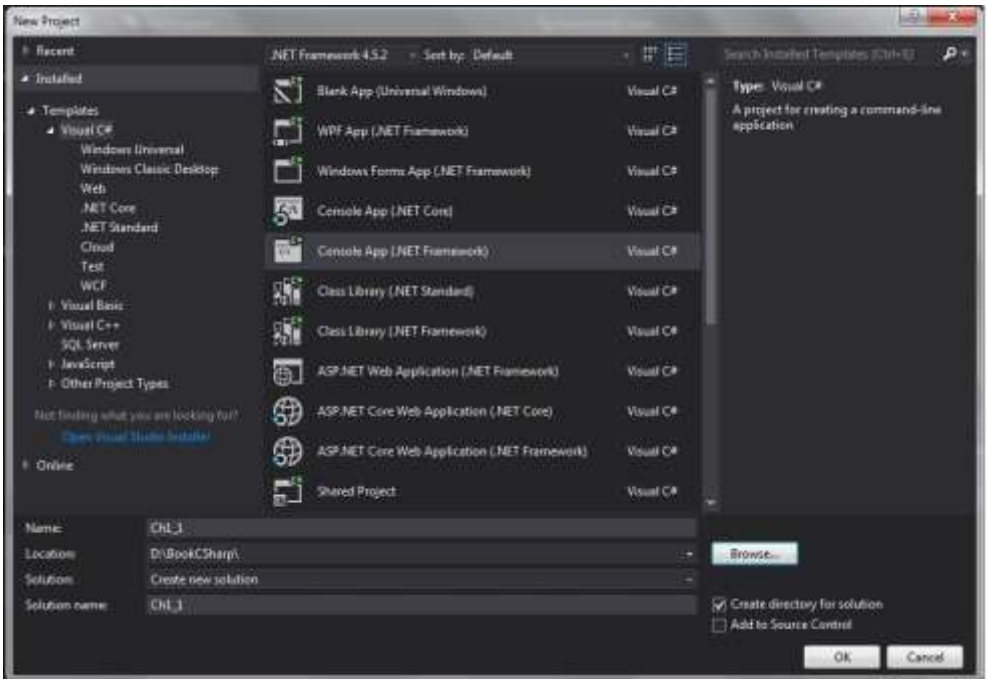


شکل ۸ – ۱ صفحه اول ویژوال استودیو.

۳. گزینه File/New/Project (کلیدهای Ctrl+Shift+N) را اجرا کنید یا آیکن 

را کلیک نمایید. درهرصورت، پنجره New Project ظاهر می‌شود (شکل ۹ – ۱). در این پنجره اطلاعات زیر را انتخاب کنید:

- در سمت چپ، زبان برنامه‌نویسی و نوع برنامه‌نویسی را انتخاب کنید. در این برنامه، زبان Visual C# و نوع برنامه‌نویسی را Windows انتخاب نمایید.
- در پنجره وسط، نوع برنامه C# را انتخاب نمایید. در این برنامه نوع Console App(.NET Framwork) را انتخاب کنید.
- در جلوی Name، نام پروژه را وارد کنید. در این برنامه Ch1_1 انتخاب‌شده است.
- در بخش Location مکان ذخیره برنامه را انتخاب کنید.



شکل ۹ – ۱ پنجره New Project.

۴. دکمه OK را کلیک کنید تا برنامه جدید ایجاد شود. دستورات این برنامه به‌صورت زیر می‌باشند:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Ch1_1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

این برنامه دارای ۵ بخش اصلی است:

۱. بخش فضاهای نام موردنیاز برنامه، این بخش فضاهای نام موردنیاز برنامه را معرفی می‌کند. این دستورات با دستور using شروع می‌شوند (دستورات زیر):

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

فضاهای نام‌هایی که نیاز ندارید را می‌توانید از برنامه‌تان حذف کنید. در اکثر برنامه‌ها فقط به فضای نام System نیاز است. بنابراین می‌توانید بقیه فضاهای نام را از برنامه حذف نمایید.

در فضای نام System توابع کتابخانه‌ای System نظیر Write()، WriteLine()، Read() و ReadLine() دستورات ورودی و خروجی کنسول وجود دارند. در ادامه با این توابع آشنا خواهید شد.

۲. تعریف فضای نام پروژه، هر برنامه جدیدی که ایجاد می‌کنید، فضای نام جدیدی به نام پروژه ایجاد می‌شود. در این برنامه این فضای نام با دستور زیر ایجاد گردید:

```
namespace Ch1_1
```

۳. تعریف کلاس Program، هر برنامه جدیدی که ایجاد می‌کنید یک کلاس به نام Program ایجاد می‌شود. از طریق این کلاس می‌توان **فیلدها، متدها، واسط‌ها** [1] را ایجاد کرد. با مفهوم کلاس در فصل‌های ۵ و ۶ بیش‌تر آشنا خواهید شد. کلاس Program با دستور زیر ایجاد گردید:

```
class Program
```

۴. متد Main()، یکی از مهم‌ترین متدهای کلاس Program می‌باشد (وجود متد Main() در تمام برنامه‌های اجرایی ضروری است). این متد به‌صورت زیر تعریف‌شده است:

```
static void Main(string[] args)
```

همان‌طور که در این دستور مشاهده می‌کنید، متد Main() با کلمه کلیدی static تعریف‌شده است. این کلمه Modifier نام دارد. کلمه کلیدی static بیان می‌کند که این متد فقط در همین کلاس قابل‌اجرا می‌باشد و از طریق هیچ **نمونه** [2] دیگری قابل‌اجرا نمی‌باشد (در فصل‌های ۵ و ۶ با متدهای static بیش‌تر آشنا خواهید شد). کلمه void در این متد تعیین می‌کند که این متد هیچ مقداری را برگشت نمی‌دهد. با تعریف متدها در فصل سوم بیش‌تر آشنا خواهید شد. کلمه args، آرگومان‌هایی را تعیین می‌کنند که از طریق خط فرمان می‌توان برای متد Main() ارسال کرد.

۵. دستورات متد Main()، در این برنامه دستورات متد Main() به‌صورت زیر می‌باشند:

```
{
}
```

چون این برنامه هیچ عملی را انجام نمی‌دهد، بنابراین هیچ دستوری ندارد.

۵. پروژه را ذخیره کنید. برای این منظور، گزینه File/Save All را اجرا نمایید.

۶. پروژه را اجرا کنید. برای این منظور، یکی از اعمال زیر را انجام دهید:

- کلید F5 را فشار دهید.
- گزینه Debug/Start Debugging را اجرا نمایید.
- دکمه ▶ را کلیک کنید.

---

[1].Interpace [2].Instance

درهرصورت پروژه اجراشده، صفحه سیاهی نمایش داده می‌شود و سریع رد می‌گردد. این صفحه سیاه، صفحه خروجی برنامه نام دارد.

دستورات خروجی

کنسول ویندوز، خط فرمان ساده ویندوز است که اجازه می‌دهد یک برنامه متنی را نمایش داده و داده‌ها را از صفحه‌کلید دریافت کند. برای انجام این کار در C# کلاسی به نام Console (در فضای نام System) وجود دارد که شامل متدهای برای ورودی و خروجی داده در یک کنسول ویندوز است. این کلاس دارای متدهای مختلفی است که متدهای ورودی و خروجی را در ادامه می‌بینید.

متدهای خروجی

کلاس Console دارای دو متد برای نمایش داده در خروجی است. این دو متد عبارت‌اند از:

جدول ۴- ۱ کاراکترهای کنترلی.

کاراکتر	کد اسکی	هدف
\’	0x0027	کاراکتر تک کتیشن (‘) را تعیین می‌کند.
\”	0x0022	کاراکتر جفت کتیشن (“) را تعیین می‌کند.
\\	0x005C	کاراکتر بک‌اسلش (Backslash) را تعیین می‌کند.
\0	0x0000	کاراکتر تهی (null) را تعیین می‌کند.
\a	0x0007	بوق سیستم را به صدا در می‌آورد.
\b	0x0008	کاراکترBackspace را تعیین می‌کند.
\f	0x000C	مکان‌نما را به صفحه بعدی انتقال می‌دهد.
\n	0x000A	مکان‌نما را به سطر بعدی انتقال می‌دهد.
\r	0x000D	کاراکتر Carrage return را تعیین می‌کند.
\t	0x0009	مکان‌نما را به Tab افقی بعدی انتقال می‌دهد.
\v	0x000B	مکان‌نما را به Tab عمودی بعدی انتقال می‌دهد.

متد ()Write، برای نمایش داده در خروجی به کار می‌رود. این متد به‌صورت زیر استفاده می‌شود:

Console.Write(formatString , subVal0, subVal1, …)

همان‌طور که در این متد می‌بینید، متد Write چند پارامتر را می‌پذیرد. این پارامترها عبارت‌اند از:

۱. پارامتر formatString، رشته فرمت نام دارد. این رشته می‌تواند شامل علامت‌گذاری‌های جایگزینی[1] باشد. یک علامت‌گذار جایگزینی، مکانی را در رشته فرمت، علامت‌گذاری خواهد کرد که از یک مقدار

[1].Subsutitution

عددی که در داخل {} قرار دارد، تشکیل می‌شود. در رشته فرمت می‌توانید از کاراکترهای کنترلی استفاده کنید، این کاراکترها در جدول ۴ – ۱ آمده‌اند.

۲. مقادیر subVal0 , subVal1 و ...، مقادیری هستند که باید جایگزین علامت‌گذاری‌های جایگزینی در خروجی شوند. این مقادیر جایگزینی شماره‌دارند که از صفر (۰) شروع می‌شود. یعنی، اولین مقدار دارای شماره صفر، دومین مقدار شماره یک و این روند ادامه می‌یابد. به‌عنوان‌مثال، دستور زیرا را ببینید:

```
Console.WriteLine("Two sample integers are {0} and {1}.",3,6);
```

این دستور خروجی زیر را نمایش می‌دهد:

```
Two sample integers are 3 and 6.
```

همان‌طور که در این خروجی مشاهده می‌گردد، مقدار جایگزینی ۳، جایگزین علامت گذار جایگزینی {0} و مقدار جایگزینی 6 جایگزین علامت گذار جایگزینی {1} شده است.

متد WriteLine()، همانند متد Write() است. با این تفاوت که پس از نمایش اطلاعات در خروجی مکان نما را به سطر بعد خروجی انتقال می‌دهد. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
Console.Write("One ");
Console.Write("Two ");
```

این دستورات خروجی مقابل را نمایش می‌دهند: One Two

اکنون دستورات زیر را ببینید:

```
Console.WriteLine("One ");
Console.WriteLine("Two ");
```

این دستورات خروجی زیر را نمایش می‌دهند:

```
One
Two
```

متد ReadKey()، برای ایجاد مکث (توقف موقت) در برنامه به کار می‌رود و به‌صورت زیر استفاده می‌شود: Console.ReadKey();

وقتی کنترل اجرای برنامه به این متد برسد، اجرای برنامه به‌طور موقت قطع خواهد شد تا کاربر کاراکتری را فشار دهد. به‌محض این‌که کاربر کاراکتری را فشار داد، اجرای برنامه ادامه می‌یابد. این متد معمولاً برای ایجاد توقف به کار می‌رود تا کاربر خروجی برنامه را ببیند.

مثال ۲–۱. برنامه‌ای که عبارت First output in c# را نمایش می‌دهد.

مراحل طراحی و اجرا

۱. پروژه جدیدی به نام Ch1-2ایجاد کنید، برای این منظور، گزینه File/New/Project را اجرا نمایید تا پنجره New Project ظاهر شود. در این پنجره، نوع برنامه را Console Application و نام آن را Ch1-2 انتخاب کرده، دکمه OK را کلیک کنید.

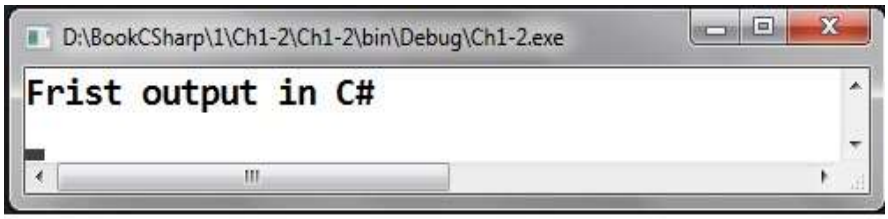
۲. به کلاس Program بروید و دستورات آن را به‌صورت زیر تغییر دهید:

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Ch1_2
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Frist output in C#");
            Console.ReadKey();
        }
    }
}
```

دستور اول داخل متد Main() عبارت First output in C# را نمایش می‌دهد و دستور دوم مکث کوتاهی ایجاد خواهد کرد تا کاربر کلیدی را فشار دهد.

۳. پروژه را ذخیره و اجرا کنید تا خروجی را به شکل زیر مشاهده نمایید:



توجه کنید که زمینه خروجی در حالت عادی مشکی می‌باشد که برای خوانایی کتاب آن را به سفید تغییر دادیم.

مثال ۳–۱. برنامه‌ای که کاربرد کاراکترهای کنترلی، جا نگه‌دار، متدهای Write و WriteLine را نمایش می‌دهد.

مراحل طراحی و اجرا

۱. پروژه جدیدی به نام Ch1-3 ایجاد کنید. برای این منظور، گزینه File/New/Project را اجرا نمایید تا پنجره New Project ظاهر شود. در این پنجره، نوع برنامه را Console Application و نام آن را Ch1-3 انتخاب کرده، دکمه OK را کلیک کنید.

۲. به کلاس Program بروید و دستورات آن را به‌صورت زیر تغییر دهید:

```
using System;
namespace Ch1_3
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("One\t{0}\nTwo\t{1}\t", 1, 2);
         Console.WriteLine("Fanavarienovin\t{0}\nKetabrah\t{1}\\",
          "www.fanavarienovin.net", "wwww.ketabrah.ir");
```

```
            Console.WriteLine("Three\t{0}\tFour\t{1}\'", 3, 4);
             Console.ReadKey();
        }
    }
}
```

اولین دستور تابع Main() متد Write() با کلاس Console ابتدا عبارت One را می‌نویسد و با کاراکتر کنترلی '\t' به تب بعدی می‌رود، سپس مقدار ۱ (اولین مقدار بعد از " بسته جایگزین {0} می‌شود) را نمایش می‌دهد، در ادامه با کاراکتر '\n' کنترل چاپ را به سطر بعدی انتقال می‌دهد و کلمه Two را نمایش می‌دهد. در پایان با کاراکتر '\t' کنترل چاپ به تب بعدی می‌رود و ۲ را نمایش می‌دهد (۲، جایگزین {1} می‌شود) . کنترل چاپ به تب بعدی انتقال می‌یابد. دستور دوم، ابتدا کلمه Fanavarienovin را نمایش می‌دهد، کنترل چاپ را به تب بعدی انتقال می‌دهد و مقدار www.fanavarienovin.net را به‌جای جا نگه‌دار {0} نمایش می‌دهد، سپس کنترل چاپ را با کاراکتر '\n' به سطر بعدی انتقال می‌دهد، کلمه ketabrah را نمایش داده، با کاراکتر '\t' کنترل چاپ را به تب بعدی منتقل کرده و عبارت www.ketabrah.ir را به‌جای جا نگه‌دار {1} نمایش می‌دهد، در پایان کاراکتر \ را نمایش داده (کاراکتر \\ برای چاپ کاراکتر \ به کار می‌رود) و کنترل چاپ را به سطر بعدی می‌برد (چون، متد WriteLine استفاده‌شده است). دستور سوم، ابتدا کلمه Three را نمایش داده و کنترل چاپ را به تب بعد انتقال می‌دهد، سپس مقدار ۳ را به‌جای جا نگه‌دار {0} نمایش می‌دهد و کنترل چاپ را به تب بعدی انتقال داده، Four را نمایش می‌دهد، در پایان، کنترل چاپ را به tab بعدی انتقال داده، به‌جای نگه‌دار {1} مقدار ۴ را نمایش می‌دهد و بالاخره ' (کاراکتر '\ کوتیشن) را نمایش می‌دهد و به سطر بعدی می‌رود.

۳. پروژه را ذخیره و اجرا کنید تا خروجی را به شکل زیر مشاهده نمایید:

متغیر

برای نگه‌داری هر چیزی لازم است که از یک ظرف متناسب با آن استفاده نمود. به‌عنوان‌مثال، در خانه برای نگه‌داری مواد غذایی، ظروف مختلفی وجود دارند که هرکدام برای نگه‌داری مواد خاصی به کار می‌روند. مثلاً، بطری برای نگه‌داری آب و... به همین ترتیب در برنامه‌نویسی، برای نگه‌داری مقادیر از ظروف مخصوصی استفاده می‌شود. ظرف نگه‌داری داده در زبان‌های برنامه‌نویسی **متغیر** نام دارد. **بنابراین، متغیر نامی است برای یک مکان از حافظه که ممکن است که در طول اجرای برنامه مقدار آن تغییر کند. ولی، در یک‌لحظه فقط یک مقدار را دارد.**

برای استفاده از متغیرها سه عمل باید انجام شود که عبارت‌اند از:

۱. نام‌گذاری متغیرها

بعدازاین که یک بچه به دنیا آمد، برای شناسایی او نامی انتخاب می‌کنید. جهت مراجعه به متغیرها نیز از نام آن‌ها استفاده می‌شود. برای نام‌گذاری بچه‌ها ثبت‌احوال از قوانینی خاصی پیروی می‌کند، به‌عنوان‌مثال، اجازه نمی‌دهد نام بچه را **رضا۱** انتخاب کنید. در زبان C# نیز برای نام‌گذاری متغیرها قوانین زیر وجود دارد:

۱. نام متغیر می‌تواند ترکیبی از حروف a تا z یا A تا Z ارقام، خط ربط (–)، ارقام ۰ تا ۹ باشد.

۲. حرف اول متغیر نمی‌تواند ارقام ۰ تا ۹ باشد.

۳. نام متغیر می‌تواند دارای هر طولی باشد.

۴. زبان C# بین حروف بزرگ و کوچک فرق می‌گذارد. یعنی، متغیرهای Count و count باهم فرق دارند.

۵. نام متغیر نمی‌تواند از همنام با کلمات کلیدی یا نام توابع انتخاب شود. برخی از نام‌های مجاز برای متغیر عبارت‌اند از: area، sum، sum1، pr_1 و .... اما، نام‌های زیر برای متغیر مجاز نیستند:

- **نام 2test:** نام متغیر نمی‌تواند با ارقام 0 تا 9 شروع شود.
- **نام test$**: در نام متغیر نمی‌توان از $ استفاده کرد.
- **نام store 2**: در نام متغیر نمی‌توان از کاراکتر فاصله[1] استفاده کرد.
- **نام jpg.:** نام متغیر نمی‌تواند با کاراکتر نقطه (.) شروع شود.

۲. معرفی متغیرها

همان‌طور که بیان گردید، هر ظرفی برای نگه‌داری نوعی غذا به کار می‌رود. بنابراین، متغیرها نیز باید دارای نوع باشند تا بتوانند انواع داده‌ها را ذخیره کنند. چون داده‌ها دارای انواع مختلف هستند، بنابراین متغیرها که داده‌ها را نگه‌داری می‌کنند، باید دارای نوع باشند. نوع متغیر تعیین می‌کند اولاً چه نوع داده‌ای می‌تواند در آن متغیر قرار گیرد و ثانیاً، این متغیر به چند بایت از حافظه نیاز دارد. تعیین نوع متغیر به‌صورت زیر می‌باشد:

نوع داده‌ای لیست متغیرها ;

نوع داده‌ای، یکی از انواع داده بیان‌شده نظیر int، float، double و غیره در C# می‌باشد. اگر تعداد متغیرها بیش از یکی باشند، با کاما (,) از هم جدا می‌شوند.

مثال ۴–۱. برنامه‌ای که متغیرهای a، b و c را با نوع int، d را با نوع double، f_1 و f_2 را با نوع float و ch را با نوع کاراکتر تعریف می‌کند.

۱. پروژه جدیدی به نام Ch1-4 ایجاد کنید.

۲. به کلاس Program بروید و دستورات آن را به‌صورت زیر تغییر دهید:

```
using System;
namespace Ch1_4
{
```

[1] .blank(space)

```
    class Program
    {
        static void Main(string[] args)
        {
            int a, b, c;
            double d;
            float f1, f2;
            char ch;
            Console.ReadKey();
        }
    }
}
```

دستور اول متد ()Main، متغیرهای a، b و c را با نوع int تعریف می‌کند، دستور دوم متغیر d را با نوع double تعریف می‌نماید، دستور سوم، متغیرهای f1 و f2 را با نوع اعشاری (float) تعریف می‌کند، دستور چهارم، متغیر ch را با نوع کاراکتری تعریف می‌نماید. اگر به ویراستار برنامه دقت کرده باشید، زیر این متغیرها **خط سبز موج** دارد رسم شده است. رسم این خط به این دلیل است که این متغیرها تعریف‌شده‌اند. اما، مقداردهی و استفاده‌نشده‌اند.

۳. پروژه را ذخیره و اجرا کنید تا خروجی را به شکل زیر مشاهده نمایید:

مقداردهی به متغیرها

هر متغیر دارای **نام**، **نوع**، **اندازه** و یک **مقدار** است.

سه روش برای مقداردهی به متغیرها وجود دارد، یکی از روش‌ها، مقداردهی به متغیرها در هنگام تعریف آن است.

مثال ۵ – ۱. برنامه زیر متغیرهای a و b را با نوع int تعریف کرده، مقادیر ۱۰ و ۱۲ را به آن‌ها تخصیص می‌دهد و متغیر PI را با نوع float با مقدار 3.14 تعریف می‌کند، ch را با نوع کاراکتری و مقدار 'F' تعریف کرده، متغیر yes را با نوع منطقی و مقدار true تعریف می‌نماید..

۱. پروژه جدیدی به نام Ch1-5 ایجاد کنید.

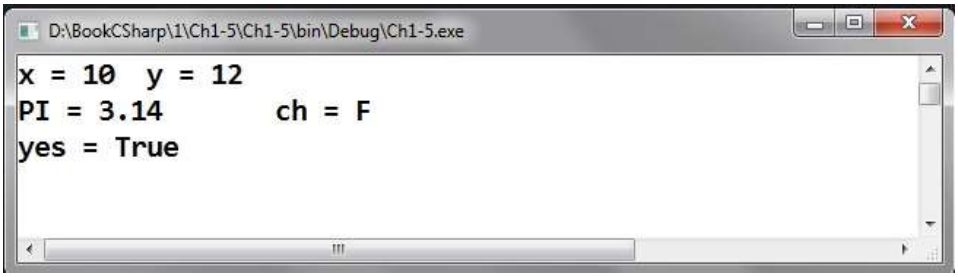
۲. به کلاس Program بروید و دستورات آن را به‌صورت زیر تغییر دهید:

```
using System;
namespace Ch1_5
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 10, y = 12;
            float PI = 3.14f;
            char ch = 'F';
```

```
            bool yes = true;
            Console.WriteLine("x = {0}\ty = {1}", x, y);
            Console.WriteLine("PI = {0}\tch = {1}\nyes = {2}",
               PI,  ch, yes);
            Console.ReadKey();
        }
    }
}
```

دستور اول متد ()Main، متغیرهای x را با مقدار اولیه ۱۰ و y را با مقدار اولیه ۱۲ تعریف می‌کند، دستور دوم، متغیر PI را با نوع float و مقدار اولیه 3.14f (f تعیین می‌کند که داده 3.14 اعشاری است) تعریف می‌کند، دستور سوم، متغیر ch را با نوع char (کاراکتری) و مقدار 'F' تعریف می‌نماید، دستور چهارم، متغیر yes را با نوع bool (منطقی) و مقدار true تعریف می‌نماید، دستور پنجم ابتدا، عبارت =x را نمایش می‌دهد، سپس مقدار متغیر x (یعنی ۱۰) را به‌جای جا نگه‌دار {0} نمایش می‌دهد، در ادامه کنترل چاپ را با کاراکتر '\t' به تب بعدی انتقال می‌دهد و عبارت = y را نمایش می‌دهد، در پایان، مقدار متغیر y (یعنی ۱۲) را به‌جای جا نگه‌دار {1} نمایش می‌دهد و دستور ششم، ابتدا عبارت =PI را نمایش داده، مقدار متغیر PI را به‌جای جا نگه‌دار {0} نمایش می‌دهد، در ادامه کنترل چاپ را با کاراکتر '\t' به تب بعدی انتقال می‌دهد و عبارت = y را نمایش می‌دهد، در پایان، مقدار متغیر y (یعنی ۱۲) را به‌جای جا نگه‌دار {1} نمایش می‌دهد و دستور ششم، ابتدا عبارت =PI را نمایش داده، مقدار متغیر PI را به‌جای نگه‌دار {0} نمایش می‌دهد، سپس کنترل چاپ را به تب بعدی انتقال می‌دهد، عبارت =Ch را نمایش داده و مقدار متغیر ch (یعنی 'F') را به‌جای نگه‌دار {1} چاپ می‌کند و در پایان با کنترل '\n' به خط بعدی می‌رود، عبارت =yes را نمایش داده، مقدار متغیر yes (یعنی True) را به‌جای جا نگه‌دار {2} چاپ می‌کند و به سطر بعدی می‌رود.

۳. پروژه را ذخیره و اجرا کنید تا خروجی را به شکل زیر مشاهده نمایید:



بعد از تعریف متغیر نیز می‌توان به آن‌ها مقدار داد. برای این منظور می‌توانید از دستور انتساب (عملگر =) یا دستورات ورودی استفاده کنید. نمونه‌ای از کاربرد عملگر = در زیر آمده است. در ادامه، دستورات ورودی جهت تخصیص مقدار به متغیرها را می‌بینید.

مثال ۶ – ۱. برنامه‌ای که سه متغیر به نام‌های a، f و yes را به ترتیب با انواع int، float و bool تعریف می‌کند و سپس، به آن‌ها مقادیر ۱۰، ۱۳,۷ و false را تخصیص می‌دهد.

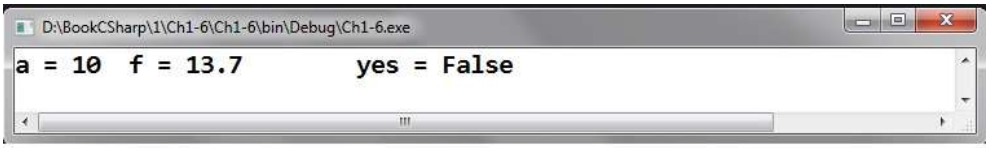
۱. پروژه جدیدی به نام Ch1-6 ایجاد کنید.

۲. به کلاس Program بروید و دستورات آن را به‌صورت زیر تغییر دهید:

```
using System;
namespace Ch1_6
{
    class Program
    {
        static void Main(string[] args)
        {
            int a;
            float f;
            bool yes;
            a = 10;
            f = 13.7f;
            yes = false;
            Console.WriteLine("a = {0}\tf = {1}\tyes = {2}",
              a, f, yes);
            Console.ReadKey();
        }
    }
}
```

دستورات اول تا سوم متد Main() به ترتیب متغیرهای a را با نوع int، f را با نوع float و yes را با نوع bool تعریف می‌کنند، دستورات چهارم تا ششم، به ترتیب به متغیرهای a، f و yes مقادیر ۱۰، 13.7f و false تخصیص می‌دهند و دستور هفتم، ابتدا عبارت a= را نمایش داده، به‌جای جا نگه‌دار {0} مقدار ۱۰ (مقدار a) را نمایش می‌دهد، با کاراکتر '\t' به تب بعدی می‌رود و عبارت f= را نمایش داده، جلوی آن به‌جای جا نگه‌دار {1} مقدار 13.7 (مقدار متغیر f) را نمایش می‌دهد و در پایان عبارت yes= را نمایش داده و روبروی آن مقدار false (مقدار متغیر yes) را به‌جای جا نگه‌دار {2} نمایش می‌دهد و کنترل چاپ را به سطر بعدی انتقال می‌دهد.

۳. پروژه را ذخیره و اجرا کنید تا خروجی را به شکل زیر مشاهده نمایید:



نکته: وقتی مقدار جدیدی در متغیر قرار می‌گیرد، این مقدار جایگزین مقدار قبلی می‌شود. یعنی، مقدار قبلی از دست می‌رود (حذف می‌گردد).

۱۴-۱. ثابت‌ها

ثابت شناسه‌ای (نام خانه‌ای از حافظه) است که مقدار آن در طول اجرای برنامه تغییر نمی‌کند. ثابت‌ها انواع مختلف دارند. ثابت‌ها می‌توانند عددی صحیح، اعشاری، کاراکتری، رشته‌ای یا منطقی باشند. ثابت‌های کاراکتری بین تک کتیشن (') قرار می‌گیرند (مانند 'C')، ثابت‌های رشته‌ای بین جفت کتیشن قرار می‌گیرند

(نظیر "C#") و ثابت‌های منطقی مقادیر true یا false هستند. به‌عنوان‌مثال، مقدار 3.1415، (عدد []) را مشخص می‌کند. این عدد یا هر ثابت دیگر ممکن است چندین مرتبه در برنامه استفاده شود. به دلیل راحتی اصلاح و تغییر مقدار ثابت‌ها به آن‌ها نام تخصیص می‌دهند. تعریف ثابت در C# به صورت‌های زیر انجام می‌شود:

1.const نوع ثابت نام ثابت = مقدار ثابت;

به‌عنوان‌مثال، دستور مقابل را ببینید:

```
const float  PI = 3.1415f;
```

این دستور ثابت PI را با مقدار 3.1415 تعریف می‌کند. اکنون دستور مقابل را ببینید.

```
PI = 3.141505;
```

این دستور نادرست است. زیرا، مقدار ثابت را نمی‌توان تغییر داد.

۱۵ – ۱. عملگرها

عملگرها[1]،نمادهایی هستند که اعمال خاصی را بر روی‌داده انجام می‌دهند. عملگرها انواع مختلف دارند که برخی از آن‌ها عبارت‌اند از:

۱. عملگرهای محاسباتی **۲.** عملگرهای رابطه‌ای (مقایسه‌ای)

۳. عملگرهای ترکیبی **۴.** عملگرهای منطقی

۵. عملگرهای خاص

جدول ۵ – ۱ عملگرهای محاسباتی.

عملگر	نام عملگر	مثال	نتیجه	توضیحات
+	جمع	۱۲ + ۳	۱۵	عملوند اول را با عملوند دوم جمع می‌کند.
-	تفریق	۱۳,۵ – ۳	۱۰,۵	عملوند دوم را از عملوند اول کم می‌کند.
*	ضرب	۱۲* ۲,۵	۳۰	عملوند اول را در عملوند دوم ضرب می‌کند.
/	تقسیم	۱۳ / ۲	۶	عملوند اول را بر عملوند دوم تقسیم می‌کند.
%	باقی‌مانده تقسیم صحیح	۱۳ ٪ ۵	۳	باقی‌مانده تقسیم صحیح عملوند اول بر عملوند دوم را محاسبه می‌کند.
++	افزایش	x = 10; x++;	۱۱	یک واحد به عملوند اضافه می‌کند.
--	کاهش	x = 10; x --;	۹	یک واحد از عملوند کم می‌نماید.

[1] .operators

۱ – ۱۵ – ۱. عملگرهای محاسباتی

این عملگرها برای انجام محاسبات بر روی داده‌های عددی به کار می‌روند (جدول ۵ – ۱). ازجمله این عملگرها می‌توان عملگرهای + (جمع)، - (تفریق)، * (ضرب)، / (تقسیم)، ٪ (باقی‌مانده تقسیم صحیح)، ++ (افزایش) و -- (کاهش) را نام برد. عملکرد عملگرهای +، -، * و / را از قبل می‌دانید. عملگر ٪ برای محاسبه باقی‌مانده تقسیم صحیح به کار می‌رود.

مثال ۷ – ۱. برنامه‌ای که کاربرد عملگرهای محاسباتی را بیان می‌کند.

۱. پروژه جدیدی به نام Ch1-7 ایجاد کنید.

۲. به کلاس Program بروید و دستورات آن را به‌صورت زیر تغییر دهید:

```
using System;
namespace ch1_7
{
    class Program
    {
        static void Main(string[] args)
        {
            int n1 = 10, n2 = 3;
            int n3 = n1 + n2;
            int n4 = n1 - n2;
            int n5 = n1 * n2;
            int n6 = n1 / n2;
            int n7 = n1 % n2;
            Console.WriteLine("{0} + {1} = {2}", n1, n2, n3);
            Console.WriteLine("{0} - {1} = {2}", n1, n2, n4);
            Console.WriteLine("{0} * {1} = {2}", n1, n2, n5);
            Console.WriteLine("{0} / {1} = {2}", n1, n2, n6);
            Console.WriteLine("{0} % {1} = {2}", n1, n2, n7);
            Console.ReadKey();
        }
    }
}
```

دستور اول متد ()Main، متغیرهای n1 و n2 را به ترتیب با مقادیر ۱۰ و ۳ تعریف می‌کند، دستور دوم، متغیر n3 را تعریف کرده، مقدار n1 + n2 را در آن قرار می‌دهد، دستور سوم، متغیر n4 را تعریف کرده، n1 – n2 را در آن قرار می‌دهد، دستور چهارم، n5 را تعریف کرده، مقدار n1*n2 را در آن قرار می‌دهد، دستور پیچم، متغیر n6 را تعریف کرده، حاصل عبارت n1/n2 را در آن قرار می‌دهد، دستور ششم، متغیر n7 را تعریف کرده، حاصل n1%n2 (باقی‌مانده تقسیم صحیح n1 بر n2) را در آن قرار می‌دهد، دستور هفتم، مقدار 10 + 3 = 13 را نمایش می‌دهد، دستور هشتم، مقدار 10 – 3 = 7 را نمایش خواهد داد، دستور نهم، مقدار 10 * 3 = 30 را نمایش می‌دهد، دستور دهم، عبارت 10 / 3 = 3 را چاپ خواهد کرد (چون n1 و n2 صحیح هستند، نتیجه تقسیم نیز عددی صحیح خواهد شد) و دستور یازدهم، عبارت 10 % 3 = 1 را نمایش می‌دهد (چون باقی‌مانده تقسیم صحیح ۱۰ بر ۳ برابر ۱ است).

۳. پروژه را ذخیره و اجرا کنید تا خروجی را به شکل زیر مشاهده نمایید:

عملگر ++ یک واحد به محتویات عملوند اضافه می‌کند. اما، عملگر -- یک واحد از محتویات عملوند کم خواهد کرد. چنانچه عملگرهای ++ و -- قبل از عملوند قرار گیرند، ابتدا یک واحد به محتویات عملوند اضافه کرده یا از آن کسر می‌کند. سپس، عبارت ارزیابی می‌گردد (مثال زیر را ببینید). اما اگر عملگرهای + + یا - - بعد از عملوند قرار گیرند، ابتدا عبارت ارزیابی خواهد شد و سپس یک واحد به مقدار عملوند اضافه خواهد شد یا از آن کم می‌گردد (به مثال زیر دقت کنید).

مثال ۸ – ۱. برنامه‌ای که عملکرد عملگرهای ++ و --را بیان می‌کند.

۱. پروژه جدیدی به نام Ch1-8 ایجاد کنید.

۲. به کلاس Program بروید و دستورات آن را به‌صورت زیر تغییر دهید:

```
using System;
namespace Ch1_8
{
    class Program
    {
        static void Main(string[] args)
        {
            int n1 = 10, n2 = 20;
            int n3 = n1 --;
            int n4 = -- n2;
            int n5 =  n1 ++;
            int n6 = ++ n2 ;
            Console.WriteLine("n1 = {0} \t n2 = {1}", n1, n2);
            Console.WriteLine("n3 = {0} \t n4 = {1}", n3, n4);
            Console.WriteLine("n5 = {0}  \t n6 = {1}", n5, n6);
            Console.ReadKey();
        }
    }
}
```

جدول ۶ – ۱ عملگرهای رابطه‌ای (مقایسه‌ای).

جدول ۶ – ۱ عملگرهای رابطه‌ای (مقایسه‌ای).

عملگر	نام عملگر	مثال	نتیجه	توضیحات
>	بزرگ‌تر	۲ > ۳	false	اگر عملوند اول بزرگ‌تر از عملوند دوم باشد، نتیجه درست است، وگرنه نتیجه نادرست می‌باشد.
>=	بزرگ‌تر یا مساوی	۵ >= ۳	true	اگر عملوند اول بزرگ‌تر یا مساوی عملوند دوم باشد، نتیجه درست است، وگرنه، نتیجه نادرست می‌باشد.
<	کوچک‌تر	۵ < ۷	true	اگر عملوند اول کوچک‌تر از عملوند دوم باشد، نتیجه درست است، وگرنه نتیجه نادرست است.
<=	کوچک‌تر یا مساوی	۵ <= ۳	false	اگر عملوند اول کوچک‌تر یا مساوی عملوند دوم باشد، نتیجه درست است، وگرنه نتیجه نادرست خواهد شد.
!=	نامساوی	۲ != ۵	true	اگر عملوند اول مخالف عملوند دوم باشد، نتیجه درست است، وگرنه، نتیجه نادرست خواهد بود.
==	تساوی	۲ == ۳	false	اگر عملوند اول مساوی عملوند دوم باشد، نتیجه درست است، وگرنه نتیجه نادرست خواهد شد.

دستور اول متد Main() متغیرهای n1 و n2 را با نوع int و مقادیر ۱۰ و ۲۰ تعریف می‌کند، دستور دوم، ابتدا متغیر n3 را تعریف کرده، مقدار ۱۰ را در آن قرار می‌دهد و سپس یک واحد از n1 کم می‌کند (یعنی، معادل دستورات زیر است):

```
n3 = n1;
n1 = n1 – 1;
```

دستور سوم، ابتدا متغیر n4 را با نوع int تعریف کرده، مقدار ۱۹ را در آن قرار می‌دهد. زیرا معادل، دستورات زیر است:

```
n2 = n2 - 1;
n4 = n2;
```

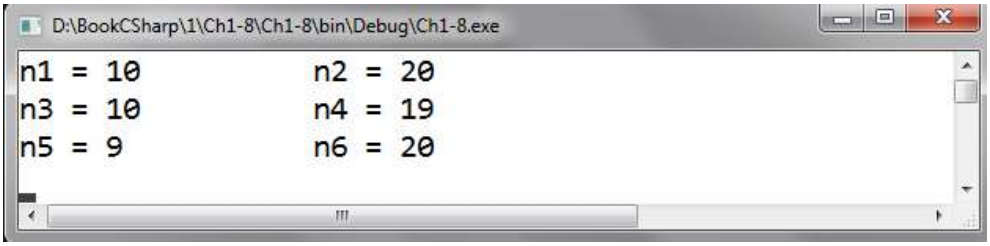
دستور چهارم، ابتدا متغیر n5 را با نوع int تعریف کرده، مقدار ۹ را در آن قرار می‌دهد و سپس به n1 یک واحد اضافه می‌نماید (معادل دو دستور زیر است):

```
n5 = n1;
n1 = n1 + 1;
```

دستور پنجم، متغیر n6 را تعریف کرده، ابتدا به n2 یک واحد اضافه می‌کند (یعنی n2 برابر با ۲۰ می‌شود) و این مقدار در n6 قرار می‌دهد (این دستور معادل دستورات زیر است):

```
n2 = n2 + 1;
n6 = n2;
```

۳. پروژه را ذخیره و اجرا کنید تا خروجی را به شکل زیر مشاهده نمایید:



۲ – ۱۵ – ۱. عملگرهای رابطه‌ای (مقایسه‌ای)

این عملگرها برای مقایسه دو عملوند به کار می‌روند و نتیجه درست یا نادرست را برمی‌گردانند. عملگرهای رابطه‌ای (مقایسه‌ای) در جدول ۶ – ۱ آمده‌اند. دقت کنید عملگر == تساوی (مساوی بودن) می‌باشد.

مثال ۹ – ۱. برنامه‌ای که عملکرد عملگرهای مقایسه‌ای را نشان می‌دهد.

۱. پروژه جدیدی به نام Ch1-9 ایجاد کنید.

۲. به کلاس Program بروید و دستورات آن را به‌صورت زیر تغییر دهید:

```
using System;
namespace Ch1_9
{
    class Program
    {
      static void Main(string[] args)
      {
       int n1 = 10, n2 = 20;
       Console.WriteLine("{0} ==  {1} is {2}", n1, n2, n1 == n2);
       Console.WriteLine("{0}  >  {1} is {2}", n1, n2, n1 > n2);
       Console.WriteLine("{0} >=  {1} is {2}", n1, n2, n1 >= n2);
       Console.WriteLine("{0}  <  {1} is {2}", n1, n2, n1 < n2);
       Console.WriteLine("{0} <=  {1} is {2}", n1, n2, n1 <= n2);
       Console.WriteLine("{0} !=  {1} is {2}", n1, n2, n1 != n2);
       Console.ReadKey();

      }
    }
}
```

دستور اول متد ()Main متغیرهای n1 و n2 را با نوع int و مقادیر ۱۰ و ۲۰ تعریف می‌کند، دستور دوم، عبارت 10 == 20 is false را نمایش می‌دهد، (چون ۱۰ برابر با ۲۰ نیست، نتیجه عملگر == برابر false خواهد بود)، دستور سوم، عبارت 10 > 20 is false را نمایش می‌دهد، چون ۱۰ از ۲۰ بزرگ‌تر نیست، دستور چهارم، عبارت 10 =< 20 is false را نمایش می‌دهد، دستور پنجم، عبارت 10 < 20 is True را نمایش می‌دهد (۱۰ از ۲۰ کوچک‌تر است). دستور ششم، عبارت 10 <= 20 is True را نمایش می‌دهد و دستور هفتم، 10 != 20 is True را نمایش می‌دهد.

۳. پروژه را ذخیره و اجرا کنید تا خروجی را به شکل زیر مشاهده نمایید:

۳-۱۵-۱. عملگرهای ترکیبی

این عملگرها، ترکیبی از عملگرهای محاسباتی و = هستند. عملکرد این عملگرها را در جدول ۷ – ۱ می‌بینید.

مثال ۱۰ – ۱. برنامه‌ای که عملکرد عملگرهای ترکیبی را نشان می‌دهد.

۱. پروژه جدیدی به نام Ch1-10 ایجاد کنید.

۲. به کلاس Program بروید و دستورات آن را به‌صورت زیر تغییر دهید:

```
using System;
namespace Ch1_10
{
    class Program
    {
        static void Main(string[] args)
        {
            int n1 = 10, n2 = 20;
            n1 += n2;
            Console.WriteLine("n1 = {0} \t n2 = {1}", n1, n2);
            n1 -= n2;
            Console.WriteLine("n1 = {0} \t n2 = {1}", n1, n2);
            n1 *= n2;
            Console.WriteLine("n1 = {0} \t n2 = {1}", n1, n2);
            n1 /= n2;
            Console.WriteLine("n1 = {0} \t n2 = {1}", n1, n2);
            n1 %= n2;
            Console.WriteLine("n1 = {0} \t n2 = {1}", n1, n2);
            Console.ReadKey();
        }
    }
}
```

دستور اول متد ()Main، متغیرهای n1 و n2 را با نوع int و به ترتیب با مقادیر ۱۰ و ۲۰ تعریف می‌کند، دستور دوم، n1 را برابر ۳۰ قرار می‌دهد، زیرا:

n1 + = n2 ⇨ n1 = n1 + n2 = 10 +20

دستور سوم، عبارت n2 = 20 n1 = 30 را نمایش می‌دهد، دستور چهارم، n1 را به مقدار ۱۰ برمی‌گرداند، زیرا:

n1 -= n2 ⇨ n1 = n1 – n2 = 30 – 10 = 10

دستور چهارم، مقدار n2 = 20 n1 = 10 را نمایش می‌دهد، دستور پنجم، مقدار n1 را به ۲۰۰ تغییر می‌دهد، زیرا:

n1 *= n2 ⇨ n1 = n1 * n2 = 10 *20 = 200

دستور ششم، مقدار n2 = 20 n1= 200 را نمایش می‌دهد، دستور هفتم، مقدار n1 را به همان ۱۰ برمی‌گرداند، زیرا:

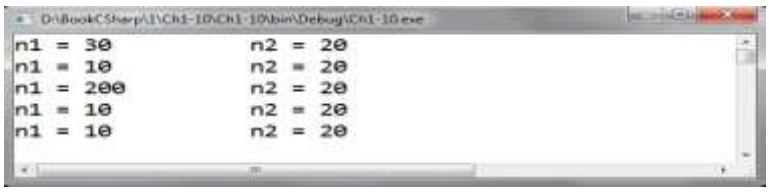
n1 /= n2 ⇨ n1 = n1 / n2 = 200 / 20 = 10

دستور هشتم، مقدار n2 = 20 n1= 10 را نمایش می‌دهد، دستور نهم، مقدار n1 را تغییر نمی‌دهد، زیرا:

n1 % = n2; ⇨ n1 = n1 % n2 = 10 % 20 = 10

دستور نهم، مقدار n2 = 20 n1= 10 را نمایش می‌دهد.

۳. پروژه را ذخیره و اجرا کنید تا خروجی را به شکل زیر مشاهده نمایید:



۴ – ۱۵ – ۱. عملگرهای منطقی

عملگرهای منطقی، بر روی عبارات منطقی درست یا نادرست عمل می‌کنند. نتیجه عملگرهای منطقی در جدول ۸– ۱آمده است. همان‌طور که در جدول ۸ – ۱ می‌بینید، هنگامی نتیجه عملگر && (**و** منطقی) درست است که هر دو عملوند نتیجه درست داشته باشند. اما نتیجه عملگر || (**یا** منطقی) هنگامی نادرست است که هر دو عملوند نادرست باشند.

جدول ۷– ۱ عملگرهای ترکیبی.

عملگر	روش استفاده	مثال	نتیجه	عملکرد
+=	x += y;	x = 3; x += 5;	8	x = x + 5;
-=	x -= y;	x = 7; x -= 3;	4	x = x - 3;
*=	x *= y;	x = 3; x *= 5;	15	x = x * 5;
/=	x /= y;	x = 17; x /= 5;	3	x = x / 5;
%=	x %= y;	x = 17; x %= 5;	2	x = x % 5;

جدول ۸ – ۱ عملکرد عملگرهای منطقی.

X	y	x && y	x || y	!x	!y
False	false	false	false	true	true
True	false	false	true	false	true
False	true	false	true	true	false
True	true	true	true	false	false

مثال ۱۱ – ۱. برنامه‌ای که عملکرد عملگرهای منطقی را نشان می‌دهد.

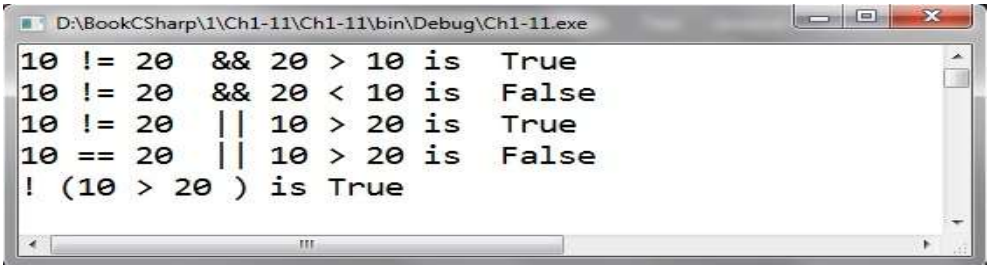
۱. پروژه جدیدی به نام Ch1-11 ایجاد کنید.

۲. به کلاس Program بروید و دستورات آن را به‌صورت زیر تغییر دهید:

```
using System;
namespace Ch1_11
{
    class Program
    {
        static void Main(string[] args)
        {
            int n1 = 10, n2 = 20;
            Console.WriteLine("{0} != {1}  && {1} > {0} is  {2}",
               n1, n2, n1 != n2 && n2 > n1);
            Console.WriteLine("{0} != {1}  && {1} < {0} is  {2}",
               n1, n2, n1 != n2 && n2 < n1);
            Console.WriteLine("{0} != {1}  || {0} > {1} is  {2}",
               n1, n2, n1 != n2 || n1 > n2);
            Console.WriteLine("{0} == {1}  || {0} > {1} is  {2}",
               n1, n2, n1 == n2 || n1 > n2);
            Console.WriteLine("! ({0} > {1} ) is {2}", n1, n2,
               !(n1 > n2) );
            Console.ReadKey();
        }
    }
}
```

دستور اول متد ()Main متغیرهای n1 و n2 را با نوع int تعریف کرده، مقادیر ۱۰ و ۲۰ را به آن‌ها تخصیص می‌دهد، دستور دوم، عبارت 10 != 20 && 20 > 10 is True را نمایش می‌دهد (چون 10 != 20 است و 10 < 20 می‌باشد، سپس نتیجه عملگر && برابر True است). دستور سوم، عبارت 10 != 20 && 20 < 10 is False را نمایش می‌دهد (چون ۲۰ کوچک‌تر از ۱۰ نیست)، پس نتیجه && نادرست (یعنی False) می‌باشد. دستور چهارم، عبارت 10 != 20 || 10 > 20 is True را نمایش می‌دهد (چون ۱۰ مخالف ۲۰ است) پس نتیجه || برابر True خواهد بود). دستور پنجم، عبارت 10 == 20 || 10 > 20 is False را نمایش می‌دهد (چون هر دو عبارت 10 == 20 و 10 > 20 نادرست هستند، نتیجه عبارت || نادرست می‌باشد و دستور ششم، عبارت ! (10 > 20) is True را نمایش می‌دهد (چون ۱۰ بزرگ‌تر از ۲۰ نیست، پس ! (10 > 20) درست (True) می‌باشد.

۳. پروژه را ذخیره و اجرا کنید تا خروجی را به شکل زیر مشاهده نمایید:



۵ – ۱۵ – ۱. عملگرهای خاص

علاوه بر عملگرهای بیان‌شده، برخی از عملگرها در C# کاربرد خاصی دارند. این عملگرها را در زیر می‌بینید:

۱. عملگر؟: برای بررسی شرط خاصی به کار می‌رود و به‌صورت زیر استفاده می‌شود:

; عبارت ۲ : عبارت ۱ ? (شرط)

در این ساختار ابتدا شرط ارزیابی می‌شود (شرط می‌تواند یک شرط ساده یا ترکیبی باشد)، اگر نتیجه ارزیابی شرط درست باشد، **عبارت۱** انجام می‌شود، وگرنه، **عبارت۲** انجام خواهد شد (مثال زیر را ببینید).

مثال ۱۲ – ۱. برنامه‌ای که کاربرد عملگر ؟ را نشان می‌دهد:

۱. پروژه جدیدی به نام Ch1-12 ایجاد کنید.

۲. به کلاس Program بروید و دستورات آن را به‌صورت زیر تغییر دهید:

```
using System;
namespace Ch1_12
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 10, y = 15;
            int z = (x > y || y < 17) ? ++x : ++y;
            Console.WriteLine("X = {0}\tY = {1}\tZ = {2}", x,
                 y, z);
            Console.ReadKey();
        }
    }
}
```

دستور اول، x و y را با نوع int تعریف کرده، مقادیر ۱۰ و ۱۵ را به ترتیب به x و y تخصیص می‌دهد. دستور دوم، ابتدا نتیجه عبارت شرطی (x > y || y <17) را بررسی می‌کند که درست می‌باشد. زیرا، ۱۵ > ۱۰ نیست، اما ۱۷ > ۱۵ است. بنابراین، false || true برابر true می‌باشد. پس، عبارت ۱ (یعنی، ++x) انجام خواهد شد. یعنی، ابتدا به x یک واحد اضافه می‌شود (مقدار x برابر با ۱۱ خواهد شد) و مقدار ۱۱ در z قرار می‌گیرد. دستور سوم، عبارت Z = 11 Y = 15 X = 11 را نمایش می‌دهد.

۳. پروژه را ذخیره و اجرا کنید تا خروجی را به شکل زیر مشاهده نمایید:

عملگر sizeof: تعداد بایت‌هایی که یک نوع اشغال می‌کند را مشخص می‌کند و به‌صورت مقابل به کار می‌رود:

sizeof (نوع داده) ;

مثال ۱۳ – ۱. برنامه‌ای که کاربرد عملگر sizeof را نشان می‌دهد:

۱. پروژه جدیدی به نام Ch1-13 ایجاد کنید.

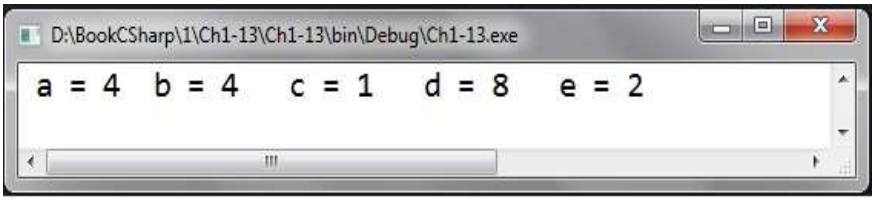
۲. به کلاس Program بروید و دستورات آن را به‌صورت زیر تغییر دهید:

```
using System;
namespace Ch1_13
{
    class Program
    {
        static void Main(string[] args)
        {
            int a, b, c, d, e;
            a = sizeof(int);
            b = sizeof(float);
            c = sizeof(bool);
            d = sizeof(double);
            e = sizeof(char);
            Console.WriteLine(" a = {0}\tb = {1}\tc = {2}\td =
              {3}\te = {4}", a, b, c, d, e);
            Console.ReadKey();
        }
    }
}
```

دستور اول متد Main() متغیرهای a، b، c، d و e را با نوع int تعریف می‌کنند، دستور دوم، اندازه نوع int (یعنی ۴) را در a قرار می‌دهد، دستور سوم، اندازه نوع float (یعنی ۴) را در b قرار می‌دهد، دستور چهارم، اندازه نوع bool (یعنی ۱) را در c قرار می‌دهد، دستور پنجم، اندازه نوع double (یعنی ۸) را در d قرار می‌دهد، دستور ششم، اندازه نوع char (یعنی ۲) را در e قرار می‌دهد و دستور هفتم، مقادیر متغیرها را به‌صورت زیر نمایش می‌دهد:

a = 4 b = 4 c= 1 d = 8 e = 2

۳. پروژه را ذخیره و اجرا کنید تا خروجی را به شکل زیر مشاهده نمایید:



عملگر (): برای تغییر اولویت‌های عملگرها در عبارت به کار می‌رود.

انتساب می‌تواند چندگانه باشد. در این صورت انتساب به‌صورت زیر انجام خواهد شد:

متغیر ۱ = متغیر ۲ = ... = متغیر n = مقدار;

به‌عنوان‌مثال، دستورات زیر a، b و n را با نوع int تعریف کرده، مقدار ۱۰ را به آن‌ها تخصیص می‌دهند:

```
int  a, b, n;
a = b = n = 10;
```

مثال ۱۴ – ۲. برنامه‌ای که عملکرد انتساب چندگانه را نشان می‌دهد.

۱. پروژه جدیدی به نام Ch1-14 ایجاد کنید.

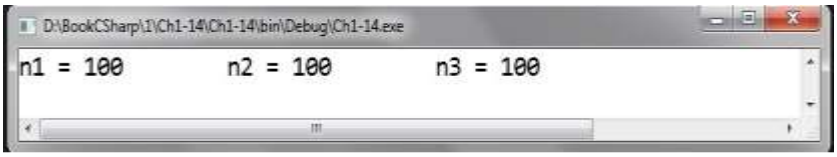
۲. به کلاس Program بروید و دستورات آن را به‌صورت زیر تغییر دهید:

```
using System;
namespace Ch1_14
{
    class Program
    {
        static void Main(string[] args)
        {
            int n1, n2, n3;
            n1 = n2 = n3 = 100;
            Console.WriteLine("n1 = {0}\tn2 = {1}\tn3 = {2}",
              n1, n2, n3);
            Console.ReadKey();
         }
    }
}
```

دستور اول متغیرهای n1، n2 و n3 را با نوع int تعریف می‌کند، دستور دوم به هر سه متغیر n1، n2 و n3 مقدار ۱۰۰ (با انتساب چندگانه) را تخصیص می‌دهد و دستور سوم، مقادیر n1، n2 و n3 را به‌صورت زیر نمایش می‌دهد

: n1= 100 n2 = 100 n3 = 100

۳. پروژه را ذخیره و اجرا کنید تا خروجی را به شکل زیر مشاهده نمایید:



۲.عملگر !: عملگر یکانی است که قبل از یک عبارت یا مقدار منطقی قرارگرفته و نتیجه عبارت را برعکس می‌نماید. یعنی، اگر مقدار عبارت true باشد، false را برمی‌گرداند و اگر مقدار عبارت false باشد، true را برگشت خواهد داد.

۳.عملگرهای بیتی، عملگرهایی هستند که عملیات بیتی را بر روی عملوندها انجام می‌دهند(جدول ۹-۱). این عملگرها عبارت‌اند از:

عملگر ~، قبل یک عدد صحیح قرارگرفته و بیت‌های آن را برعکس می‌نماید. یعنی، بیت‌های یک را به صفر و بیت‌های صفر آن را به یک تبدیل می‌کند. به‌عنوان‌مثال، دستورات زیر را ببینید:

جدول ۹ – ۱ عملکرد عملگرهای بیتی.

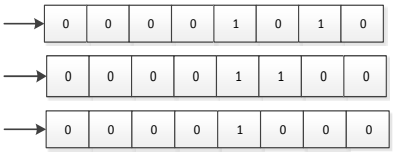
~Y	~X	X ^ Y	X | Y	X & Y	Y	X
1	1	0	0	0	0	0
1	0	1	1	0	0	1
0	1	1	1	0	1	0
0	0	0	1	1	1	1

```
byte x = 12;
byte y = ~x;
```

دستور اول، x را برابر 00001100 باینری قرار می‌دهد و دستور دوم، y را برابر 11110011 (نقیض x) قرار می‌دهد.

عملگر &، بین دو عدد قرارگرفته، آن‌ها را بیت به بیت "**و بیتی**" می‌کند. یعنی، اگر هر دو بیت یک باشند، نتیجه بیت متناظر یک خواهد شد، وگرنه، نتیجه بیت متناظر صفر می‌گردد. به‌عنوان‌مثال، دستورات زیر را ببینید:

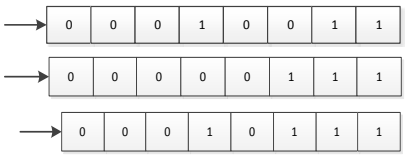
```
byte x = 10;
byte y = 12;
byte z = x & y;
```



دستور اول، x را با نوع بایت تعریف کرده، مقدار 00001010 (معادل ۱۰) را در آن قرار می‌دهد، دستور دوم y را با نوع byte تعریف نموده، مقدار 00001100 را در آن قرار می‌دهد و دستور سوم، z را با نوع byte تعریف کرده، مقدار 00001000 (نتیجه و بیتی x و y) را در آن قرار می‌دهد.

عملگر |، بین دو عملوند عددی قرارگرفته آن‌ها را بیت به بیت "**یا بیتی**" می‌نماید. نتیجه بیتی "یا بیتی" زمانی صفر است که هر دو بیت صفر باشند، وگرنه، نتیجه بیتی آن یک است. به‌عنوان‌مثال، دستورات زیر را مشاهده کنید:

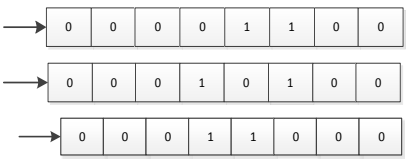
```
byte x = 19;
byte y = 7;
byte z = x | y;
```



دستور اول x را با نوع بایت تعریف کرده، مقدار ۱۹ را در آن قرار می‌دهد، دستور دوم، y را با نوع بایت تعریف کرده، مقدار 7 را در آن قرار می‌دهد و دستور سوم، z را با نوع بایت تعریف کرده، مقدار 19 | 7 (یعنی ۲۳) را در آن قرار می‌دهد (مقادیر بیتی را در بالا می‌بینید).

عملگر ^، بین دو عملوند قرار می‌گیرد و آن‌ها (به‌صورت بیت به بیت)، "**یا انحصاری**" می‌کند. یعنی زمانی نتیجه "**یا انحصاری بیتی**" یک است که دقیقاً یکی از بیت‌ها یک باشد، وگرنه نتیجه آن صفر خواهد شد. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
byte x = 12;
byte y = 20;
byte z=x ^ y;
```



دستور اول x را با نوع بایت تعریف کرده، مقدار 00001100 (معادل ۱۲) را در آن قرار می‌دهد، دستور دوم y را با نوع byte تعریف کرده، مقدار 00010100 (معادل ۲۰) را در آن قرار می‌دهد و دستور سوم، z را با نوع بایت تعریف کرده، مقدار 20 ^ 12 (یعنی 00011000 معادل ۲۴) را در آن قرار می‌دهد.

مثال ۱۵ – ۲. برنامه‌ای که عملکرد عملگرهای !، &&، ||، ~، &، |، ^ را نشان می‌دهد.

۱. پروژه جدیدی به نام Ch1-15 ایجاد کنید.

۲. به کلاس Program بروید و دستورات آن را به‌صورت زیر تغییر دهید:

```
using System;
namespace Ch1_15
{
    class Program
    {
        static void Main(string[] args)
        {
            bool b1 = true, b2 = false;
            byte n1 = 10, n2 = 7;
            Console.WriteLine("{0}\t &&\t{1} = {2}", b1, b2,
               b1 && b2);
            Console.WriteLine("{0}\t ||\t{1} = {2}", b1, b2,
               b1 || b2);
            Console.WriteLine("!{0} = {1}\t !{2} = {3}", b1,
               !b1, b2, !b2);
            Console.WriteLine("{0}\t& \t{1} = {2}", n1, n2,
               n1 & n2);
            Console.WriteLine("{0}\t| \t{1} = {2}", n1, n2,
               n1 | n2);
            Console.WriteLine("{0}\t^ \t {1} = {2}", n1, n2,
               n1 ^ n2);
            Console.WriteLine("~{0} = {1}\t ~ {2} = {3}", n1,
               ~n1, n2, ~n2);
            Console.ReadKey();
        }
    }
}
```

دستور اول، متغیرهای b1 و b2 را با نوع bool (منطقی) و مقادیر true و false تعریف می‌کند، دستور دوم، متغیرهای n1 و n2 را با نوع بایت با مقادیر ۱۰ و ۷ تعریف می‌کند، دستور سوم، عبارت True && False = False را نمایش می‌دهد (چون b1 برابر true، b2 برابر false و نتیجه b1 && b2 برابر false است). دستور چهارم، True = False || True را نمایش می‌دهد (چون b1 برابر True، b2 برابر false و نتیجه b1 || b2 برابر true می‌باشد). دستور پنجم، !false = True! !True = False! را نمایش می‌دهد. دستور ششم، عبارت 10 & 7 = 2 را نمایش می‌دهد. زیرا:

```
   10 = 00001010
    7 = 00000110
10 & 7 = 00000010 = 2
```

دستور هفتم، عبارت 10 | 7 = 15 چون:

```
   10 = 00001010
    7 = 00000111
10 | 7 = 00001111 = 15
```

دستور هشتم، عبارت 13 = 7 ^ 10 را نمایش می‌دهد، چون:

```
   10 = 00001010
    7 = 00000111
10 & 7 = 00001101 = 13
```

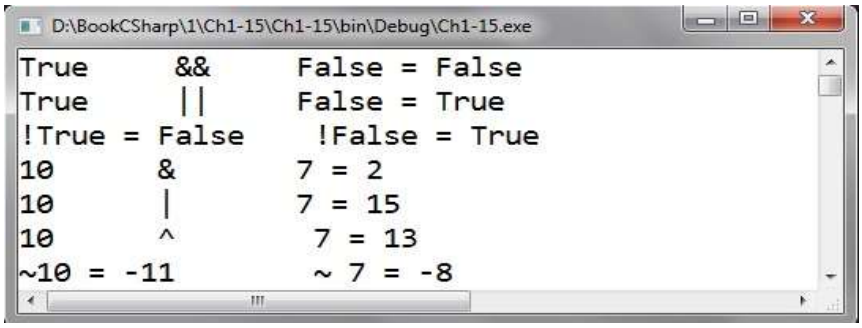
دستور نهم، مقدار -8 = 7~ -11 = 10~ چون

```
   10 = 00001010
~10 = 11110101 = -11
```

و

```
    7 = 00000110
~ 7 = 11111000 = -8
```

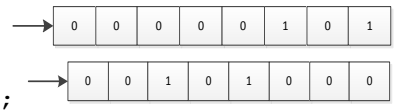
۳. پروژه را ذخیره و اجرا کنید تا خروجی را به شکل زیر مشاهده نمایید:



عملگر >>، بعد از یک عملوند قرارگرفته و آن را به تعداد مشخصی بیت شیفت به چپ می‌دهد.

به‌عنوان‌مثال، دستور زیر را ببینید:

```
byte x = 5;
byte y=x << 3;
```

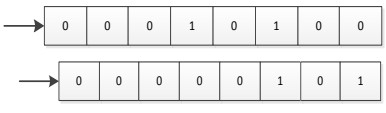


دستور اول، x را با نوع بایت تعریف کرده، مقدار 00000101 (معادل ۵) را در آن قرار می‌دهد، دستور دوم، x را سه بیت به سمت چپ شیفت داده، در y قرار می‌دهد (یعنی 00101000 (معادل ۴۰) را در y قرار می‌دهد. همان‌طور که مشاهده کردید، با هر شیفت به سمت چپ عدد در ۲ ضرب می‌شود. پس با سه شیفت به چپ عدد ۵ در $۲^۳$ (یعنی ۸) ضرب گردید. این قانون تا زمانی برقرار است که هیچ مقدار ۱ (بیت یک) از سمت چپ خارج نشود.

عملگر <<، بعد از یک عملوند قرارگرفته و آن را به تعداد مشخص‌شده به سمت راست شیفت می‌دهد.

دستورات زیر را در نظر بگیرید:

```
byte x = 20;
byte y = x >> 2;
```



دستور اول، x را با نوع بایت تعریف کرده، مقدار 00010100 (معادل ۲۰) را در آن قرار می‌دهد، دستور دوم، y را با نوع بایت تعریف کرده، سپس مقدار x را دو بیت به سمت راست شیفت می‌دهد، یعنی مقدار 0000101 (معادل ۵) را در آن قرار می‌دهد. همان‌طور که مشاهده می‌شود، با هر شیفت به راست، عدد بر ۲ تقسیم می‌گردد.

مثال ۱۶ – ۲. برنامه‌ای که عملکرد عملگرهای <<، >> را نشان می‌دهد.

۱. پروژه جدیدی به نام Ch1-16 ایجاد کنید.

۲. به کلاس Program بروید و دستورات آن را به‌صورت زیر تغییر دهید:

```
using System;
namespace Ch1_16
{
    class Program
    {
        static void Main(string[] args)
        {
            byte n1 = 50, n2 = 20;
            Console.WriteLine("{0} >> 2  = {1}", n1, n1 >> 2);
            Console.WriteLine("{0} <<  4  = {1}", n2, n2 << 4);
            Console.WriteLine("{0} bit(4)   = {1}",n1,n1>>4 &1);
            Console.ReadKey();
        }
    }
}
```

دستور اول، متغیرهای n1 و n2 را با نوع int به ترتیب مقادیر ۵۰ و ۲۰ تعریف می‌کند، دستور دوم، عبارت 12= 2 << 50 را نمایش می‌دهد (چون ۵۰ را دو بار به سمت راست شیفت می‌دهد، پس بر $۲^۲$ (یعنی ۴) تقسیم می‌گردد که نتیجه ۱۲ است. این عمل به‌صورت زیر انجام می‌شود:

50 = 00110010
50 >> 2 = 00001100 ⇨ 2

دستور سوم، عبارت 320 = 4 >> 20 را نمایش می‌دهد، چون، ۲۰ را ۴ بار به سمت چپ شیفت می‌دهد، پس هر بار شیفت به چپ عدد را در ۲ ضرب می‌کند، لذا، عدد در $۲^۴$ (یعنی در ۱۶ ضرب می‌شود که نتیجه 320 (20 * 16) خواهد بود:

20 = 00010100
20 << 4 = 101000000 =320

دستور چهارم، بیت چهارم n1 را برمی‌گرداند که عبارت مقابل را نمایش می‌دهد: 50 bit(4) = 1
یعنی 00110010 = 50 بیت چهارم آن ۱ است.

۳. پروژه را ذخیره و اجرا کنید تا خروجی را به شکل زیر مشاهده نمایید:

عملگر ??، یک عملگر شرطی است که دارای ساختار زیر می‌باشد:

متغیر ۱ = متغیر ۲ ?? مقدار ;

چنانچه متغیر ۲ برابر null باشد، **مقدار** را در متغیر ۱ قرار می‌دهد، وگرنه، مقدار متغیر ۲ را در متغیر ۱ قرار می‌دهد. در این ساختار متغیرهای ۱ و ۲ باید به‌صورت شرطی تعریف شوند.

مثال ۱۷ – ۱. برنامه‌ای که عملکرد عملگر ?? را نشان می‌دهد.

۱. پروژه جدیدی به نام Ch1-17 ایجاد کنید.

۲. به کلاس Program بروید و دستورات آن را به‌صورت زیر تغییر دهید:

```
using System;
namespace Ch1_17
{
    class Program
    {
        static void Main(string[] args)
        {
            int? x = null;
            int? y = 12;
            int m, n;
            m = x ?? 0;
            n = y ?? 0;
            Console.WriteLine("M = {0}\tN = {1}", m, n);
            Console.ReadKey();
        }
    }
}
```

خطوط ۱ و ۲، متغیرهای x و y را به‌صورت شرطی با نوع int (int?) تعریف می‌کنند. دستور سوم، m و n را با نوع int تعریف می‌نماید. دستور چهارم، چون مقدار x برابر null است، مقدار صفر (یعنی مقدار دوم) را در m قرار می‌دهد. اما دستور پنجم، چون مقدار y برابر ۱۲ است (null نمی‌باشد)، مقدار y یعنی ۱۲ را در n قرار می‌دهد. دستور ششم، مقادیر m و n را نمایش می‌دهند.

۳. پروژه را ذخیره و اجرا کنید تا خروجی را به شکل زیر مشاهده نمایید: