فصل 2

مروری بر برنامه نویسی C#

* نمونه ای از برنامه C#
* شناسه ها
* کلمات کلیدی
* Main : نقطه شروع یک برنامه
* فضای خالی
* دستورات
* متن خروجی از یک برنامه
* توضیحات : یادداشت کد نویسی

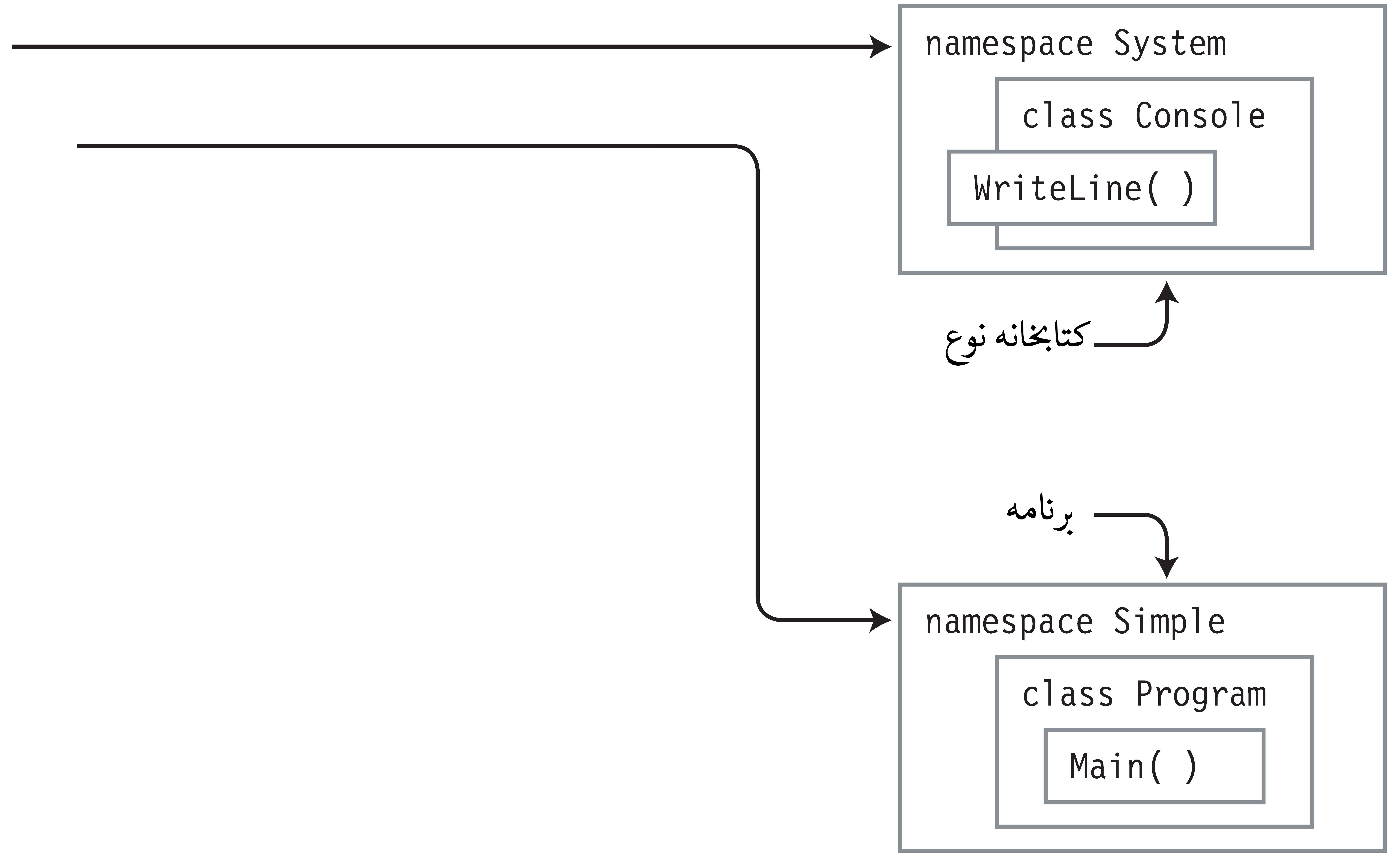
نمونه ای از برنامه C#

در این فصل به مفاهیم پایه ای برای یادگیری C# می پردازیم. ما در سراسر متن از نمونه کد های فراوانی استفاده خواهیم کرد. ابتدا نشان خواهیم داد که یک برنامه C# چگونه به نظر می رسد و بخش های مختلف آن چه معنی می دهند.

ما با یک برنامه ساده شروع می کنیم و تمام اجزای آن را توضیح می دهیم . در این فصل طیف وسیعی از موضوعات را، از ساخت یک برنامه C# گرفته تا تولید خروجی برنامه به صفحه نمایش، معرفی می کنیم.

با این مقدمات از طریق کد منبع ، ما می توانیم آزادانه از نمونه کد ها در تمام متن استفاده کنیم. بنابراین ، برخلاف فصل های بعدی، در این فصل یک یا دو موضوع به طور مفصل بررسی می شود، و بسیاری از موضوعات را به طور مختصر رسیدگی می کند.

بیایید ابتدا به یک نمونه ساده از برنامه C# نگاهی بیندازیم. منبع برنامه به طور کامل در منطقه بالا و به صورت سایه دار در شکل 1-2 نشان داده شده است، این کد در یک فایل متنی به نام SimpleProgram.cs قرار دارد. اگر تمام کد را خوانده اید، در مورد فهمیدن همه جزئیات آن نگران نباشید. در جدول 1-2 توضیحات خط به خط کد آورده شده است. خروجی برنامه در منطقه پایین و به صورت سایه دار در شکل آورده شده است. بخش سمت راست تصویری گرافیکی از بخش های این برنامه را نشان می دهد.

* هنگامی که کد کامپایل و اجرا شود، رشته "Hi there!" در یک پنجره بر روی صفحه نمایش، نشان داده می شود.
* خط 5 شامل دو کاراکتر اسلش است. این کاراکتر ها و همه چیزهای بعد از آنها روی این خط ، توسط کامپایلر نادیده گرفته می شود. این کاراکتر های اسلش توضیحات یک خطی[[1]](#footnote-1) نامیده می شود.

|  |  |
| --- | --- |
| SimpleProgram.cs | |
| using System;  namespace Simple  {  Class Program // تعریف کلاس  {  Static void Main()  {  Console.WriteLine("Hi there!");  }  }  } | 1  2  3  4  5  6  7  8  9  10  11  12 |
| این کد خروجی زیر را تولید می کند: |  |
| Hi there! |  |

شکل 1-2. برنامه SimpleProgram

جدول 1-2. خط به خط برنامه SimpleProgram

|  |  |
| --- | --- |
| **شماره خط** | **توضیحات** |
| خط 1 | به کامپایلر می گوید که این برنامه از فضای نام System استفاده می کند. |
| خط 3 | یک فضای نام جدید به نام Simple را تعریف می کند.   * فضای نام جدید با آکولاد باز در خط 4 شروع می شود و تا آکولاد بسته در خط 12 ادامه دارد. * هر نوع تعریف شده در این بخش، عضو هایی از فضای نام هستند. |
| خط 5 | یک نوع کلاس جدید به نام Program تعریف می کند.   * هر عضو تعریف شده ما بین جفت آکولاد ها در خط های 6 و 11 عضو هایی هستند که این کلاس را تشکیل می دهند. |
| خط 7 | یک متد به نام Main که به عنوان یک عضو از کلاس Program است را تعریف می کند.   * در این برنامه ، Main تنها عضوی از کلاس Program است. * Main یک تابع خاص است که کامپایلر آن را به عنوان نقطه شروع برنامه فراخوانی می کند. |
| خط 9 | تنها شامل یک دستور ساده است ، این خط به منزله بدنه Main است.   * دستورات با یک نقطه و ویرگول خاتمه پیدا می کنند. * این دستور از یک کلاس به نام Console در فضای نام System استفاده می کند تا در پنجره صفحه نمایش پیغامی را چاپ کند. * بدون دستور using در خط 1، کامپایلر نمی فهمد که کجا را باید برای کلاس Console جستجو کند. |

توضیحات بیشتر درباره SimpleProgram

یک برنامه C# شامل یک یا چند نوع تعریف است. بخش عمده ای از این کتاب صرف توضیح انواع مختلفی است که شما می توانید ایجاد و در برنامه خود از آن استفاده کنید. نوع ها در یک برنامه می تواند به هر ترتیبی تعریف شده باشد .در مثال SimpleProgram ، تنها یک نوع کلاس تعریف شده است .

فضای نام[[2]](#footnote-2) مجموعه ای از نوع های تعریف شده است که با نام آن مرتبط است. SimpleProgram از دو فضای نام استفاده کرده است. فضای نام جدیدی به نام Simple ایجاد می کند که در آن کلاس Program تعریف شده است و دیگری فضای نام System است که در آن کلاس Console تعریف شده است.

برای کامپایل برنامه ، شما می توانید از ویژوال استودیو[[3]](#footnote-3) و یا از کامپایلر خط فرمان[[4]](#footnote-4) استفاده کنید. برای استفاده از کامپایلر خط فرمان، در ساده ترین شکل خود، دستور زیر را در پنجره Command استفاده کنید.

|  |
| --- |
| csc SimpleProgram.cs |

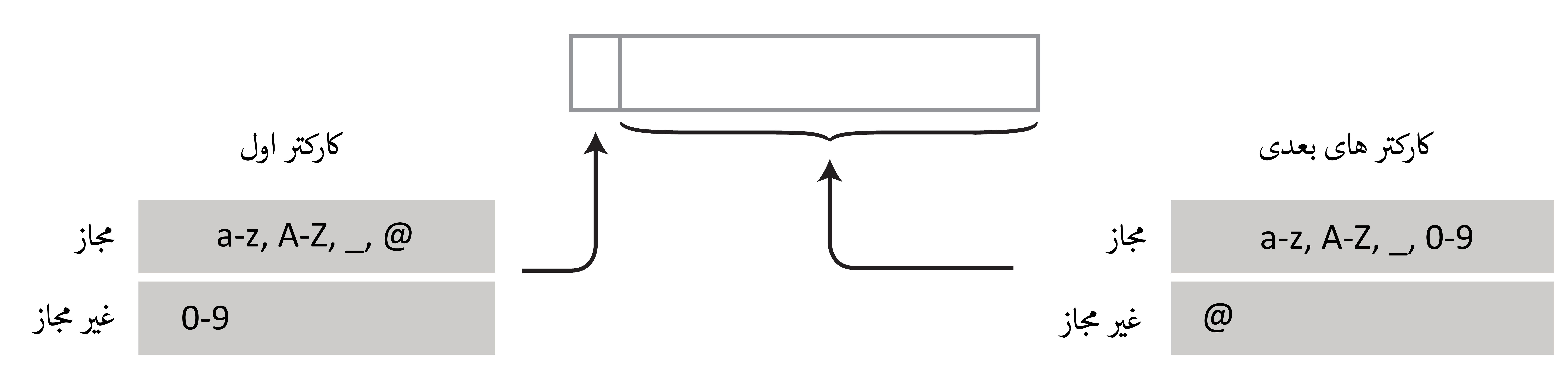
در این فرمان، csc نام کامپایلر خط فرمان و SimpleProgram.cs نام فایل منبع است. csc مخفف عبارت "C-Sharp Compiler" است.

شناسه ها

شناسه ها[[5]](#footnote-5)، رشته های کاراکتری هستند که در نام مواردی از قبیل متغییر ها، متد ها، پارامتر ها، و یا نمونه ای از دیگر ساختار های برنامه نویسی که بعداً آنها را پوشش می دهیم استفاده می شوند.

می توان از طریق الحاق کلمات معنا دار به یک صفت و استفاده از حروف بزرگ و کوچک ( به عنوان مثال CardDeck، PlayersHand، FirstName، SocialSecurityNum ) شناسه های معتبری ایجاد کرد. شکل 2-2 قواعد استفاده از کاراکتر ها را برای شناسه ها نشان می دهد بعضی کاراکتر ها را نمی توان در موقعیت های مشخصی استفاده کنیم.

* کاراکتر های الفبا و زیر خط ( از ابتدا a تا انتهای z ، از ابتدای A تا انتهای Z و \_ ) در هر موقعیتی مجاز هستند.
* اعداد در مکان ابتدا مجاز نیستند اما هر جای دیگر مجاز هستند.
* کاراکتر @ در ابتدا مجاز است اما هر جای دیگر مجاز نیست . اگر چه استفاده از کاراکتر @ اجازه داده شده است ، ولی برای استفاده همگان توصیه نمی شود.



شکل 2-2. کاراکتر های مجاز در شناسه ها

شناسه ها به حروف بزرگ و کوچک حساس هستند. به عنوان مثال نام متغییر myVar و MyVar شناسه های مختلفی هستند.

به عنوان مثال ، تکه کد زیر، همه متغییر ها متفاوت و از نوع integer و معتبر هستند. اما استفاده از نام مشابه برنامه نویسی را بیشتر مستعد خطا می کند و اشکال زدایی برنامه مشکل تر خواهد شد. اینها اشکال زدایی کد شما را در چند لحظه نمایش نمی دهند.

|  |
| --- |
| // طبق قواعد معتبر است، ولی این کار را نکنید!  int totalCycleCount;  int TotalCycleCount;  int TotalcycleCount; |

توصیه می شود از قراردادهای نام گذاری که در فصل 7 آورده شده است، استفاده کنید.

کلمات کلیدی

کلمات کلیدی، رشته کاراکتری هستند که در تعریف زبان C# استفاده شده اند. جدول 3-2 لیست کاملی از کلمات کلیدی C# را ارائه می دهد .

مواردی که مهم است درباره کلمات کلیدی بدانید به شرح زیر است :

* کلمات کلیدی نمی توانند در نام های متغییر ها یا هر شکل دیگری از شناسه استفاده شود، مگر این که با کاراکتر @ آغاز شده باشد.
* همه کلمات کلیدی C# به طور کامل از حروف کوچک تشکیل شده اند.

جدول 3-2. کلمات کلیدی C#

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| typeof | short | out | int | extern | const | abstract |
| uint | sizeof | override | interface | false | continue | as |
| ulong | stackalloc | params | internal | finally | decimal | base |
| unchecked | static | private | is | fixed | default | bool |
| unsafe | string | protected | lock | float | delegate | break |
| ushort | struct | public | long | for | do | Byte |
| using | switch | readonly | namespace | foreach | double | case |
| virtual | this | ref | new | goto | else | catch |
| void | throw | return | null | if | enum | char |
| volatile | true | sbyte | object | implicit | event | checked |
| while | try | sealed | operator | in | explicit | class |

کلمات کلیدی متنی[[6]](#footnote-6) شناسه هایی هستند که در برخی از ساختارهای زبان به عنوان کلمات کلیدی عمل می کند. در آن موقعیت آنها معانی خاص دارند، اما برخلاف کلمات کلیدی، که همیشه نمی توانند به عنوان شناسه استفاده شوند، کلمات کلیدی متنی می توانند به عنوان شناسه ها در بخش های دیگر از کد استفاده شود. جدول 4-2 لیستی از کلمات کلیدی متنی را شامل می شود.

جدول 4-2. کلمات کلیدی متنی C#

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| dynamic | descending | by | await | async | ascending | add |
| into | in | group | global | get | from | equals |
| select | remove | partial | orderby | on | let | join |
|  |  | yield | where | var | value | set |

Main: نقطه شروع یک برنامه

هر برنامه C# باید کلاسی با یک متد Main داشته باشد. در برنامه SimpleProgram که قبلا نشان داده شده است، این متد در کلاس Program تعریف شده بود.

* نقطه شروع اجرای هر برنامه C# اولین دستوری است که در متد Main قرار دارد.
* نام Main باید با حرف بزرگ شروع شود.

ساده ترین شکل از Main به شکل زیر است:

|  |
| --- |
| static void Main( )  {  دستورات  } |

فضای خالی

فضاهای خالی در برنامه به کاراکتر هایی اشاره دارد که نیازی نیست در کاراکتر های خروجی استفاده شوند. فضاهای خالی در کد منبع توسط کامپایلر نادیده گرفته می شوند، اما توسط برنامه نویس استفاده می شود تا کد شفاف تر و راحت تر خوانده شود. برخی از کاراکتر های فضای خالی عبارتند از :

* Space
* Tab
* New line
* Carriage return

به عنوان مثال، تکه کد های زیر به رغم اختلاف هایی که در ظاهر خود دارند دقیقا توسط کامپایلر به یک شکل عمل می کنند.

|  |
| --- |
| // قالب بندی زیبا  Main()  {  Console.WriteLine("Hi, there!");  }  // تنها متن طولانی  Main(){Console.WriteLine("Hi, there!");} |

دستورات

دستورات در C# بسیار شبیه به دستوارت C و C++ است . در این بخش شکل کلی از دستورات معرفی می شود؛ شکل کامل دستورات در فصل 9 پوشش داده شده است.

دستور[[7]](#footnote-7) ، دستورالعملی از کد منبع است که یک نوع را توصیف می کند یا به برنامه گفته می شود که کاری انجام دهد.

* یک دستور ساده با نقطه و ویرگول خاتمه می یابد.

به عنوان مثال ، کد زیر به ترتیب دو دستور ساده است. اولین دستور یک متغییر عدد صحیح با نام var1 و با مقدار 5 تعریف می کند. دستور دوم مقدار متغییر var1 را در پنجره ی صفحه نمایش چاپ می کند.

|  |
| --- |
| int var1 = 5;  System.Console.WriteLine("The value of var1 is {0}", var1); |

بلوک ها

بلوک ، دنباله ای از دستورات (صفر یا بیشتر) است که توسط مجموعه ای از جفت آکولاد ها محصور است ؛ و آن به عنوان یک دستور ساختاری عمل می کند.

همان طور که در کد زیر نشان داده شده است، می توان بلوکی از مجموعه دستورات مثال قبل را، با قرار دادن دستورات در میان آکولاد ها ایجاد کرد:

|  |
| --- |
| {  int var1 = 5;  System.Console.WriteLine("The value of var1 is {0}", var1);  } |

مواردی که مهم است درباره بلوک بدانید به شرح زیر است:

* هنگامی که نیاز است بیش از یک دستور استفاده شود می تواند از بلوک استفاده کرد.
* برخی از ساختار های برنامه نویسی که به بلوک نیاز دارند. و بلوک این ساختارها را نمی توان با یک دستور ساده جایگزین کرد.
* اگر چه یک دستور ساده با نقطه و ویرگول خاتمه می یابد ولی یک بلوک با نقطه ویرگول دنبال نمی شود.( در واقع ، کامپایلر اجازه این کار را خواهد داد ، اما این روش خوبی نیست زیرا آن را به عنوان یک بیانیه خالی پردازش می کند.)

|  |
| --- |
| { نقطه و ویرگول پایان  ↓ نقطه و ویرگول پایان  int var2 = 5; ↓  System.Console.WriteLine("The value of var1 is {0}", var1);  }  ↑ نقطه و ویرگول پایان ندارد |

متن خروجی از یک برنامه

پنجره کنسول[[8]](#footnote-8) یک پنجره فرمان ساده و سریع است که اجازه می دهد یک برنامه متنی را نمایش دهد و از صفحه کلید ورودی دریافت کند. در BCL کلاسی به نام Console فراهم شده است ( در فضای نام System )، که شامل متد هایی برای ورود و خروج اطلاعات به پنجره کنسول است.

Write

Write عضوی از کلاس Console است. این یک رشته متنی را به پنجره کنسول می فرستد. در ساده ترین شکل آن Write یک رشته از متن را به پنجره می فرستد. این رشته باید در علامت نقل قول[[9]](#footnote-9) قرار گیرد.

خط زیر نمونه کدی را نشان می دهد که از عضو Write استفاده کرده است:

|  |
| --- |
| Console.Write("This is trivial text.");  ↑  رشته خروجی |

این کد خروجی زیر را در پنجره کنسول تولید می کند.

|  |
| --- |
| This is trivial text. |

کد زیر مثال دیگری است که سه رشته را به پنجره کنسول می فرستد:

|  |
| --- |
| System.Console.Write ("This is text1. ");  System.Console.Write ("This is text2. ");  System.Console.Write ("This is text3. "); |

این کد خروجی زیر را تولید می کند. توجه داشته باشید که Write خط جدیدی بعد از رشته ایجاد نمی کند، بنابراین خروجی این سه دستور در یک خط ایجاد می شود .

|  |
| --- |
| This is text1. This is text2. This is text3.  ↑ ↑ ↑  اولین دستور دومین دستور سومین دستور |

WriteLine

WriteLine عضو دیگر از Console است که کاری همانند تابع Write انجام می دهد اما کاراکتر خط جدید را به انتهای رشته خروجی اضافه می کند.

برای مثال ، اگر شما با استفاده از کد قبلی، WriteLine را جایگزین Write کنید، خروجی بر روی خطوط جداگانه تولید می شوند:

|  |
| --- |
| System.Console.WriteLine("This is text 1.");  System.Console.WriteLine("This is text 2.");  System.Console.WriteLine("This is text 3."); |

این کد خروجی زیر را در پنجره کنسول تولید می کند:

|  |
| --- |
| This is text 1.  This is text 2.  This is text 3. |

رشته فرمت

شکل کلی از دستورات Wirte و WriteLine بیش از یک پارامتر واحد را در برمی گیرد.

* اگر بیش از یک پارامتر واحد وجود داشته باشد، پارامتر ها توسط کاما از هم جدا می شوند.
* همیشه اولین پارامتر یک رشته است که رشته فرمت[[10]](#footnote-10) نام دارد. رشته فرمت می تواند شامل نشانگرهای جایگزین باشد.
  + نشانگر جایگزینی مکانی در رشته فرمت است که باید یک مقدار به جای آن در رشته خروجی جایگزین شود.
  + نشانگر جایگزینی شامل یک عدد صحیح است که با آکولاد محصور شده است. این عدد صحیح، موقعیت عددی از مقدار جایگزینی است که مورد استفاده قرار می گیرد. پارامتر های بعد از رشته فرمت، که ارزش های جایگزینی نامیده می شوند. این ارزش های جایگزینی اعدادی هستند که از 0 شروع می شوند.

نحوه کاربرد به شرح زیر است:

|  |
| --- |
| Console.WriteLine( *FormatString, SubVal0, SubVal1, SubVal2*, ... ); |

به عنوان مثال، دستور زیر دارای دو نشانگر جایگزین است و با 0 و 1 شماره گزاری شده اند و دو مقدار جایگزینی دارند که مقادیر آنها به ترتیب 3 و 6 هستند.

|  |
| --- |
| نشانگر جایگزین  ↓ ↓  Console.WriteLine("Two sample integers are {0} and {1}.", 3, 6);  ↑ ↑  رشته فرمت مقادیر جایگزین |

این کد خروجی زیر را در پنجره کنسول تولید می کند:

|  |
| --- |
| Two sample integers are 3 and 6. |

نشانگرهای چند گانه و مقادیر

در C# شما می توان هر تعداد از نشانگر ها و مقادیر را استفاده کرد.

* مقادیر می توانند به هر منظوری استفاده شوند.
* مقادیر می توانند به هر تعداد بار در رشته فرمت جایگزین شوند.

به عنوان مثال، دستور زیر از سه نشانگر و تنها از دو مقدار استفاده شده است. توجه کنید که مقدار 1 قبل از مقدار 0 و دو بار استفاده شده است.

|  |
| --- |
| Console.WriteLine("Three integers are {1}, {0} and {1}.", 3, 6); |

این کد خروجی زیر را در پنجره کنسول تولید می کند:

|  |
| --- |
| Three integers are 6, 3 and 6. |

مقدار یک نشانگر نباید از تعداد مقادیر جایگزینی بیشتر باشد. اگر این کار انجام شود ، کامپایلر خطایی تولید نمی کند اما در زمان اجرا خطا می گیرد.

برای مثال، در دستور زیر، دو مقدار جایگزینی در موقعیت های 0 و 1 وجود دارد. دومین نشانگر به مکان 2 رجوع شده است ( که وجود ندارد). این دستور در زمان اجرا یک خطا تولید می کند.

|  |
| --- |
| مکان 0 مکان 1  ↓ ↓  Console.WriteLine("Two integers are {0} and {2}.", 3 , 6); // خطا  ↑  مکانی با مقدار 2 وجود ندارد |

قالب بندی رشته های عددی

در طول این متن، نمونه کد ها برای نشان دادن مقادیر استفاده می شوند. استفاده از نشانگر جایگزینی ساده معمولاً شامل یک عدد صحیح بین جفت آکولاد ها است.

با این حال گاهی اوقات لازم است که برای نشان دادن رشته خروجی قالب های مناسب تری استفاده شود. به عنوان مثال، ممکن است که بخواهیم یک مقدار را به عنوان نرخ ارز و یا به عنوان یک مقدار با تعداد معینی از اعشار نشان دهیم. می توان برای انجام این کار از قالب بندی رشته ها استفاده کرد.

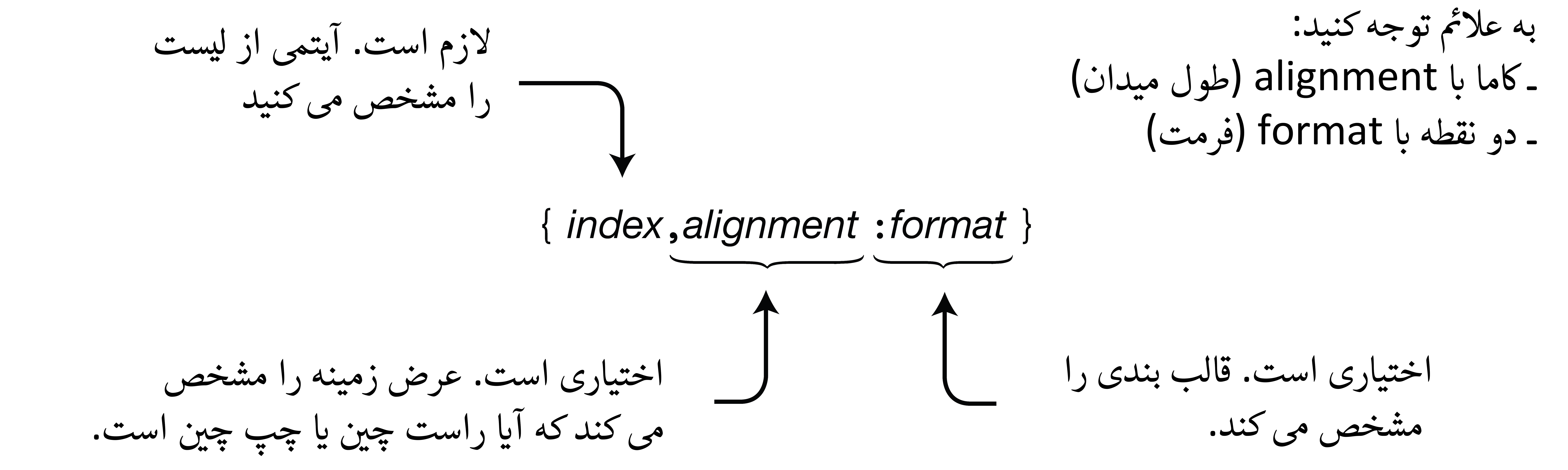
به عنوان مثال، کد زیر شامل دو دستور است که در خط اول مقدار 500 را بدون هیچ گونه قالب بندی اضافی چاپ می کند. و در خط دوم، رشته فرمت مشخص می کند که این عدد باید با فرمت ارز چاپ شود.

|  |
| --- |
| Console.WriteLine("The value: {0}." , 500); // چاپ کردن عدد  Console.WriteLine("The value: {0:C}.", 500); // چاپ کردن عدد با فرمت ارز  ↑  قالب بندی |

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| The value: 500.  The value: $500.00. |

تفاوت بین این دو دستور این است که قسمت قالب بندی دستور دوم شامل اطلاعات اضافی است. نحوه تعیین فرمت شامل سه فیلد در داخل آکولاد ها است: شاخص[[11]](#footnote-11) ، طول میدان[[12]](#footnote-12) ، فرمت[[13]](#footnote-13) . شکل 3-2 نحوه تعیین فرمت را نشان می دهد.



شکل 3-2. نحوه تعیین فرمت

اولین فیلد در تعیین فرمت index است. همانطور که می دانید، index مشخص می کند که کدام رشته از لیست باید قالب بندی شود. وجود index لازم است و شماره آیتم های لیست از صفر شروع می شوند.

تعیین طول میدان

طول میدان، مشخص کننده حداقل عرض زمینه از نظر تعداد کاراکتر ها است. و دارای مشخصات زیر است:

* اختیاری است و با کاما از index جدا می شود.
* شامل عددی صحیح مثبت یا منفی است.
  + عدد صحیح نشان دهنده حداقل تعداد کاراکتر هایی است که در زمینه استفاده شده اند.
  + راست یا چپ بودن متن با علامت مشخص می شود. یک عدد مثبت مشخص می کند که متن در سمت راست باشد و عدد منفی مشخص می کند که متن در سمت چپ باشد.

|  |
| --- |
| Index— از آیتم صفرم در لیست استفاده می شود.  ↓  Console.WriteLine("{0, 10}", 500);  ↑  طول میدان – از ده کاراکتر راست چین تشکیل شده است. |

به عنوان مثال، کد زیر دو قالب بندی را از متغییر myInt نشان می دهد. در حالت اول، مقدار myInt به عنوان رشته راست چین با حداقل ده کاراکتر نمایش داده می شود. و در حالت دوم، مقدار myInt به عنوان رشته ای چپ چین نمایش داده می شود، موارد قالب بندی بین دو نوار عمودی هستند، به طوری که در خروجی می توان محدوده رشته ها را دید.

|  |
| --- |
| int myInt = 500;  Console.WriteLine("|{0, 10}|", myInt); // Aligned right  Console.WriteLine("|{0,-10}|", myInt); // Aligned left |

این کد خروجی زیر را تولید می کند؛ ده کاراکتر بین نوار های عمودی قرار دارد:

|  |
| --- |
| | 500|  |500 | |

نمایش واقعی مقدار ممکن است بیشتر و یا کمتر از مقدار تعیین شده باشد:

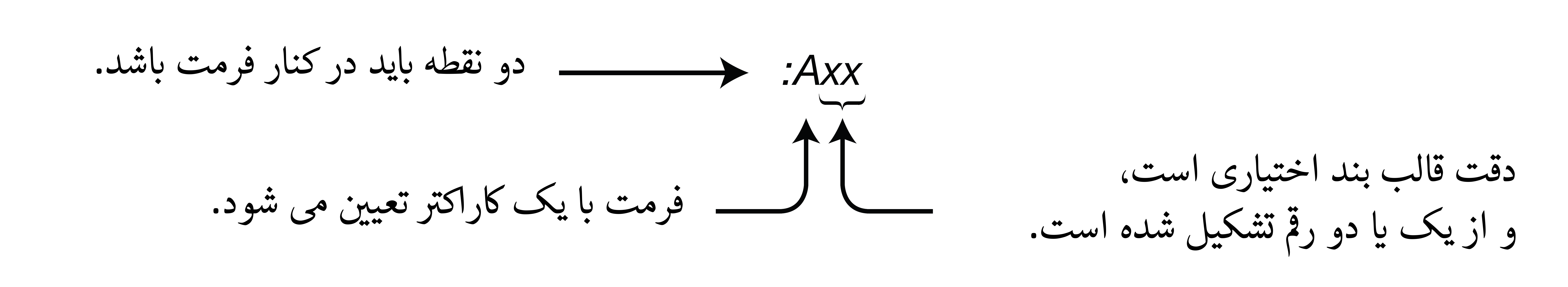
* اگر طول کاراکتر هایی قابل نمایش کمتر از طول میدان باشد کاراکتر های باقی مانده را با فضای خالی پر می کند.
* اگر طول کاراکتر های قابل نمایش بیشتر از طول میدان باشد کل کاراکتر ها چاپ و طول میدان نادیده گرفته می شود.

فرمت

فرمت، شکل بندی فرم است که نمایش عددی را مشخص می کند. به عنوان مثال، برای نمایش کاراکتر ارز و یا قالب بندی مبنای شانزده.

همانطور که در شکل 4-2 نشان داده شده است، حوزه فرمت دارای سه بخش است:

* بعد از کاراکتر دو نقطه باید فرمت تعیین شود.
* فرمت، کاراکتری است که تنها می تواند شامل یکی از نه کاراکتر فرمت باشد. این کاراکتر را می توان هم به صورت بزرگ و هم به صورت کوچک به کاربرد.
* دقت قالب بندی اختیاری است، و از یک یا دو عدد تشکیل شده است. معنای واقعی آن به فرمت استفاده شده بستگی دارد.



شکل 4-2. حوزه فرمت

کد زیر مثالی از نحوه تعیین فرمت را نشان می دهد:

|  |
| --- |
| Index— از آیتم صفرم در لیست استفاده می شود.  ↓  Console.WriteLine("{0:F4}", 12.345678);  ↑  فرمت – ممیز ثابت ، با چهار رقم اعشار |

کد زیر نمونه هایی از فرمت های مختلف را نشان می دهد:

|  |
| --- |
| double myDouble = 12.345678;  Console.WriteLine("{0,-10:G} -- General", myDouble);  Console.WriteLine("{0,-10} -- Default, same as General", myDouble);  Console.WriteLine("{0,-10:F4} -- Fixed Point, 4 dec places", myDouble);  Console.WriteLine("{0,-10:C} -- Currency", myDouble);  Console.WriteLine("{0,-10:E3} -- Sci. Notation, 3 dec places", myDouble);  Console.WriteLine("{0,-10:x} -- Hexadecimal integer", 1194719 ); |

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| 12.345678 -- General  12.345678 -- Default, same as General  12.3457 -- Fixed Point, 4 dec places  $12.35 -- Currency  1.235E+001 -- Sci. Notation, 3 dec places  123adf -- Hexadecimal integer |

فرمت های عددی استاندارد

جدول 4-2 خلاصه نه فرمت عددی استاندارد است. ستون اول لیست کاراکتر های فرمت را تعیین می کند.

جدول 4-2. فرمت های عددی استاندارد

|  |  |
| --- | --- |
| نام و کاراکتر ها | معنی |
| ارز[[14]](#footnote-14)  S و s | مقدار با استفاده از نماد ارز قالب بندی می شود، نماد ارز مورد استفاده به تنظیمات کامپیوتر بستگی دارد.   |  |  | | --- | --- | | مثال :  خروجی: | Console.WriteLine("{0 :C}", 12.5);  $12.50 | |
| دسیمال[[15]](#footnote-15)  D و d | رشته ای از اعداد بر مبنای ده دهی، همچنین برای اعداد مثبت و منفی مناسب است. تنها می تواند با اعداد صحیح استفاده شود.  دقت قالب بندی: حداقل تعداد ارقامی که در خروجی استفاده می شود. اگر تعداد ارقام کمتر باشد سمت چپ را با صفر پر می کند.   |  |  | | --- | --- | | مثال :  خروجی: | Console.WriteLine("{0 :D4}", 12);  0012 | |
| ممیز ثابت[[16]](#footnote-16)  F و f | رشته ای از اعداد بر مبنای اعداد اعشاری با ممیز ثابت، همچنین برای اعداد مثبت و منفی مناسب است.  دقت قالب بندی: تعداد رقم های اعشار   |  |  | | --- | --- | | مثال :  خروجی: | Console.WriteLine("{0 :F4}", 12.3456789);  12.3457 | |
| عمومی[[17]](#footnote-17)  G و g | بسته به مقدار می تواند یکی از فرمت های، ممیز ثابت و یا نماد علمی باشد. اگر هیچ فرمتی انتخاب نشود این فرمت به طور پیش فرض استفاده می شود.  دقت قالب بندی: به مقدار آن بستگی دارد.   |  |  | | --- | --- | | مثال :  خروجی: | Console.WriteLine("{0 :G4}", 12.3456789);  12.35 | |
| مبنای شانزده[[18]](#footnote-18)  X و x | رشته ای از اعداد مبنای شانزده است.  دقت قالب بندی: حداقل تعداد ارقامی که در رشته خروجی استفاده می شود. اگر تعداد ارقام کمتر باشد سمت چپ را با صفر پر می کند.   |  |  | | --- | --- | | مثال :  خروجی: | Console.WriteLine("{0 :x}", 180026);  2bf3a | |

|  |  |  |  |
| --- | --- | --- | --- |
| عدد[[19]](#footnote-19)  N و n | مشابه ممیز ثابت است اما با این تفاوت که بعد از اعشار عدد را سه رقم سه رقم با کاما از هم جدا می کند. نحوه جدا کردن عدد به تنظیمات کامپیوتر بستگی دارد.  دقت قالب بندی: تعداد رقم های اعشار   |  |  | | --- | --- | | مثال :  خروجی: | Console.WriteLine("{0 :N2}", 12345678.54321);  12,345,678.54 | |
| درصد[[20]](#footnote-20)  P و p | رشته ای که نشان دهنده درصد است. عدد در 100 ضرب می شود.  دقت قالب بندی: تعداد رقم های اعشار   |  |  | | --- | --- | | مثال :  خروجی: | Console.WriteLine("{0 :P2}", 0.1221897);  12.22 % | |
| رفت و برگشت[[21]](#footnote-21)  R و r | رشته ای برای خروجی انتخاب می شود که رشته تبدیل شده مقدار عددی استفاده شده در متد Parse را بازگرداند. این کار باعث می شود که مقدار واقعی انتخاب شود. متد Parse در فصل 25 بررسی شده است.  دقت قالب بندی: نادیده گرفته می شود.   |  |  | | --- | --- | | مثال :  خروجی: | Console.WriteLine("{0 :R}", 1234.21897);  1234.21897 | |
| نماد علمی[[22]](#footnote-22)  E و e | نماد علمی شامل جزء کسری و توان است. حرف E به این دلیل استفاده می شود که توان بعد از حرف E آورده شود.  دقت قالب بندی: تعداد رقم های اعشار   |  |  | | --- | --- | | مثال :  خروجی: | Console.WriteLine("{0 :e4}", 12.3456789);  1.2346e+001 | |

توضیحات: یادداشت های کد نویسی

در حال حاظر شما توضیحات[[23]](#footnote-23) تک خطی را دیده اید. بنابراین من در اینجا درباره نوع دومی از توضیحات محدوده ای و نوع سومی از توضیحات به نام توضیحات اسنادی[[24]](#footnote-24) بحث می کنم.

* توضیحات محدوده ای از دو کاراکتر مشخص شروع می شود و تا دو کاراکتر مشخص دیگر ادامه پیدا می کند.
* متن بین جفت علامت ها توسط کامپایلر نادیده گرفته می شود.
* توضیحات محدوده ای می توانند هر تعداد خط ادامه پیدا کند.

برای مثال، کد زیر یک توضیح محدوده ای را نشان می دهد که چندین خط را پوشش داده است :

|  |
| --- |
| ↓ شروع توضیحات چند خطی  /\*  This text is ignored by the compiler.  Unlike single-line comments, delimited comments  Like this one can span multiple lines.  \*/  ↑ پایان توضیحات |

همچنین توضیحات محدوده ای می تواند فقط شامل بخشی از یک خط باشد. به عنوان مثال، دستور زیر نشان می دهد که متنی از درون یک خط توضیح داده شده است. نتیجه دستور تنها تعریف متغییر var2 است.

|  |
| --- |
| شروع توضیحات  ↓  int /\*var 1,\*/ var2;  ↑  پایان توضیحات |

|  |
| --- |
| * توجه توضیحات یک خطی و محدوده ای در C# درست مثل C و C++ عمل می کند. |

اطلاعات بیشتر درباره توضیحات

مواردی که مهم است درباره توضیحات بدانید به شرح زیر است:

* اجازه استفاده از توضیحات تو در تو وجود ندارد. در یک زمان فقط یکی از توضیح می تواند اثر کند. اگر شما از توضیحات تو در تو استفاده کنید، توضیحات از اول شروع می شود تا به پایان اولین دامنه برسد نتیجه می دهد.
* دامنه برای انواع توضیحات خاص به شرح زیر است:
  + برای توضیحات یک خطی، توضیحات تا زمانی که خط به پایان برسد اثر بخش است.
  + برای توضیحات محدوده ای، توضیحات تا زمانی که به پایان اولین محدوده برسد اثر بخش است.

استفاده از توضیحات زیر نادرست است:

|  |
| --- |
| ↓ بازکردن توضیحات  /\* This is an attempt at a nested comment.  /\* ← به دلیل این که داخل توضیحات است نادیده گرفته می شود  Inner comment  \*/ ←به دلیل این که اولین پایان دهنده محدوده است توضیحات را می بندد  \*/ ←خطای نحوی می دهد به دلیل این که توضیحات برای این باز نشده است  ↓ باز کردن توضیحات ↓ به دلیل این که داخل توضیحات است نادیده گرفته می شود  // Single-line comment /\* Nested comment?  \*/ ← خطای نحوی می دهد به دلیل این که توضیحات برای این باز نشده است |

توضیحات اسنادی

همچنین C# نوع سومی از توضیحات را فراهم کرده است: توضیحات اسنادی. توضیحات اسنادی حاوی متن XML است که می تواند برای مستند سازی برنامه مورد استفاده قرار گیرد. این نوع توضیحات همانند توضیحات تک خطی است، بجز این که برای آنها به جای دو اسلش از سه اسلش متوالی استفاده می شود. توضیحات اسنادی را در فصل 25 مورد بررسی قرار می دهیم.

کد زیر روشی از توضیحات اسنادی را نشان می دهد:

|  |
| --- |
| /// <summary>  /// This class does...  /// </summary>  class Program  {  ... |

خلاصه ای از انواع توضیحات

توضیحات درونی، بخشی از متن هستند که توسط کامپایلر نادیده گرفته می شود اما شامل کد های آن سند است. برنامه نویسان توضیحات را در کد خود برای شرح دادن آن سند قرار می دهند. جدول 5-2 خلاصه ای از انواع توضیحات است.

جدول 5-2. انواع توضیحات

|  |  |  |  |
| --- | --- | --- | --- |
| نوع | شروع | پایان | توضیحات |
| تک خطی | // |  | متنی که از ابتدا علامت شروع می شود و تا پایان خط جاری توسط کامپایلر نادیده گرفته می شود. |
| محدوده ای | /\* | \*/ | متن بین شروع و پایان علامت ها توسط کامپایلر نادیده گرفته می شود. |
| اسنادی | /// |  | توضیحات از این نوع شامل متن XML است. به این معنا که از ابزاری برای تولید برنامه اسنادی استفاده شده است. جزئیات در فصل 25 بررسی می شود. |

1. Single-line comment [↑](#footnote-ref-1)
2. namespace [↑](#footnote-ref-2)
3. Visual Studio [↑](#footnote-ref-3)
4. Command-line [↑](#footnote-ref-4)
5. Idetifier [↑](#footnote-ref-5)
6. Contextual [↑](#footnote-ref-6)
7. Statement [↑](#footnote-ref-7)
8. Console window [↑](#footnote-ref-8)
9. quotation – (double quotes) [↑](#footnote-ref-9)
10. Format string [↑](#footnote-ref-10)
11. Index [↑](#footnote-ref-11)
12. Alignment [↑](#footnote-ref-12)
13. Format [↑](#footnote-ref-13)
14. Currency [↑](#footnote-ref-14)
15. Decimal [↑](#footnote-ref-15)
16. Fixed-point [↑](#footnote-ref-16)
17. General [↑](#footnote-ref-17)
18. Hexadecimal [↑](#footnote-ref-18)
19. Number [↑](#footnote-ref-19)
20. Percent [↑](#footnote-ref-20)
21. Round-trip [↑](#footnote-ref-21)
22. Scientific [↑](#footnote-ref-22)
23. comment [↑](#footnote-ref-23)
24. documentation comment [↑](#footnote-ref-24)