
برنامه‌سازی پیشرفته با زبان C#

تألیف

دکتر همایون موتمنی
دکتر جواد وحیدی
دکتر رمضان عباس‌نژادورزی



فن آوری نوین

سرشناسه	: موتمنی، همایون، ۱۳۵۰ -
عنوان و نام پدیدآور	: برنامه‌سازی پیشرفته با زبان C# /تالیف همایون موتمنی، جواد وحیدی، رمضان عباس‌نژادورزی.
مشخصات نشر	: بابل: فناوری نوین، ۱۳۹۶.
مشخصات ظاهری	: ۲۸۸ ص.: مصور، جدول .
شابک	: ۲۸۰۰۰۰ ریال 978-600-7272-17-6
وضعیت فهرست نویسی	: فیپا
موضوع	: سی شارپ (زبان برنامه‌نویسی کامپیوتر)
موضوع	: C # (Computer program language)
موضوع	: برنامه‌نویسی شی‌گرا
موضوع	: Object-oriented programming (Computer science)
شماره افزوده	: وحیدی، جواد، ۱۳۴۸ -
شماره افزوده	: عباس‌نژاد ورزی، رمضان، ۱۳۴۸ -
رده بندی کنگره	: QA۷۶/۷۳ ۱۳۹۶ م۷۷/۹۵ س/
رده بندی دیویی	: ۰۰۵/۱۳۳
شماره کتابشناسی ملی	: ۵۱۲۹۹۱۵



فن‌آوری نوین

www.fanavarienovin.net

بابل، صندوق پستی ۷۳۴۴۸-۴۷۱۶۷

تلفن: ۰۱۱-۳۲۲۵۶۶۸۷

برنامه‌سازی پیشرفته با زبان C#

تألیف: دکتر همایون موتمنی - دکتر جواد وحیدی - دکتر رمضان عباس‌نژادورزی

ناشر: فن‌آوری نوین

چاپ اول: زمستان ۹۶

جلد: ۱۰۰۰

شابک: ۹۷۸ - ۶۰۰ - ۷۲۷۲ - ۱۷ - ۶

حروف چینی و صفحه‌آرایی: فن‌آوری نوین

قیمت: ۲۸۰۰۰ تومان

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

فهرست مطالب

فصل اول: آشنایی با زبان C#	۹
۱-۱. فرآیند برنامه‌نویسی در دات‌نت	۹
۱-۲. مجموعه کتابخانه کلاس دات‌نت Framework	۱۰
۱-۳. فضای نام	۱۱
۱-۴. آموزش زبان‌های برنامه‌نویسی	۱۲
۱-۵. شناسه‌ها	۱۳
۱-۶. کلمات کلیدی	۱۴
۱-۷. فضای سفید	۱۵
۱-۸. لیترال‌ها	۱۵
۱-۹. توضیحات	۱۶
۱-۱۰. کاراکترهای ویژه (Punctuators)	۱۷
۱-۱۱. انواع داده	۱۸
۱-۱۲. انواع مقدار	۱۸
۱-۱۳. انواع ارجاع	۲۰
۱-۱۴. ثابت‌ها	۳۳
۱-۱۵. عملگرها	۳۳
۱-۱۵-۱. عملگرهای محاسباتی	۳۴
۱-۱۵-۲. عملگرهای رابطه‌ای (مقایسه‌ای)	۳۷
۱-۱۵-۳. عملگرهای ترکیبی	۳۸
۱-۱۵-۴. عملگرهای منطقی	۳۹
۱-۱۵-۵. عملگرهای خاص	۴۱
۱-۱۶. اولویت عملگر	۴۹
۱-۱۷. تبدیل نوع	۵۱
۱-۱۸. تمرین	۶۷
فصل دوم: ساختارهای کنترلی	۷۲
۲-۱. ساختارهای تصمیم‌گیری	۷۲
۲-۱-۱. ساختار تصمیم if	۷۲
۲-۱-۲. ساختار if تودرتو	۷۷
۲-۱-۳. ساختار switch	۸۰

۸۳	۲-۲. ساختارهای تکرار.....
۸۳	۲-۲-۱. ساختار تکرار for.....
۸۹	۲-۲-۲. دستور break.....
۹۰	۲-۲-۳. دستور continue.....
۹۰	۲-۲-۴. ساختار while.....
۹۳	۲-۲-۵. ساختار تکرار do while.....
۱۰۸	۲-۳. تمرین‌ها.....

فصل سوم: متدها و پیاده‌سازی آن‌ها ۱۱۵

۱۱۵	۳-۱. انواع متدها.....
۱۱۶	۳-۱-۱. متدهای کتابخانه‌ای ۲.....
۱۱۸	۳-۱-۲. متدهایی که برنامه‌نویس می‌نویسد.....
۱۲۲	۳-۲. ارسال پارامترها به متدها.....
۱۲۲	۳-۲-۱. ارسال پارامتر از طریق مقدار.....
۱۲۴	۳-۲-۲. ارسال پارامتر از طریق ارجاع.....
۱۲۷	۳-۳. متدهای بازگشتی.....
۱۳۰	۳-۴. متدهای همنام.....
۱۳۳	۳-۵. تعریف آرگومان‌های اختیاری با مقدار پیش فرض.....
۱۳۵	۳-۶. تعریف متدی با تعداد پارامتر نامعلوم.....
۱۴۳	۳-۷. تمرین.....

فصل چهارم: آرایه‌ها و رشته‌ها ۱۴۹

۱۵۰	۴-۱. تعریف آرایه‌های یک‌بعدی.....
۱۵۰	۴-۲. مقداردهی عناصر آرایه.....
۱۵۱	۴-۲-۱. مقداردهی به خانه‌های آرایه به صورت مجزا.....
۱۵۱	۴-۲-۲. مقداردهی اولیه به عناصر آرایه در هنگام تعریف آن.....
۱۵۱	۴-۲-۳. مقداردهی به خانه‌های آرایه با حلقه‌های تکرار و دستورات ورودی.....
۱۵۲	۴-۳. نمایش مقادیر آرایه.....
۱۵۲	۴-۳-۱. نمایش مقادیر هر عنصر به صورت مجزا.....
۱۵۲	۴-۳-۲. نمایش مقادیر آرایه با حلقه‌های تکرار while for و do while.....
۱۵۲	۴-۳-۳. نمایش عناصر آرایه با حلقه foreach.....
۱۵۵	۴-۴. تولید اعداد تصادفی.....
۱۵۶	۴-۵. ارسال آرایه‌ها به متدها.....
۱۵۶	۴-۵-۱. ارسال عناصر آرایه به متدها.....
۱۵۸	۴-۵-۲. ارسال نام آرایه‌ها به متدها.....
۱۶۰	۴-۶. مرتب‌سازی آرایه.....

۱۶۴	۴-۷. جستجوی مقادیر آرایه.....
۱۶۴	۴-۷-۱. جستجوی خطی (ترتیبی).....
۱۶۶	۴-۷-۲. جستجوی دودویی در آرایه مرتب شده.....
۱۶۸	۴-۸. حذف عناصر آرایه.....
۱۶۸	۴-۹. درج عنصری بین عناصر آرایه.....
۱۷۲	۴-۱۰. ارسال آرایه از طریق پارامتر نوع params.....
۱۷۳	۴-۱۱. آرایه‌های دوبعدی.....
۱۷۴	۴-۱۱-۱. تعریف آرایه دوبعدی (مستطیلی).....
۱۷۵	۴-۱۱-۲. مقداردهی عناصر آرایه دوبعدی.....
۱۷۶	۴-۱۱-۳. نمایش مقادیر آرایه دوبعدی.....
۱۸۰	۴-۱۲. آرایه‌های دندانه‌ای.....
۱۸۲	۴-۱۳. معرفی آرایه‌ای از اشیا.....
۱۸۳	۴-۱۴. رشته‌ها.....
۱۸۴	۴-۱۵. متدهایی برای دست‌کاری رشته.....
۱۸۸	۴-۱۶. تمرین.....

فصل پنجم: برنامه‌نویسی مبتنی بر شیء: کلاس‌ها..... ۱۹۵

۱۹۵	۵-۱. کلاس‌ها.....
۱۹۶	۵-۱-۱. تعریف کلاس‌ها.....
۱۹۸	۵-۱-۲. نمونه‌سازی کلاس‌ها.....
۱۹۹	۵-۲. اعضای کلاس.....
۱۹۹	۵-۲-۱. دسترسی به اعضای کلاس.....
۲۰۰	۵-۲-۲. انواع اعضای کلاس.....
۲۰۸	۵-۳. مقداردهی اولیه به اعضای کلاس با متد سازنده.....
۲۱۵	۵-۴. اعضای static.....
۲۱۷	۵-۵. متدهای static.....
۲۲۰	۵-۶. ارجاع this.....
۲۲۳	۵-۷. اعضای فقط خواندنی (read only).....
۲۲۵	۵-۸. ایندکسرها (Indexer).....
۲۲۸	۵-۹. Delegate.....
۲۳۵	۵-۱۰. نمایش متغیرها و متدها با گزینه ClassView Diagram.....
۲۳۵	۵-۱۱. تمرین.....

فصل ششم: برنامه‌نویسی شیء‌گرا: وراثت، چندریختی و واسط‌ها..... ۲۳۸

۲۳۸	۶-۱. وراثت.....
-----	-----------------

۲۴۰	۶-۲. کلاس مشتق چه اعضای از کلاس پایه را به ارث می‌برد.....
۲۴۱	۶-۳. تعریف کلاس مشتق.....
۲۴۲	۶-۴. پایه تمام کلاس.....
۲۴۲	۶-۵. سازنده‌ها و مخرب‌ها در کلاس‌های مشتق.....
۲۴۶	۶-۶. متدهای مجازی.....
۲۴۷	۶-۷. پنهان نمودن اعضای کلاس پایه.....
۲۵۱	۶-۸. اعضای انتزاعی.....
۲۵۲	۶-۸-۱. کلاس‌های انتزاعی.....
۲۵۷	۶-۹. کلاس‌ها و متدهای sealed.....
۲۵۸	۶-۱۰. کلاس Static.....
۲۶۲	۶-۱۱. واسط‌ها.....
۲۶۸	۶-۱۲. تعریف مجدد عملگرها.....
۲۸۲	۶-۱۳. تمرین.....
۲۸۴	منابع:

مقدمه

زبان C# در فناوری دات نت (NET) توسط مایکروسافت ارائه شده است که کاملاً شیء‌گرا است. امروزه اکثر دانشجویان رشته کامپیوتر با این زبان آشنایی دارند. برنامه‌های متعددی از قبیل تحت کنسول، دسک‌تاپ، بانک اطلاعاتی، طراحی صفحات وب، WPF، WCF، تحت شبکه و دستگاه‌های موبایل را می‌توانید با زبان C# بنویسید این کتاب تأکید بیشتری به نوشتن برنامه‌های تحت کنسول نموده است.

از طرف دیگر، زبان C# به‌عنوان سرفصل درس برنامه‌سازی پیشرفته در رشته‌ها کامپیوتر، فناوری اطلاعات، ICT و علوم کامپیوتر تدریس می‌شود.

در حال حاضر کتاب‌های زیادی برای زبان برنامه‌نویسی C# ارائه شده است که جای تقدیر و تشکر دارد. هر یک از این کتاب‌ها نوع خاص از زبان برنامه‌نویسی C# را مورد بررسی قرار می‌دهند. اما، این کتاب تمرکز بیشتری روی برنامه‌های تحت کنسول دارد. کتاب حاضر با بیان مسائل متعدد تحت کنسول و حل آن‌ها، دانشجویان محترم را با زبان برنامه‌نویسی C# آشنا می‌کند.

کتاب به‌صورت گام‌به‌گام با جملات کوتاه و ساده بیان گردیده است. برنامه‌های متن کتاب را می‌توانید از سایت انتشارات فن‌آوری نوین به آدرس www.fanavarienovin.net دریافت نمایید.

در پایان امیدوارم این اثر مورد توجه جامعه انفورماتیک کشور، اساتید و دانشجویان عزیز قرار گیرد.

مؤلفین

fanavarienovin@gmail.com

آشنایی با زبان C#

پیشرفت‌های زبان‌های برنامه‌نویسی نظیر ++C و جاوا، موجب ایجاد مشکلات و نیازمندی‌های جدیدی گردید. ایجاد یکپارچگی اجزای نرم‌افزاری از زبان‌های مختلف برنامه‌نویسی مشکل بود و در نصب این ابزار مشکلات مشترکی وجود داشت. به همین دلیل بود که نسخه جدید قطعات^۱ 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 برنامه‌نویسان را قادر می‌سازد، به راحتی 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 با قابلیت برنامه‌نویسی ویژوال، امکان ایجاد برنامه‌هایی با استفاده از محیط توسعه مجتمع (IDE)^۲ را تأمین کرده است. با استفاده از IDE، برنامه‌نویس می‌تواند به راحتی 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)^۳ 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 اختیاران می‌گذارند. اما، در دات‌نت زبان‌های مختلف به یکدیگر شبیه شده‌اند و امکانات یکسانی را در اختیاران قرار می‌دهند. چون این زبان‌ها در هنگام اجرا از زبان مشترک زمان اجرا (CLR)^۴ استفاده می‌کنند. بنابراین، در زمان اجرا این مورد که در طراحی برنامه از چه زبانی استفاده شده است، تفاوتی ندارد. از آنجایی که زبان‌های مختلف گرامرهای متفاوتی دارند، بنابراین باید زبانی

^۱. Components

^۳.RAD(Rapid Application Development)

^۲. IDE(Integrated Development Environment)

^۴. 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 استودیو دات‌نت از کامپایلرها و زبان‌های مختلفی تشکیل شده است که عبارت‌اند از:

۱. ویژوال بیسیک

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 به عنوان محیط زمان اجرای نهایی مورد استفاده قرار داده‌اند. برخی از این زبان‌ها عبارت‌اند از: Smalltalk, RPG, Python, Perl, Mercury, ML, Fortran, Cobol, APL, و غیره.

گام سوم، نوشتن کد مورد نیاز 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)^۱ وجود دارد. این بخش شامل هزاران کلاس می‌باشد که هر کدام 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^۲ هستند. این نوع برنامه‌ها، از طریق سرویس دهنده بانک اطلاعاتی یا چندین وب‌سرویس، اطلاعات مورد نیازشان را دریافت کرده، پردازش‌های مورد نیاز را بر روی آن انجام داده، صفحات مبتنی بر HTML ایجاد می‌نمایند تا این صفحات از طریق مرورگرهای وب^۳ در کامپیوتر سرویس گیرنده^۴ قابل نمایش باشند.

^۱.Framework Class Library ^۲. Hyper Text Markup Language ^۳. Browser ^۴.Client
^۵.Windows Service Control Manager

۴. سرویس‌های ویندوز، سرویس‌هایی را می‌توان در دات‌نت ایجاد کرد که توسط مدیر کنترل سرویس ویندوز (SCM)^۵ و نیز دات‌نت Framework قابل کنترل هستند. معمولاً این سرویس‌ها برای تبادل اطلاعات بین برنامه‌های مختلف استفاده می‌شوند.

۵. وب‌سرویس‌ها، سرویس‌ها یا توابعی هستند که به راحتی از طریق شبکه وب قابل دسترسی و فراخوانی هستند.

۶. قطعات و کتابخانه کلاس، در دات‌نت Framework می‌توان قطعات^۱ و کتابخانه‌هایی با کلاس‌های جدید ایجاد نمود. این قطعات و کتابخانه‌های کلاس‌های جدید را به راحتی 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^۲ 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 نکنند. 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 به راحتی 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 وقتی که برنامه‌ای از نوع Console Application ایجاد می‌نمایید. اگر برنامه‌هایی با نوع‌های دیگر به برنامه اضافه کنید، ممکن است فضاهای نام دیگر به پروژه تان اضافه شود). علاوه بر فضاهای نامی که به‌طور خودکار به برنامه

¹.Components ².Namespace

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

```
using نام فضای نام;
```

به عنوان مثال، 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 (به صورت 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 همانام 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:

- مانند هر زبان طبیعی ابتدا باید علائم تشکیل دهنده زبان را آموخت. به عنوان مثال، زبان فارسی از علائم الف تا ی، ارقام ۰ تا ۹ و علائم خاص مانند !، :، ؟ و غیره تشکیل شده است. هر کدام 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 تشکیل شده است. ابتدا باید مفاهیم هر یک از این علائم را در زبان C# آموخت.
- 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. به عنوان مثال، کلمات **بابا**، **آب**، **داد**، در زبان فارسی مفهوم خاصی دارند. ولی کلمات **تیتانم** و **بکیاپ** مفهوم خاصی ندارند. به کلماتی که در زبان دارای مفهوم خاص هستند، **کلمات کلیدی** می‌گویند. در زبان C# کلمات کلیدی نظیر **while**، **else if**، **for** و **int** وجود دارند. در آموزش این زبان ابتدا باید کلمات کلیدی را شناخت. معنی و کاربرد هر کدام از آن‌ها را باید آموخت.

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

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. به‌عنوان مثال، برای تعریف داده‌های نوع صحیح به کار می‌رود و به‌صورت زیر استفاده می‌گردد:

متغیر n, ... , متغیر ۲, متغیر ۱ int

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^۱، نام‌هایی هستند که برنامه‌نویس به عناصر C# از قبیل کلاس‌ها^۲، فضاهای نام^۳، متدها^۴، فیلدها^۵، خواص^۶ و غیره انتخاب می‌کند. به‌عنوان مثال، (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 ۱-۲). شناسه‌ها در C# با رنگ سبز مشخص می‌گردند.

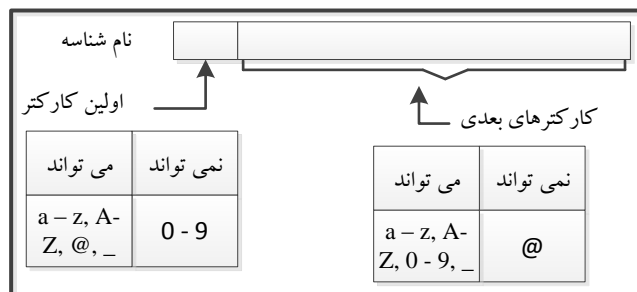
کاراکترهای الفبایی (A تا Z، a تا z) و خط ربط (-) می‌توانند هر مکان نام شناسه قرار گیرند.

^۱. برای کسب اطلاعات بیشتر در زمینه LINQ به کتاب آموزش گام‌به‌گام LINQ تألیف رمضان عباس نژاد ورزی انتشارات فناوری نوین مراجعه فرمایید.

^۲. Identifiers ^۳. Namespaces ^۴. Methods ^۵. Fields ^۶. 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 ۱-۲ روش نام‌گذاری شناسه‌ها.

✚ ارقام صفر تا ۹ نمی‌توانند در اولین مکان نام شناسه قرار گیرند، ولی می‌توانند در مکان‌های دیگر نام شناسه مورد استفاده 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 شناخته شده‌اند و مفهوم خاصی در آن زبان دارند، کلمات کلیدی^۱ 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).

^۱.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 ۱-۳ برخی از کلمات کلیدی در برنامه C#.

جدول ۱-۲ 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	unchecked
break	delegate	float	protected	lock	string	unsafe
byte	do	for	public	long	struct	ushort
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، داده‌هایی هستند که به صورت 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^۱ 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. به عنوان مثال، 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.

^۱.Comments

آشنایی با زبان C# ۱۷

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. به‌عنوان مثال، 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 چند سطری را ایجاد می‌کنند.

در C# نمی‌توانید توضیحات تودرتو تعریف کنید. به‌عنوان مثال، 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

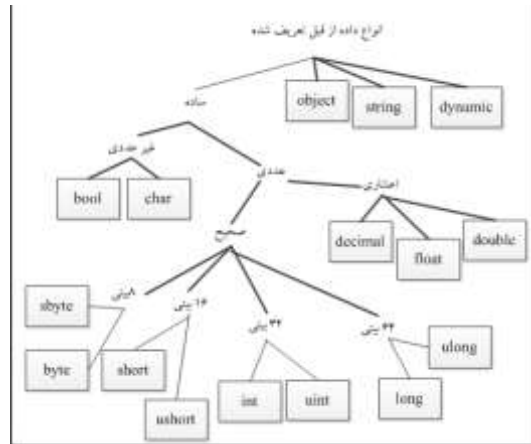
این کاراکترها برای نگهداری گروهی از اجزا یا جداکننده‌ها به کار می‌روند. برخی از این کاراکترها را در (شکل ۶-۱) می‌بینید. در این شکل، کاراکتر ؛ انتهای دستورات C# را مشخص می‌کند. کاراکتر {، بلاک C# را باز می‌کند و کاراکتر }، بلاک باز شده C# را می‌بندد. هر بلاک در C# با کاراکتر { شروع و با کاراکتر } خاتمه 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.



شکل ۷ - 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. به‌عنوان مثال، دستور زیر مقدار ۲۵ را به متغیری به نام *i* نسبت می‌دهد:

```
int i = 25;
```

مقدار	نام	نوع
25	i	Int

این عمل به شکل مقابل انجام می‌شود:

وقتی متغیر نوع مقدار ذخیره می‌شود، C# نوشته‌ای از نوع، نام

شناسه و مقدار را نگهداری می‌کند و هنگامی که متغیر نوع مقدار را کپی می‌کنید، متغیر دوم جداگانه‌ای با همان نوع و مقدار ایجاد می‌شود. به‌عنوان مثال، دستور زیر کپی *i* را در *j* نشان می‌دهد:

```
int j = i;
```

مقدار	نام	نوع
25	i	int
25	j	int

این عمل به شکل مقابل انجام می‌گردد:

این متغیرها ارتباطی باهم ندارند. اگر مقدار متغیری عوض

شود، متغیر دیگر مقدار قبلی‌اش را نگهداری می‌کند. به‌عنوان مثال،

```
i = 50;
```

دستور مقابل را ببینید:

مقدار	نام	نوع
50	i	int
25	j	int

این دستور، مقدار *i* را به ۵۰ تغییر می‌دهد، اما، مقدار *j* همان ۲۵

باقی خواهد ماند (شکل مقابل را مشاهده کنید):

تمام انواع تعریف شده در جدول ۴ - ۱ (به‌جز نوع object و

string) متغیرهای از نوع مقدار را تعریف می‌کنند.

جدول ۳ - ۱ انواع داده‌های C#				
مقدار پیش فرض	معادل .NET	محدوده	هدف	نام
0	System.SByte	-128....127	عدد صحیح ۸ بیتی با علامت	sbyte
0	System.Byte2550	عدد صحیح ۸ بیتی بدون علامت	byte
0	System.Int16	-32768...32767	عدد صحیح ۱۶ بیتی با علامت	short
0	System.UInt16	0....65535	عدد صحیح ۱۶ بیتی بدون علامت	ushort
0	System.Int32	-2147483648... 2147483647	عدد صحیح ۳۲ بیتی با علامت	Int
0	System.UInt32	0....4294964295	عدد صحیح ۳۲ بیتی بدون علامت	UInt
0	System.Int64	- 922337203685477 5808... 922337203685477 807	عدد صحیح ۶۴ بیتی با علامت	Long
0	System.UInt64	0... 18446744073 709551615	عدد صحیح ۶۴ بیتی بدون علامت	Ulong
0.0f	System.Single	$105 \times 10^{-45} \dots 3.4 \times 10^{38}$	عدد اعشاری با دقت معمولی	Float
0.0d	System.Double	5 $\times 10^{-324} \dots 1.7$ $\times 10^{-308}$	عدد اعشاری با دقت مضاعف	Double
False	System.Boolean	True, False	نوع منطقی	Bool
\x0000	System.Char	U+0000 U+ffff	کاراکتر یونیکد،	Char
0m	System.Decimal	$\pm 1.0 \times 10^{28} \dots$ $\pm 7.9 \times 10^{28}$	عدد ده‌دهی با ۲۵ رقم اعشار	Decima l
Null	System.object	-----	کلاس پایه‌ای که همه انواع دیگر از آن مشتق می‌شوند	Object
Null	System.string	-----	مجموعه‌ای از کاراکترهای یونیکد	String

۱۳ - ۱. انواع ارجاع

انواع ارجاع (Reference Types) به دو شکل یک شیء و ارجاع به آن شیء می‌باشند. به‌عنوان مثال،

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:

نوع	نام	ارجاع به	نوع	مقدار
StringBuilder	Obj1	→	StringBuilder	نوع ارجاع

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.

وقتی که تغییری از نوع ارجاع را کپی می‌کنید، کپی جدیدی از ارجاع ایجاد خواهد شد، اما، یک شیء

جدید نیست. به‌عنوان مثال، دستور زیر را مشاهده کنید:

```
StringBuilder obj2 = obj1;
```

این دستور، بیان می‌کند که `obj1` و `obj2` هر دو شیء به مقدار "نوع ارجاع" اشاره می‌کنند (شکل زیر را

ببینید):

نوع	نام	ارجاع به	نوع	مقدار
StringBuilder	Obj1	→	StringBuilder	"نوع ارجاع"
StringBuilder	Obj2	→	StringBuilder	"نوع ارجاع"

همان‌طور که در این شکل می‌بینید، فقط یک شیء از نوع `StringBuilder` داریم، اما دو ارجاع به نام‌های

`obj1` و `obj2` 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. به‌عنوان مثال، دستور زیر را در

نظر بگیرید:

آشنایی با زبان C# ۲۱

```
StringBuilder obj = null;
```

ارجاع به	نام	نوع
Null	obj	StringBuilder

انواع تهی پذیر

انواع تهی پذیر^۱، اجازه می دهند تا متغیری از نوع مقدار ایجاد کنید که می تواند یک داده معتبر و غیر معتبر را بپذیرد. بنابراین می توانید قبل از استفاده از متغیر به معتبر بودن آن مطمئن شوید. یک نوع تهی پذیر همیشه بر پایه نوع دیگر است. ایجاد نوع تهی پذیر به صورت زیر است:

نام متغیر ؟ نوع پایه

به عنوان مثال، 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 می دهد.

اکنون با عملگر `!=` می توان 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 می کند. اگر *i* برابر تهی باشد، در `True`، وگرنه `False` در *b* قرار می گیرد.

به سه طریق می توان به متغیر از نوع تهی پذیر مقدار داد که عبارت اند از:

۱. مقداردهی در هنگام تعریف

۲. مقداردهی از طریق یک متغیر تهی پذیر دیگر

۳. مقدار `null`

به عنوان مثال، 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 نام های `myI1`، `myI2`، `myI3` از نوع تهی پذیر تعریف می کند، دستور دوم،

مقدار ۲۸ را در `myI1` قرار می دهد، دستور سوم، مقدار متغیر تهی پذیر `myI1` را در متغیر تهی پذیر `myI2`

قرار می دهد و دستور چهارم، مقدار (`null`) را در متغیر `myI3` قرار می دهد.

دستورات

دستورات در زبان C# دو نوع اند:

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

به عنوان مثال، دستور زیر را مشاهده کنید:

¹.Nullable Types

```
int i = 5;
```

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.

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

به‌عنوان مثال، دستورات مقابل را ببینید:

این دستورات تشکیل بلاک را می‌دهند.

ساختار برنامه C#

برای این که 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.

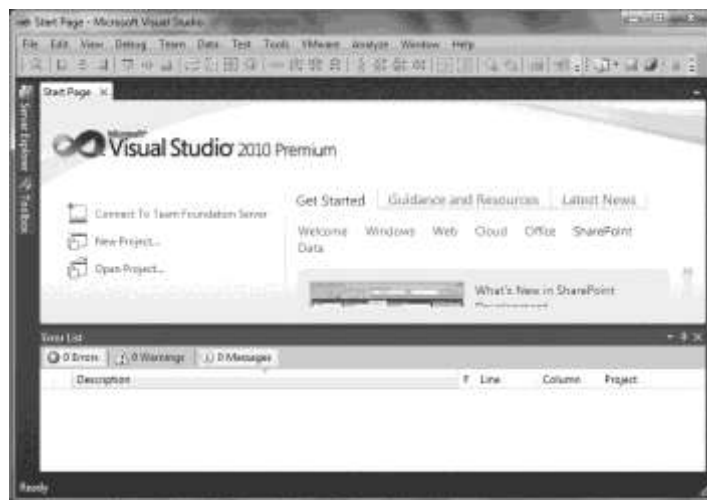
مثال ۱-۱ برنامه‌ای که مراحل ایجاد و اجرای یک برنامه در C# را نشان می‌دهد (هدف این برنامه آشنایی با ایجاد و اجرای برنامه در C# است).

مراحل طراحی و اجرا

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

Start/All Programs/Microsoft Visual Studio 2017/Microsoft Visual Studio 2017

۲. اکنون صفحه اول ویژوال استودیو ظاهر می‌شود (شکل ۸-۱).



شکل ۸-۱ صفحه اول ویژوال استودیو.

۳. گزینه `File/New/Project` (کلیدهای `Ctrl+Shift+N`) را اجرا کنید یا آیکون `New Project...` را کلیک نمایید. در هر صورت، پنجره `New Project` ظاهر می‌شود (شکل ۹-۱). در این پنجره اطلاعات زیر را انتخاب کنید:

آشنایی با زبان C# ۲۳

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 Application را انتخاب کنید.

در جلوی Name، نام پروژه را وارد کنید. در این برنامه Ch1_1 انتخاب شده 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 به صورت 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^۱ را ایجاد کرد. با مفهوم کلاس در فصل‌های ۵ و ۶ بیشتر تر 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 به صورت زیر تعریف شده است:

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

همان‌طور که در این دستور مشاهده می‌کنید، متد Main() با کلمه کلیدی static تعریف شده است. این کلمه Modifier نام دارد. کلمه کلیدی static بیان می‌کند که این متد فقط در همین کلاس قابل اجرا می‌باشد و از طریق هیچ نمونه^۲ دیگری قابل اجرا نمی‌باشد (در فصل‌های ۵ و ۶ با متدهای static بیشتر تر 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 تر 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() به صورت 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.


^۱.Interface

^۲.Instance

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.

در هر صورت پروژه اجرا شده، 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 صفحه کلید 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 بک اسلش (Backslash) را تعیین می‌کند.
'\0'	0x0000	کاراکتر تهی (null) را تعیین می‌کند.
'\a'	0x0007	بوق سیستم را به صدا در می‌آورد.
'\b'	0x0008	کاراکتر Backspace 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 به صورت 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 می تواند شامل علامت گذاری های جایگزینی^۱ باشد. یک علامت گذار جایگزینی، مکانی را در رشته فرمت، علامت گذاری خواهد کرد که از یک مقدار عددی که در داخل `{}` قرار دارد، تشکیل می شود. در رشته فرمت می توانید از کاراکترهای کنترلی استفاده کنید، این کاراکترها در جدول ۴ - ۱ آمده اند.

۲. مقادیر `subVal0` , `subVal1` و ...، مقادیری هستند که باید جایگزین علامت گذاری های جایگزینی در خروجی شوند. این مقادیر جایگزینی شماره دارند که از صفر (۰) شروع می شود. یعنی، اولین مقدار دارای شماره صفر، دومین مقدار شماره یک و این روند ادامه می یابد. به عنوان مثال، دستور زیر را ببینید:

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

این دستور خروجی زیر را نمایش می دهد:

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

همان طور که در این خروجی مشاهده می گردد، 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 می دهد. به عنوان مثال، دستورات زیر را ببینید:

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

```
One Two
```

این دستورات خروجی مقابل را نمایش می دهند:

اکنون دستورات زیر را ببینید:

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

این دستورات خروجی زیر را نمایش می دهند:

```
One
Two
```

متد `ReadKey()`، برای ایجاد مکث (توقف موقت) در برنامه به کار می رود و به صورت زیر استفاده

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

می شود:

وقتی کنترل اجرای برنامه به این متد برسد، اجرای برنامه به طور موقت قطع خواهد شد تا کاربر کاراکتری را فشار دهد. به محض این که کاربر کاراکتری را فشار داد، اجرای برنامه ادامه می یابد. این متد معمولاً برای ایجاد توقف به کار می رود تا کاربر خروجی برنامه را ببیند.

مثال ۲-۱. برنامه ای که عبارت `First output in c#` را نمایش می دهد.

مراحل طراحی و اجرا

¹.Substitution

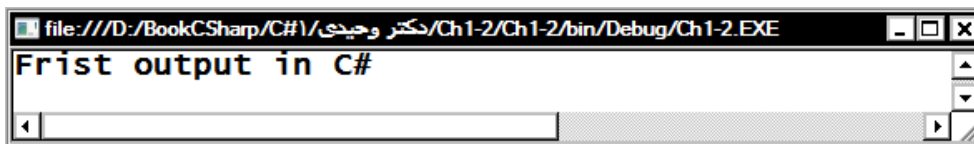
۱. پروژه جدیدی به نام Ch1-2 ایجاد کنید، برای این منظور، گزینه File/New/Project را اجرا نمایید تا پنجره New Project ظاهر شود. در این پنجره، نوع برنامه را Console Application و نام آن را Ch1-2 انتخاب کرده، دکمه OK را کلیک کنید.

۲. به کلاس Program بروید و دستورات آن را به صورت 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 Ch# First output in را نمایش می دهد 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 می باشد که برای خوانایی کتاب آن را به سفید تغییر دادیم.

مثال ۳-۱. برنامه ای که کاربرد کاراکترهای کنترلی، جا نگره دار، متدهای Write و WriteLine را نمایش می دهد.

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 به صورت زیر تغییر دهید:

```
using System;
namespace Ch1_3
{
    class Program
```

```

    {
        static void Main(string[] args)
        {
            Console.WriteLine("One\t{0}\nTwo\t{1}\t", 1, 2);
            Console.WriteLine("Fanavarienovin\t{0}\nKetabrah\t{1}\\",
                "www.fanavarienovin.net", "www.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 به جای 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 به جای 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 استفاده شده است). دستور سوم، ابتدا کلمه Three را نمایش داده و کنترل چاپ را به تب بعد انتقال می‌دهد، سپس مقدار ۳ را به جای جا نگه‌دار {0} نمایش می‌دهد و کنترل چاپ را به تب بعدی انتقال داده، Four را نمایش می‌دهد، در پایان، کنترل چاپ را به تب بعدی انتقال داده، به جای نگه‌دار {1} مقدار ۴ را نمایش می‌دهد و بالاخره ' (کاراکتر \ کتیشن) را نمایش می‌دهد و به سطر بعدی می‌رود.

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

```

file:///D:/BookCSharp/C#1/ویدیو/Ch1-3/Ch1-3/Min/Debug/Ch1-3 EXE
One      1
Two      2      Fanavarienovin  www.fanavarienovin.net
Ketabrah www.ketabrah.ir\
Three    3
Four     4'

```

متغیر

برای نگه‌داری هر چیزی لازم است که از یک ظرف متناسب با آن استفاده نمود. به عنوان مثال، در خانه برای نگه‌داری مواد غذایی، ظروف مختلفی وجود دارند که هر کدام 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 یک لحظه فقط یک مقدار را دارد.

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

۱. نام‌گذاری متغیرها

بعد از این 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، به‌عنوان مثال، اجازه نمی‌دهد نام بچه را رضا^۱ 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: sum، area، sum1، pr_1 و ... اما، نام‌های زیر برای متغیر مجاز نیستند:

✚ نام test2: 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^۱ 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 به صورت زیر می‌باشد:

; لیست متغیرها نوع داده‌ای

نوع داده‌ای، یکی از انواع داده بیان شده 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، f₁ و f₂ را با نوع float و ch را با نوع کاراکتر تعریف می‌کند.

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

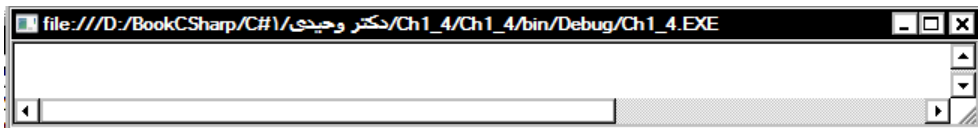
^۱ blank(space).

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

```
using System;
namespace Ch1_4
{
    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 تعریف شده‌اند. اما، مقداردهی و استفاده نشده‌اند.

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



۳. 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 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 به صورت زیر تغییر دهید:

```
using System;
namespace Ch1_5
{
    class Program
    {
        static void Main(string[] args)
        {
```

آشنایی با زبان C# ۳۱

```

    {
        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 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 به جای 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 به جای جا نگه‌دار {1} نمایش می‌دهد و دستور ششم، ابتدا عبارت PI= را نمایش داده، مقدار متغیر PI را به جای 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 به جای جا نگه‌دار {1} نمایش می‌دهد و دستور هفتم، ابتدا عبارت PI= را نمایش داده، مقدار متغیر PI را به جای جا 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 به جای جا 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 به جای 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:

```

file:///D:/BookCSharp/C#/ویدیو/Ch1-5/Ch1-5/bin/Debug/Ch1-5.EXE
x = 10 y = 12
PI = 3.14 ch = F
yes = True

```

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.

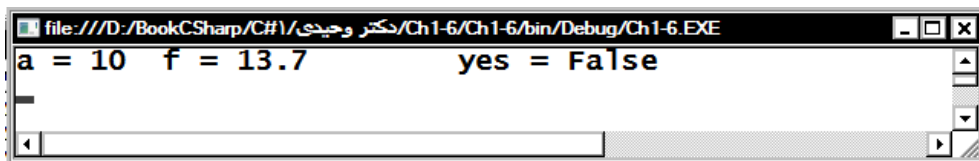
۱. پروژه جدیدی به نام Ch1-6 ایجاد کنید.

۲. به کلاس Program بروید و دستورات آن را به صورت 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 می کنند، دستورات چهارم تا ششم، به ترتیب به متغیرهای a، f و yes مقادیر ۱۰، 13.7f و false تخصیص می دهند و دستور هفتم، ابتدا عبارت a= را نمایش داده، به جای جا نگه دار {0} مقدار ۱۰ (مقدار a) را نمایش می دهد، با کاراکتر '\t' به تب بعدی می رود و عبارت f= را نمایش داده، جلوی آن به جای جا نگه دار {1} مقدار 13.7 (مقدار متغیر f) را نمایش می دهد و در پایان عبارت yes= را نمایش داده و روبروی آن مقدار false (مقدار متغیر yes) را به جای جا نگه دار {2} نمایش می دهد و کنترل چاپ را به سطر بعدی انتقال می دهد.

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



نکته: وقتی مقدار جدیدی در متغیر قرار می گیرد، این مقدار جایگزین مقدار قبلی می شود. یعنی، مقدار قبلی از دست می رود (حذف می گردد).

۱۴-۱. ثابت ها

ثابت شناسه ای (نام خانه ای از حافظه) است که مقدار آن در طول اجرای برنامه تغییر نمی کند. ثابت ها انواع مختلف دارند. ثابت ها می توانند عددی صحیح، اعشاری، کاراکتری، رشته ای یا منطقی باشند. ثابت های

آشنایی با زبان C# ۳۳

کاراکتری بین تک کتیشن (!) قرار می گیرند (مانند 'C')، ثابت های رشته ای بین جفت کتیشن قرار می گیرند (نظیر "C#") و ثابت های منطقی مقادیر true یا false هستند. به عنوان مثال، مقدار 3.1415، (عدد π) را مشخص می کند. این عدد یا هر ثابت دیگر ممکن است چندین مرتبه در برنامه استفاده شود. به دلیل راحتی اصلاح و تغییر مقدار ثابت ها به آن ها نام تخصیص می دهند. تعریف ثابت در C# به صورت های زیر انجام می شود:

مقدار ثابت = نام ثابت نوع ثابت 1.const

به عنوان مثال، دستور مقابل را ببینید:

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

این دستور ثابت PI را با مقدار 3.1415 تعریف

```
PI = 3.141505;
```

می کند. اکنون دستور مقابل را ببینید.

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

جدول ۵-۱ عملگرهای محاسباتی.				
عملگر	نام عملگر	مثال	نتیجه	توضیحات
+	جمع	$12 + 3$	۱۵	عملوند اول را با عملوند دوم جمع می کند.
-	تفریق	$13,5 - 3$	۱۰,۵	عملوند دوم را از عملوند اول کم می کند.
*	ضرب	$12 * 2,5$	۳۰	عملوند اول را در عملوند دوم ضرب می کند.
/	تقسیم	$13 / 2$	۶	عملوند اول را بر عملوند دوم تقسیم می کند.
%	باقی مانده تقسیم صحیح	$13 \% 5$	۳	باقی مانده تقسیم صحیح عملوند اول بر عملوند دوم را محاسبه می کند.
++	افزایش	$x = 10;$ $x++;$	۱۱	یک واحد به عملوند اضافه می کند.
--	کاهش	$x = 10;$ $x--;$	۹	یک واحد از عملوند کم می نماید.

۱۵-۱. عملگرها

عملگرها، نمادهایی هستند که اعمال خاصی را بر روی داده انجام می دهند. عملگرها انواع مختلف دارند


که برخی از آن ها عبارت اند از:

۱. عملگرهای محاسباتی
۲. عملگرهای رابطه ای (مقایسه ای)
۳. عملگرهای ترکیبی
۴. عملگرهای منطقی
۵. عملگرهای خاص

¹.operators

۱-۱۵-۱. عملگرهای محاسباتی

این عملگرها برای انجام محاسبات بر روی داده‌های عددی به کار می‌روند (جدول ۵-۱). از جمله 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 به صورت 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 پنجم، 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:

```
file:///D:/BookCSharp/C#/۱/ویدیو/ch1-7/ch1-7/bin/Debug/ch1-7.EXE
10 + 3 = 13
10 - 3 = 7
10 * 3 = 30
10 / 3 = 3
10 % 3 = 1
```

عملگر ++ یک واحد به محتویات عملوند اضافه می کند. 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 می کند. سپس، عبارت ارزیابی می گردد (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 می گردد (به مثال زیر دقت کنید).

مثال ۸-۱. برنامه‌ای که عملکرد عملگرهای ++ و -- را بیان می کند.

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

۲. به کلاس Program بروید و دستورات آن را به صورت 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();
        }
    }
}
```

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

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

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 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;
```

دستور ششم، عبارت، $n2 = 20$ $n1 = 10$ را نمایش می‌دهد، دستور هفتم، عبارت $n3 = 10$ $n4 =$


19 را نمایش خواهد داد و دستور هشتم، عبارت $n6 = 20$ $n5 = 9$ را چاپ می‌نماید.

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

```
file:///D:/BookCSharp/C#/1/دکتر وحیدی/Ch1-8/Ch1-8/bin/Debug/Ch1-8.EXE
n1 = 10      n2 = 20
n3 = 10      n4 = 19
n5 = 9       n6 = 20
```

۲-۱۵-۱. عملگرهای رابطه‌ای (مقایسه‌ای)

این عملگرها برای مقایسه دو عملوند به کار می‌روند و نتیجه درست یا نادرست را برمی‌گرداند. 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 به صورت 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، عبارت 10 != 20 is True را نمایش می‌دهد.

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

```
file:///D:/BookCSharp/C#1/دکتر وحیدی/Ch1-9/Ch1-9/bin/Debug/Ch1-9.EXE
10 == 20 is False
10 > 20 is False
10 >= 20 is False
10 < 20 is True
10 <= 20 is True
10 != 20 is True
```

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 به صورت 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:

$$n1 += n2 \Rightarrow n1 = n1 + n2 = 10 + 20$$

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

$$n1 -= n2 \Rightarrow n1 = n1 - n2 = 30 - 20 = 10$$

آشنایی با زبان C# ۳۹

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

$$n1 *= n2 \Rightarrow n1 = n1 * n2 = 10 * 20 = 200$$

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

$$n1 /= n2 \Rightarrow n1 = n1 / n2 = 200 / 20 = 10$$

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

$$n1 \% = n2; \Rightarrow n1 = n1 \% n2 = 10 \% 20 = 10$$

دستور نهم، مقدار $n2 = 20$ را نمایش می‌دهد.

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

```
file:///D:/BookCSharp/C#/دکتر وحیدی/Ch1-10/Ch1-10/bin/Debug/Ch1-10.EXE
n1 = 30      n2 = 20
n1 = 10      n2 = 20
n1 = 200     n2 = 20
n1 = 10      n2 = 20
n1 = 10      n2 = 20
```

۴-۱۵-۱. عملگرهای منطقی

عملگرهای منطقی، بر روی عبارات منطقی درست یا نادرست عمل می‌کنند. نتیجه عملگرهای منطقی در جدول ۸-۱ آمده 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.

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;

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.