درس اول : آغاز کار با C#

در این درس با ارائه چند برنامه و مثال ساده به طرز کار زبان C# می‌پردازیم. اهداف این درس عبارتند از :

- فهم ساختار پایه‌ای یک برنامه C#
- آشنایی با Namespace
- آشنایی با کلاس (Class)
- آشنایی با عملکرد متد Main()
- آشنایی با ورودی/خروجی یا I/O

لیست ۱-۱، یک برنامهٔ ساده با عنوان **Welcome** در زبان C#

```
// Namespace اعلان
using  System;

// کلاس آغازین برنامه
class WelcomeCSS
 {
 // آغاز کار اجرای برنامه
  public static void Main()
   {
   // نوشتن متن در خروجی
    Console.WriteLine("Welcome to the C# Persian Tutorial!");
   }
 }
```

برنامهٔ لیست ۱-۱ دارای ۴ پارامتر اصلی است، اعلان **Namespace**، کلاس، متد **Main()** و یک دستور زبان **C#**.

در همین جا باید به یک نکته اشاره کنم، برای زبان C# همانند بیشتر زبانهای برنامه‌سازی دو نوع کامپایلر وجود دارد. یک نوع کامپایلر که به کامپایلر ***Command Line*** معروف است و نوع دیگر کامپایلر ***Visual*** است. کامپایلرهای ***Command Line*** محیطی شبیه به محیط **DOS** دارند و با دادن یک سری دستورات به اجرا در می‌آیند. کامپایلرهای ***Visual*** محیطی همانند ویندوز دارند که با دارا بودن محیط گرافیکی و ابزارهای خاص، برنامه‌نویس را در امر برنامه‌سازی کمک می‌کنند. از نمونه‌های هر یک از کامپایلرها، می‌توان به ***Microsoft C# Command Line Compiler*** که یک کامپایلر ***Command Line*** و ***Visual C#*** که یک کامپایلر ***Visual*** است، اشاره کرد. البته در حال حاضر بیشتر از کامپایلرهای ویژوال استفاده می‌شود.

من سعی می‌کنم در آینده به توضیح محیط **Visual C#** و **Visual Studio.Net** بپردازم. اما فعلاً برای اجرای برنامه‌ها می‌توانید از **Visual Studio.Net** استفاده کنید. پس از نصب آن، وارد محیط **C#** شده و در قسمت انتخاب برنامهٔ جدید گزینهٔ **Console** را جهت اجرای برنامه‌ها انتخاب نمایید.

برای این درس، فعلاً به توضیحات بیشتر دربارهٔ محیط ویژوال نمی‌پردازم اما در آینده به توضیح کامل محیط **Visual Studio.Net** خواهم پرداخت.

برای اجرای کد بالا در صورتیکه از محیط ویژوال استفاده می‌کنید باید بر روی دکمهٔ **Run** کلیک کنید و در صورتیکه کامپایلر ***Command Line*** دارید با دستور زیر می‌توانید برنامه را اجرا کنید : **csc Welcome.cs**

پس از اجرای برنامه، کامپایلر برای شما یک فایل قابل اجرا(**Executable**) تحت نام **Welcome.exe** تولید می‌کند.

نکته : در صورتیکه از Visual Studio.Net(VS.Net) استفاده کنید، پس از اجرای برنامه، یک صفحه برای نمایش خروجی به سرعت باز شده و بسته می‌شود و شما قادر به دیدن خروجی نخواهید بود. برای اینکه بتوانید خروجی برنامه را ببینید، در انتهای برنامه دستور زیر را وارد نمایید :

Console.ReadLine();

*استفاده از این دستور باعث می‌شود تا برنامه منتظر دریافت یک ورودی از کاربر بماند، که در این حالت شما می‌توانید خروجی برنامه خود را دیده و سپس با زدن کلید **Enter** برنامه را خاتمه دهید.*

نکتهٔ دیگری که در مورد زبان برنامه‌نویسی **C#** باید مورد توجه قرار دهید اینست که این زبان **Case Sensitive** است، بدین معنا که به حروف کوچک و بزرگ حساس است یعنی برای مثال **ReadLine** با **readLine** متفاوت است به طوریکه اولی جزو دستورات زبان **C#** و دومی به عنوان یک نام برای متغیر یا یک تابع که از طرف کاربر تعریف شده است در نظر گرفته می‌شود.

اعلان **Namespace** به سیستم اعلان می‌نماید که شما از توابع کتابخانه‌ای ***System*** جهت اجرای برنامه‌ها خود استفاده می‌نمایید. دستوراتی مانند **WriteLine** و **ReadLine** جزو توابع کتابخانه‌ای **System** می‌باشند. اغلب دستورات و توابع مهم و کلیدی استفاده از کنسول ورودی/خروجی در این کتابخانه می‌باشد. در صورتیکه در ابتدای برنامه از **using System** استفاده نکنید، باید در ابتدای هر یک از دستورات برنامه که مربوط این کتابخانه است، از کلمهٔ **System** استفاده نمایید. بعنوان مثال در صورت عدم استفاده از **using System** باید از دستور **System.Console.WriteLine()** به جای **Console.WriteLine()** استفاده نمایید.

تعریف کلاس، ***Class Welcome CSS***، شامل تعریف داده‌ها(متغیرها) و متدها جهت اجرای برنامه است. یک کلاس، جزو معدود عناصر زبان **C#** است که بوسیلهٔ آن می‌توان به ایجاد یک شی **(Object)** از قبیل واسط ها **(Interfaces)** و ساختارها**(Structures)**، پرداخت. توضیحات بیشتر در این زمینه در درس‌های آینده ذکر می‌شوند. در این برنامه کلاس هیچ داده و متغیری ندارد و تنها شامل یک متد است. این متد، *رفتار* **(Behavior)** این کلاس را مشخص می‌کند.

متد درون این کلاس بیان می‌کند که این کلاس چه کاری را پس از اجرا شدن انجام خواهد داد. کلمهٔ کلیدی **()Main** که نام متد این کلاس نیز می‌باشد جزو کلمات رزرو شده زبان **C#** است که مشخص می‌کند برنامه از کجا باید آغاز به کار نماید. وجود متد **()Main** در تمامی برنامه‌های اجرایی ضروری است. در صورتیکه یک برنامه حاوی متد **()Main** نباشد بعنوان توابع سیستمی همانند **dll** های ویندوز در نظر گرفته می‌شود.
قبل از کلمهٔ **()Main** کلمه دیگری با عنوان ***static*** آورده شده است. این کلمه در اصطلاح **Modifier** می‌گویند. استفاده از **static** برای متد **()Main** بیان می‌دارد که این متد تنها در در همین کلاس قابل اجراست و هیچ نمونه‌ای **(Instance)** دیگری از روی آن قابل اجرا نمی‌باشد. استفاده از **static** برای متد **()Main** الزامی است زیرا در ابتدای آغاز برنامه هیچ نمونه‌ای از هیچ کلاس و شی‌ای موجود نمی‌باشد و تنها متد **()Main** است که اجرا می‌شود. (در صورتیکه با برخی اصطلاحات این متن از قبیل کلاس، شی، متد و نمونه آشنایی ندارید، این به دلیل آنست که این مفاهیم جزو مفاهیم اولیهٔ برنامه‌نویسی شی‌گرا **(OOP)** هستند. سعی می‌کنم در درس‌های آینده به توضیح این مفاهیم نیز بپردازم، ولی فعلاً در همین حد کافی می‌باشد.)

هر متد باید دارای یک مقدار بازگشتی باشد، یعنی باید مقداری را به سیستم بازگرداند، در این مثال نوع بازگشتی ***void*** تعزیف شده است که نشان دهنده آنست که این متد هیچ مقداری را باز نمی‌گرداند یا به عبارت بهتر خروجی ندارد. همچنین هر متد می‌تواند دارای پارامترهایی نیز باشد که لیست پارامترهای آن در داخل پرانتزهای جلوی آن قرار می‌گیرد. برای سادگی کار در این برنامه متد ما دارای هیچ پارامتری نیست ولی در ادامهٔ همین درس به معرفی پارامترها نیز می‌پردازم.

متد **()Main** رفتار و عمل خود را بوسیله **Console.WriteLine(…)** مشخص می‌نماید. ***Console*** کلاسی در ***System*** است و ***WriteLine()*** متدی در *کلاس* ***Console***. در زبان **C#** از اپراتور "." (نقطه **dot**) جهت جداسازی زیرروتین‌ها و زیرقسمتها استفاده می‌کنیم. همانطور که ملاحظه می‌کنید چون **WriteLine()** یک متد درون کلاس **Console** است به همین جهت از "." جهت جداسازی آن استفاده کرده‌ایم.

در زبان **C#**، برای قرار دادن توضیحات در کد برنامه از // استفاده می‌کنیم. بدین معنا که کامپایلر در هنگام اجرای برنامه توجهی به این توضیحات نمی‌کند و این توضیحات تنها به منظور بالا بردن خوانایی متن و جهت و کمک به فهم بهتر برنامه قرار می‌گیرند.

استفاده از // تنها در مواردی کاربرد دارد که توضیحات شما بیش از یک خط نباشد. در صورت تمایل برای استفاده از توضیحات چند خطی باید در ابتدای شروع توضیحات از */ و در انتها آن از /* استفاده نمایید. در این حالت تمامی مطالبی که بین /* */ قرار می‌گیرند به عنوان توضیحات (Comments) در نظر گرفته می‌شوند.

تمامی دستورات (Statements) با ";"، سمی کولون، پایان می‌یابند. کلاس‌ها و متدها با { آغاز شده و با } خاتمه می‌یابند. تمامی دستورات بین { } یک بلوک را می‌سازند.

بسیاری از برنامه‌ها از کاربر ورودی دریافت می‌کنند. انواع گوناگونی از این ورودی‌ها می‌توانند به عنوان پارامتری برای متد Main() در نظر گرفته شوند. لیست ۲-۱ برنامه‌ای را نشان می‌دهد نام کاربر را از ورودی دریافت کرده و آن را بر روی صفحه نمایش می‌دهد. این ورودی به صورت پارامتری برای متد Main() در نظر گرفته شده است.

لیست ۲-۱ : برنامه‌ای که ورودی را از کاربر، بعنوان پارامتر Main()، دریافت می‌کند.

```
// Namespace اعلان
using System;
// کلاس آغازین برنامه
class NamedWelcome
  {
  // آغاز اجرا برنامه
   public static void Main(string[] args)
     {
    // نمایش بر روی صفحه
      Console.WriteLine("Hello, {0}!", args[0]);
      Console.WriteLine("Welcome to the C# Persian Tutorial!");
     }
  }
```

توجه داشته باشید که این برنامه، ورودی را به صورت Command-Line دریافت می‌کند و در هنگام اجرای برنامه باید ورودی را در Command-Line وارد نمایید. در صورتیکه ورودی را وارد ننمایید برنامه دچار مشکل شده و متوقف خواهد شد.

همان طور که در لیست ۲-۱ مشاهده می‌نمایید، پارامتر متد Main() با عنوان *args* مشخص شده است. با استفاده از این نام در داخل متد می‌توان آن استفاده نمود. نوع این پارامتر از نوع آرایه‌ای از نوع رشته (string[]) در نظر گرفته شده است. انواع(types) و آرایه‌ها را در درس‌های بعدی بررسی می‌کنیم. فعلاً بدانید که آرایه رشته‌ای جهت نگهداری چندین کاراکتر مورد استفاده قرار می‌گیرد. [] مشخص کننده آرایه هستند که مانند یک لیست عمل می‌کند.

همانطور که ملاحظه می‌کنید در این برنامه دو دستور Console.WriteLine(...) وجود دارد که اولین دستور مقداری با دستور دوم متفاوت است. همانطور که مشاهده می‌کنید داخل دستور Console.WriteLine(...) عبارتی به شکل *{0}* وجود دارد. این آرگومان، نشان می‌دهد که به جای آن چه مقداری باید نمایش داده شود که در این جا args[0] نشان داده می‌شود. عبارتی که داخل " " قرار دارد عیناً در خروجی نمایش داده می‌شود، به جای آرگومان {0}، مقداری که پس از " قرار دارد، قرار می‌گیرد. حال به آرگومان بعدی یعنی args[0] توجه کنید. مقدار صفر داخل [] نشان می‌دهد که کدام عنصر از آرایه مورد استفاده است. در C# اندیس آرایه از صفر شروع می‌شود به همین جهت برای دسترسی به اولین عنصر آرایه باید از اندیس صفر استفاده کنیم.(همانطور که قبلاً نیز گفتم آرایه‌ها را در درس‌های آینده توضیح خواهم داد، هدف از این درس تنها آشنایی با C# است.!)

مجدداً به آرگومان {0} بازگردیم. این آرگومان نشان می دهد که یک مقدار باید در رشته خروجی قرار گیرد، این مقدار همان args[0] است. اگر شما این برنامه را از طریق Command-Line اجرا نمایید خروجی شبیه به زیر خواهید گرفت :

```
>Hello!, Meysam!
>Welcome to C# Persian Tutorial!
```

همان گونه که می‌بینید، پس از اجرای برنامه نام شما که از طریق **Command-Line** آنرا وارد نموده‌اید در خروجی ظاهر می‌شود. استفاده از آرگومان {n}، که در آن **n** یک مقدار عددی است، جهت فرمت دادن به متن خروجی است که بر روی صفحه به نمـایش در می‌آید. مقدار **n** از صفر آغاز شده و به ترتیب افزایش می‌یابد. به مثال زیر توجه کنید :

Console.WriteLine("Hello! ,{0} ,{1}, {2}",args[0],args[1],args[2]);

این خط از برنامه سه مقدار **args[0],args[1],args[2]** را در خروجی به ترتیب نمایش می‌دهد. ملاحظه می‌نماییـد کـه چـون ۳ مقدار را می‌خواهیم نمایش دهیم، سه بار از آرگومان {n} استفاده کرده‌ایم و هر بار یک واحد به مقدار قبلی افزوده‌ایم. در آینـده بیشتر با این مفاهیم آشنا می‌شویم.

مطلبی که باید در مورد لیست ۲-۱ به آن توجه شود آنست که این برنامه تنها از طریق **Command-Line** قابـل اجراسـت و در صورتیکه کاربر از این مطلب که برنامه باید دارای ورودی به صورت **Command-Line** باشـد، بـی اطـلاع باشـد و ورودی را در **Command-Line** وارد نکند، برنامه متوقف شده و اجرا نمی‌شود. پس برای رفع چنین مشکلی باید از روش بهتری جهت دریافت ورودی از کاربر استفاده کرد.

لیست ۳-۱ : یک برنامه که قابلیت محاوره با کاربر را دارد.

```
// Namespace اعلان
using System;

// کلاس آغازین برنامه
class InteractiveWelcome
  {
  //آغاز اجرای برنامه
   public static void Main()
    {
    // متنی بر روی صفحه نمایش داده می‌شود
     Console.Write("What is your name?: ");
    //متنی نمایش داده شده و برنامه منتظر دریافت ورودی می‌ماند
     Console.Write("Hello, {0}! ", Console.ReadLine());
     Console.WriteLine("Welcome to the C# Persian Tutorial!");
    }
  }
```

همانطوریکه در این برنامه دیده می شود، متد **Main()** دارای پارامتر نیست. در عوض یک خط به متن برنامه لیست ۲-۱ اضافه شده است. در اولین خط از این برنامه، متنی با عنوان اینکه *نام شما چیست؟* بر روی صفحه ظاهر می‌شود. سپس در خط بعدی پس از نوشتن کلمه **Hello,** ، برنامه منتظر دریافت ورودی از کاربر می‌شود. بدین معنی که این بار تا زمانیکه کاربر متنی را به عنوان نام خود وارد نکند اجرای برنامه به پیش نخواهد رفت و خط بعدی اجرا نمی‌شود. این برنامه روش ایجاد ارتباط از طریق برنامه با کاربر را نمایش می‌دهد. در این مثال کاربر دقیقاً متوجه می‌شود که چه زمانی باید اطلاعات را وارد نماید و این اطلاعات چه باید باشد در حالیکه در مثال قبل چنین نبود. همانگونه که می‌بینید در این برنامه آرگومان {0} مستقیماً از طریق دستور **Console.ReadLine()** دریافت می‌شود و بلافاصله در خروجی نمایش داده می‌شود. دستور **ReadLine()** نیز یکی از متدهای کلاس **Console** است که بوسیله آن رشته ورودی خوانده می‌شود.

خروجی برنامه فوق به شکل زیر است :

What is your name?:

(سپس برنامه منتظر دریافت متنی از ورودی توسط کاربر می‌ماند)

(پس از اینکه کاربر رشته‌ای را وارد کرد و کلید**Enter** را فشار داد، متن زیر نمایش داده می‌شود.)

Hello, Meysam!

(سپس اجرای برنامه به دستور بعدی منتقل می‌شود)

Welcome to the C# Persian Tutorial!

خروجی کامل برنامه :

What is your name?:
Hello, Meysam! Welcome to the C# Persian Tutorial!

توجه کنید که **ReadLine()** به عنوان یک متد، مقداری را به سیستم بازمی‌گرداند. این مقدار در این برنامه به آرگومان **{0}** برگردانده می‌شود. این خط از برنامه را می‌توان طور دیگری هم نوشت :

```
string myName=Console.ReadLine();
Console.WriteLine("Hello, {0}!",myName);
```

در این حالت ما یک متغیر از نوع رشته با نام **myName** تعریف کرده‌ایم که مقدار ورودی در آن ذخیره می‌شود و سپس از این مقدار به عنوان مقداری که **{0}** می‌پذیرد استفاده کرده‌ایم.

در این درس آموختید که ساختار کلی یک برنامه **C#** چگونه است. هر برنامه **C#** از یک کلاس اصلی تشکیل می‌شود که این کلاس شامل داده‌ها و متغیرها و متدهایی می‌باشد. متد آغازین برنامه که برنامه با آن شروع به اجرا می‌کند، متد **Main()** است. با استفاده از توابع کتابخانه‌ای می‌توان به کلاسها و متدهای **C#** دسترسی پیدا کرد. از جمله این توابع **System** بود که یکی از کلاسهای آن **Console** و چند متد این کلاس، متدهای **WriteLine()** و **ReadLine()** بودند.

درس دوم – عبارات، انواع و متغیرها در C#

در این درس به معرفی عبارات، انواع و متغیرها در زبان **C#** می‌پردازیم. هدف از این درس بررسی موارد زیر است :

- آشنایی با متغیرها
- فراگیری انواع **(Types)** ابتدایی در **C#**
- فراگیری و درک عبارات **(Expressions)** در **C#**
- فراگیری نوع رشته‌ای **(String)** در زبان **C#**
- فراگیری چگونگی استفاده از آرایه‌ها **(Arrays)** در زبان **C#**

متغیرها، به بیان بسیار ساده، مکانهایی جهت ذخیره اطلاعات هستند. شما اطلاعاتی را در یک متغیر قرار می‌دهید و از این اطلاعات بوسیله متغیر در عبارات **C#** استفاده می‌نمایید. کنترل نوع اطلاعات ذخیره شده در متغیرها بوسیله تعیین کردن *نوع* برای هر متغیر صورت می‌پذیرد.

C# زبانی بسیار وابسته به انواع است، بطوریکه تمامی عملیاتی که بر روی داده‌ها و متغیرها در این زبان انجام می‌گیرد با دانستن نوع آن متغیر میسر می‌باشد. قوانینی نیز برای تعیین اینکه چه عملیاتی بر روی چه متغیری انجام شود نیز وجود دارد.(بسته به نوع متغیر)

انوع ابتدایی زبان **C#** شامل : یک نوع منطقی **(Boolean)** و سه نوع عددی اعداد صحیح **(integer)**، اعداد اعشاری **(Floating points)** و اعداد دسیمال **(Decimal)** می‌باشد.(به انواع **Boolean** از اینرو منطقی می‌گوییم که تنها دارای دو حالت منطقی صحیح **(True)** و یا غلط **(False)** می‌باشند.)

*مثال ۱ – نشان دادن مقادیر منطقی **(Boolean)***

```
using System;
class Booleans
{
  public static void Main()
   {
    bool content = true;
    bool noContent = false;
    Console.WriteLine("It is {0} that C# Persian provides C# programming language
       content.", content);
    Console.WriteLine("The statement above is not {0}.", noContent);
   }
}
```

در این مثال، مقادیر منطقی متغیرهای **Boolean** به عنوان قسمتی از جمله در خروجی نمایش داده می‌شوند. متغیرهای ***bool*** تنها می‌توانند یکی از دو مقدار **true** یا **false** را داشته باشند، یعنی همانند برخی از زبانهای برنامه‌سازی مشابه، مانند **C** و یا **C++**، مقدار عددی نمی‌پذیرند، زیرا همانگونه که می‌دانید در این دو زبان هر مقدار عددی صحیح مثبت بغیر از صفر به عنوان **true** و عدد صفر به عنوان **false** در نظر گرفته می‌شود و در حقیقت نوع **bool** در این دو زبان نوعی **integer** می‌باشند. اما در زبان **C#** انواع ***bool*** یکی از دو مقدار **true** یا **false** را می‌پذیرند. خروجی برنامه بالا به صورت زیر است :

```
It is True that C# Persian provides C# programming language content.
The statement above is not False.
```

جدول زیر تمامی انواع عددی صحیح **C#**، اندازه آنها و رنج قابل قبول آنها را نشان می‌دهد.

نوع	*اندازه به بیت*	*رنج قابل قبول*
sbyte	۸	۱۲۸- تا ۱۲۷
byte	۸	۰ تا ۲۵۵
short	۱۶	۳۲۷۶۸- تا ۳۲۷۶۷
ushort	۱۶	۰ تا ۶۵۵۳۵
int	۳۲	۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷
uint	۳۲	۰ تا ۴۲۹۴۹۶۷۲۹۵
long	۶۴	۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۰۸- تا ۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۰۷
ulong	۶۴	۰ تا ۱۸۴۴۶۷۴۴۰۷۳۷۰۹۵۵۱۶۱۵

از این انواع برای محاسبات عددی استفاده می‌گردد. یک نوع دیگر را نیز می‌توان در این جدول اضافه نمود و آن نوع ***char*** است. هر چند شاید از نظر بسیاری از دوستانی که با زبانهای دیگر برنامه‌سازی کار کرده‌اند این تقسیم بندی غلط به نظر آید، اما باید گفت که در زبان **C#** نوع ***char*** نیز نوع خاصی از انواع عددی است که رنجی بین صفر تا ۶۵۵۳۵ دارد و اندازه آن نیز ۱۶ بیتی است، اما به جای نمایش دادن مقادیر عددی تنها می‌تواند بیان کننده یک کاراکتر باشد. در آینده در این مورد بیشتر توضیح خواهم داد.

جدول زیر تمامی انواع عددی اعشاری زبان **C#** را نمایش می‌دهد.

نوع	*اندازه به بیت*	*دقت*	*رنج قابل قبول*
float	۳۲	۷ رقم	1.5×10^{-45} تا 3.4×10^{38}
double	۶۴	۱۵-۱۶ رقم	5.0×10^{-324} تا 1.7×10^{308}
decimal	۱۲۸	۲۸-۲۹ رقم دسیمال	1.0×10^{-28} تا 7.9×10^{28}

انواعی از نوع **floating point** هنگامی استفاده می‌شوند که محاسبات عددی به دقت‌های اعشاری نیاز داشته باشند. همچنین برای منظورهای تجاری استفاده از نوع ***decimal*** بهترین گزینه است. این نوع تنها در زبان **C#** وجود دارد و در زبانهای مشابه به آن نظیر **Java** چنین نوعی در نظر گرفته نشده است.

در یک زبان برنامه‌سازی نتایج بوسیله ایجاد یک سری *عبارت* تولید می‌گردند. *عبارات* از ترکیب متغیرها و عملگرها در *دستورالعمل‌های* یک زبان ایجاد می‌گردند.(توجه نمایید که عبارت معادل **expression** و دستورالعمل معادل **statement** می‌باشد که ایندو با یکدیگر متفاوت می‌باشند.) جدول زیر عملگرهای موجود در زبان **C#**، حق تقدم آنها و شرکت‌پذیری آنها را نشان می‌دهد.

نوع عمل	*عملگر(ها)*	*شرکت‌پذیری*
عملیات ابتدایی	**(x) x.y f(x) a[x] x++ x-- new typeof sizeof checked unchecked**	از چپ
عملیات یکانی	**+ - ! ~ ++x --x (T)x**	از چپ
عملیات ضربی	*** / %**	از چپ
عملیات جمعی	+ -	از چپ
عمل شیفت	**<< >>**	از چپ
عملیات رابطه‌ای	**< > <= >= is**	از چپ
عملیات تساوی	**== !=**	از راست
عمل **AND** منطقی	**&**	از چپ
عمل **OR** منطقی	**|**	از چپ
عمل **XOR** منطقی	**^**	از چپ
عمل **AND** شرطی	**&&**	از چپ
عمل **OR** شرطی	**||**	از چپ
عمل شرطی	**?:**	از چپ
عمل انتساب	**= *= /= %= += -= <<= >>= &= ^= |=**	از راست

شرکت‌پذیری از چپ بدین معناست که عملیات از چپ به راست محاسبه می‌شوند. شرکت‌پذیری از راست بدین معناست که تمامی محاسبات از راست به چپ صورت می‌گیرند. به عنوان مثال در یک عمل تساوی، ابتدا عبارات سمت راست تساوی محاسبه شده و سپس نتیجه به متغیر سمت چپ تساوی تخصیص داده می‌شود.

*مثال ۲- عملگرهای یکانی **(Unary)***

```
using System;
class Unary
{
  public static void Main()
  {
    int unary = 0;
    int preIncrement;
    int preDecrement;
    int postIncrement;
    int postDecrement;
    int positive;
    int negative;
    sbyte bitNot;
    bool logNot;
    preIncrement = ++unary;
    Console.WriteLine("Pre-Increment: {0}", preIncrement);
    preDecrement = --unary;
```

```
    Console.WriteLine("Pre-Decrement: {0}", preDecrement);
    postDecrement = unary--;
    Console.WriteLine("Post-Decrement: {0}", postDecrement);
    postIncrement = unary++;
    Console.WriteLine("Post-Increment: {0}", postIncrement);
    Console.WriteLine("Final Value of Unary: {0}", unary);
    positive = -postIncrement;
    Console.WriteLine("Positive: {0}", positive);
    negative = +postIncrement;
    Console.WriteLine("Negative: {0}", negative);
    bitNot = 0;
    bitNot = (sbyte)(~bitNot);
    Console.WriteLine("Bitwise Not: {0}", bitNot);
    logNot = false;
    logNot = !logNot;
    Console.WriteLine("Logical Not: {0}", logNot);
  }
}
```

به هنگام محاسبه عبارات، دو عملگر **x++** و **x—** (که در اینجا کاراکتر **x** بیان کننده آن است که عملگرهای ++ و — در جلوی عملوند قرار می‌گیرند **post-increment** و **post-decrement**) ابتدا مقدار فعلی عملوند **(operand)** خود را باز می‌گرداند و سپس به عملوند خود یک واحد اضافه کرده یا از آن یک واحد می‌کاهند. عملگر ++ یک واحد به عملوند خود اضافه می‌کند و عملگر — یک واحد از عملوند خود می‌کاهد. بدین ترتیب عبارت **x++** معادل است با عبارت **x=x+1** و یا **x+=1** اما همانطور که گفته شد باید توجه داشته باشید که این عملگرها(++ و --) ابتدا مقدار فعلی عملوند خود را برگشت می‌دهند و سپس عمل خود را روی آنها انجام می‌دهند. بدین معنی که در عبارت **x=y++** در صورتیکه در ابتدای اجرای برنامه مقدار **x=0** و **y=1** باشد، در اولین اجرای برنامه مقدار **x** برابر با ۱ یعنی مقدار **y** می‌شود و سپس به متغیر **y** یک واحد افزوده می‌شود، در صورتیکه اگر این عبارت را بصورت **x=++y** بنویسیم در اولین اجرای برنامه، ابتدا به مقدار متغیر **y** یک واحد افزوده می‌شود و سپس این مقدار به متغیر **x** تخصیص داده می‌شود که در این حالت مقدار متغیر **x** برابر با ۲ می‌شود.(در مورد عملگر — نیز چنین است.) پس با این توضیح می‌توان گفت که دو عملگر **x++** و **x—** ابتدا به عملوند خود یک واحد اضافه یا یک واحد از آن کم می‌کنند و سپس مقدار آنها را باز می‌گردانند.

در مثال ۲، مقدار متغیر **unary** در قسمت اعلان برابر با ۰ قرار گرفته است. هنگامیکه از عملگر **x++** استفاده می‌کنیم، به مقدار متغیر **unary** یک واحد افزوده می‌شود و مقدارش برابر با ۱ می‌گردد و سپس این مقدار، یعنی ۱، به متغیر **preIncrement** تخصیص داده می‌شود. عملگر **x—** مقدار متغیر **unary** را به ۰ باز می‌گرداند و سپس این مقدار را به متغیر **preDecrement** نسبت می‌دهد.

هنگامیکه از عملگر **x--** استفاده می‌شود، مقدار متغیر **unary**، یا همان مقدار صفر، به متغیر **postDecrement** تخصیص داده می‌شود و سپس از مقدار متغیر **unary** یک واحد کم شده و مقدار این متغیر به ۱- تغییر می‌کند. سپس عملگر **x++** مقدار متغیر **unary**، یعنی همان ۱-، را به متغیر **postIncrement** تخصیص می‌دهد و سپس یک واحد به مقدار متغیر **unary** می‌افزاید تا مقدار این متغیر برابر با ۰ (صفر) شود.

مقدار متغیر **bitNot** در هنگام اعلان برابر با صفر است. با استفاده از عملگر نقیض بیتی **(~)** (یا عملگر مکمل‌گیری) متغیر **bitNot** بعنوان یک بایت در نظر گرفته می‌شود و مقدار آن منفی یا نقیض می‌شود. در عملیات بیتی نقیض بدین معناست که تمامی یکها به صفر و تمامی صفرها به یک تبدیل شوند. در این حالت نمایش باینری عدد صفر یا همان **00000000** به نقیض آن یعنی **11111111** تبدیل می‌گردد.

در این مثال به عبارت **(sbyte)(~bitNot)** توجه نمایید. هر عملی که بر روی انـواع **short**، **unshort**، **byte** و **sbyte** انجام شود، مقداری از نوع **int** را باز می‌گرداند. بمنظور اینکه بتوانیم نتیجه دلخواه را به متغیر **bitNot** تخصیص دهـیم بایـد از فرمت **operator (Type)** استفاده نماییم که در آن **Type** نوعی است می‌خـواهیم نتیجـه مـا بـه آن نـوع تبـدیل شـود و **operator** عملی است که بر روی متغیر صورت می‌پذیرد. به بیان دیگر چون می‌خواهیم مقدار متغیر **bitNot** بصورت بیتی در نظر گرفته شود، پس باید نتیجه عمل ما بصورت بیتی در آن ذخیره شود که استفاده از نوع **sbyte** باعث می‌شود تا نتیجه به فرم بیتی (یا بایتی) در متغیر ما ذخیره شود. باید توجه نمایید که استفاده از فرمـت **(Type)** یـا در اصـطلاح عمـل **Casting**، در مواقعی که می‌خواهیم تغییری از یک نوع بزرگتر به نوع کوچکتر ایجاد نماییم، مورد استفاده قرار گیرد، چرا کـه در ایـن حـالات ممکن است با از دست دادن اطلاعات مواجه باشیم. در این مثال چون می‌خواهیم نوع بزرگتر **int** را به(۳۲ بیتی) به نوع کـوچکتر **sbyte** (۸ بیتی) تبدیل نماییم، بدین منظور باید بطور صریح از عمل **Casting** استفاده نماییم تا اطلاعاتی در این تبدیل از بین نرود. در مورد تبدیل انواع کوچکتر به انواع بزرگتر(مثلا تبدیل **sbyte** به **int**) نیازی به استفاده از عمل **Casting** نیست چـرا که امکان از بین رفتن اطلاعات وجود ندارد. در ضمن باید به یک نکته مهم توجه نمایید و آن تبدیل انواع علامتدار **(Signed)** و بدون علامت **(Unsigned)** به یکدیگر است. در این حالت خطر بسیار مهمی داده‌های شما را تهدید می‌نمایـد. بحـث در مـورد مسائل پیچیده‌تر در مورد تبدیل انواع علامتدار و و بدون علامت به یکدیگر در اینجا نمی‌گنجد و سعی می‌کنم تا آنها را در مطالب بعدی و در جای لازم مورد بحث و بررسی قرار دهم.(در صورتیکه برخی از مطالب این قسمتها برای شما به خوبی قابل درک نیست، نگران نباشید چراکه در آینده در مثالهایی که خواهید دید تمامی این مطالب را در عمل نیز حس کرده و با آنها آشنا خواهید شد.)

عملگر بعدی که در این برنامه مورد استفاده قرار گرفته است، عملگر نقیض منطقی یا همان **"!"** است که امکان تغییر مقدار یـک متغیر **Boolean** را از **true** به **false** و بالعکس را فراهم می‌آورد. در مثال بالا(مثال شماره ۲) مقدار متغیـر **logNot** پـس از استفاده از عملگر **"!"** از **false** به **true** تغییر کرده است. با توجه به توضیحات اخیر خروجی زیر از برنامه مثال ۲ مـورد انتظـار است :

```
Pre-Increment: 1
Pre-Decrement 0
Post-Decrement: 0
Post-Increment -1
Final Value of Unary: 0
Positive: 1
Negative: -1
Bitwise Not: -1
Logical Not: True
```

مثال ۳ – عملگرهای دوتایی

```
using System;
class Binary
{
  public static void Main()
  {
    int x, y, result;
    float floatResult;
    x = 7;
    y = 5;
    result = x+y;
    Console.WriteLine("x+y: {0}", result);
    result = x-y;
    Console.WriteLine("x-y: {0}", result);
    result = x*y;
    Console.WriteLine("x*y: {0}", result);
    result = x/y;
    Console.WriteLine("x/y: {0}", result);
```

```
   floatResult = (float)x/(float)y;
   Console.WriteLine("x/y: {0}", floatResult);
   result = x%y;
   Console.WriteLine("x%y: {0}", result);
   result += x;
   Console.WriteLine("result+=x: {0}", result);
  }
}
```

خروجی این برنامه به فرم زیر است :

```
x+y: 12
x-y: 2
x*y: 35
x/y: 1
x/y: 1.4
x%y: 2
result+=x: 9
```

مثال ۳ استفاده‌های متفاوتی از عملگرهای دوتایی را نشان می‌دهد.(منظور از عملگر دوتایی، عملگری است که دارای دو عملوند می‌باشد مانند عملگر جمع "+"). بسیاری از عملگرهای مورد استفاده در این مثال عملگرهای ریاضی هستند و نتیجه عمل آنها مشابه عملی است که از آنها در ریاضیات دیده‌اید. از نمونه این عملگرها می‌توان به عملگرهای جمع "+"، تفریق "-"، ضرب "*" و تقسیم "/" اشاره نمود.

متغیر **floatResult** از نوع اعشاری یا **float** تعریف شده است. در این مثال نیز صریحاً از عمل **Casting** جهت اسفاده از دو متغیر **x** و **y** که از نوع **int** هستند، برای انجام عملی که نتیجه‌اش از نوع **float** است، استفاده کرده‌ایم.

در این مثال از عملگر "٪" نیز استفاده کرده‌ایم. این عملگر در عملیات تقسیم کاربرد دارد و باقیمانده تقسیم را برمی‌گرداند. یعنی دو عملوند خود را بر یکدیگر تقسیم می‌کند و باقیمانده این تقسیم را برمی‌گرداند.

در این مثال همچنین فرم جدیدی از عمل انتساب را بصورت **result+=x** مشاهده می‌نمایید. استفاده از عملگرهای انتسابی که خود ترکیبی از دو عملگر هستند، جهت سهولت در امر برنامه‌نویسی مورد استفاده قرار می‌گیرند. عبارت فوق معادل **result = result+x** می‌باشد. یعنی مقدار قبلی متغیر **result** با مقدار متغیر **x** جمع می‌شود و نتیجه در متغیر **result** قرار می‌گیرد.

یکی دیگر از انواعی که تا کنون با آن سر و کار داشته‌ایم *نوع رشته‌ای* ***(string)*** است. یک رشته، از قرار گرفتن تعدادی کاراکتر در کنار یکدیگر که داخل یک زوج کوتیشن " " قرار گرفته‌اند، ایجاد می‌گردد. بعنوان مثال **"Hi This is a string type"**. در اعلان متغیرها نیز در صورت تعریف متغیری از نوع رشته‌ای، در صورت نیاز به تخصیص مقدار به آن، حتماً کاراکترهایی که می‌خواهیم بعنوان یک رشته به متغیرمان نسبت دهیم را باید داخل یک زوج کوتیشن " " قرار دهیم. به مثال زیر توجه نمایید.

```
string Name;
...
Name = "My name is Meysam";
```

همانطور که در این مثال مشاهده می‌نمایید، متغیری از نوع رشته‌ای تحت نام **Name** تعریف شده است و سپس در جایی از برنامه که نیاز به تخصیص مقدار برای این متغیر وجود دارد، عبارت مورد نظر را داخل دو کوتیشن قرار داده و به متغیر خود تخصیص داده‌ایم. رشته‌ها از پر کاربرد ترین انواع در زبان‌های برنامه‌سازی جهت ایجاد ارتباط با کاربر و دریافت اطلاعات از کاربر می‌باشند.(همانطور که در درس قبل اول نیز گفته شد، دستور **Console.ReadLine()** یک رشته را از ورودی دریافت می‌نماید.) در مثالهایی که در طی درسهای این سایت خواهید دید، نمونه‌های بسیاری از کاربرد انواع مختلف و نیز نوع رشته‌ای را خواهید دید.

آرایه‌ها(Arrays)

یکی دیگر از انواع داده‌ای در زبان C#، *آرایه‌ها* ***(Arrays)*** می‌باشند. یک آرایه را به عنوان مخزنی برای نگهداری اطلاعات در نظر می‌گیریم که دارای لیستی از محلهایی است که در آنها اطلاعات ذخیره شده است و از طریق این لیست می‌توان به اطلاعات آنها دسترسی پیدا نمود. به هنگام اعلان آرایه‌ها باید نوع، اندازه و تعداد بعد آنها را نیز معین نمود.

مثال ۴- آرایه‌ها و عملیات بر روی آنها

```
using System;
class Array
{
  public static void Main()
   {
    int[] myInts = { 5, 10, 15 };
    bool[][] myBools = new bool[2][];
    myBools[0] = new bool[2];
    myBools[1] = new bool[1];
    double[,] myDoubles = new double[2, 2];
    string[] myStrings = new string[3];
    Console.WriteLine("myInts[0]: {0}, myInts[1]: {1}, myInts[2]: {2}", myInts[0],
    myInts[1], myInts[2]);
   myBools[0][0] = true;
   myBools[0][1] = false;
   myBools[1][0] = true;
   Console.WriteLine("myBools[0][0]: {0}, myBools[1][0]: {1}", myBools[0][0],
   myBools[1][0]);
   myDoubles[0, 0] = 3.147;
   myDoubles[0, 1] = 7.157;
   myDoubles[1, 1] = 2.117;
   myDoubles[1, 0] = 56.00138917;
   Console.WriteLine("myDoubles[0, 0]: {0}, myDoubles[1, 0]: {1}", myDoubles[0,
0],   myDoubles[1, 0]);
   myStrings[0] = "Joe";
   myStrings[1] = "Matt";
   myStrings[2] = "Robert";
   Console.WriteLine("myStrings[0]: {0}, myStrings[1]: {1}, myStrings[2]: {2}",
   myStrings[0], myStrings[1], myStrings[2]);
   }
}
```

خروجی مثال ۴ بصورت زیر است :

```
myInts[0]: 5, myInts[1]: 10, myInts[2]: 15
myBools[0][0]: True, myBools[1][0]: True
myDoubles[0, 0]: 3.147, myDoubles[1, 0]: 56.00138917
myStrings[0]: Joe, myStrings[1]: Matt, myStrings[2]: Robert
```

در این مثال انواع مختلفی از آرایه‌ها اعلان شده‌اند. در ابتدا یک آرایه تک بعدی، سپس آرایه‌ای دندانه‌دار و در نهایت نیز یک آرایه دو بعدی در این مثال اعلان شده‌اند.

اولین اعلان در این برنامه مربوط به اعلان آرایه تک بعدی **myInts** می‌باشد که از نوع **int** بوده و دارای ۳ عضو می‌باشد که تعداد این اعضا با اعلان چند مقدار در داخل { } معین شده است. همانطور که از این اعلان دریافت میشود، آرایه تک بعدی بصورت زیر تعریف می‌شود :

```
type[] arrayName;
```

که در آن **type** نوع آرایه و **arrayName** نام آرایه ایست که تعریف می‌نمائیم.

اما در ابتدا گفته شد که به هنگام اعلان آرایه‌ها اندازه آنها نیز باید مشخص شود. برای تعیین اندازه آرایه، یعنی تعدا عناصری که آرایه در خود جای می‌دهد، می‌توان به چند روش عمل نمود. اولین و ساده‌ترین روش که در این مثال نیز آورده شده است، تخصیص مقادیری به آرایه در داخل یک زوج { } است. بسته به نوع آرایه، تعداد عناصری که داخل این زوج { } قرار می‌گیرند، تعداد عناصر آرایه می‌باشند و مقادیر عناصر آرایه نیز همان مقادیری است که داخل { } قرار گرفته است. به عنوان مثال در مثال ۴، اولین آرایه ما دارای ۳ عنصر است که مقادیر آنها به ترتیب برابر با ۵، ۱۰ و ۱۵ می‌باشد.

روش دیگر جهت تعیین اندازه آرایه استفاده از روش تعریف کامل آرایه است که به فرم کلی زیر می‌باشد.

```
type[] arrayName = new type[n];
```

که در این تعریف، استفاده از کلمه کلیدی **new** باعث ایجاد نمونه‌ای جدید از نوع مورد نظر، می‌شود. **n** نیز تعداد عناصر آرایه است که می‌خواهیم آنرا تولید نماییم. در این حالت باید توجه داشت که آرایه‌ای تهی را تولید نموده‌ایم و هیچ عنصری را در آرایه جای نداده‌ایم و در برنامه باید آرایه را مقدار دهی نماییم. به مثال زیر توجه کنید.

```
int[] myArray = new int[15];
```

این مثال آرایه‌ای تک بعدی از نوع **int** را با اندازه ۱۵ عنصر تولید می‌نماید. یعنی این آرایه قادر است تا ۱۵ عنصر از نوع **int** را در خود ذخیره نماید.

گونه دیگری از آرایه‌ها، آرایه‌های چند بعدی **(Multi Dimensional Arrays)** هستند که برای نگهداری اطلاعات از چندین بعد استفاده می‌کنند و بیشتر برای نگداری جداول و ماتریسها مورد استفاده قرار می‌گیرند. فرم کلی اعلان این آرایه‌ها بصورت زیر است :

```
type[ , , ... , ]  arrayName = new type[n1, n2, .... , nm];
```

که در آن تعداد ابعاد آرایه با ویرگول مشخص شده و **n1** تا **nm** نیز تعداد عناصر هر یک از ابعاد است. بعنوان مثال تعریف یک آرایه سه بعدی به فرم زیر است :

```
char[ , , ] charArray = new char[3,5,7];
```

در این مثال یک آرایه سه بعدی از نوع **char** تولید کرده‌ایم که ابعاد آن به ترتیب دارای ۳، ۵ و ۷ عنصر می‌باشند.

نوع دیگری از آرایه‌ها، آرایه‌های دندانه‌دار **(Jagged Arrays)** هستند. این نوع آرایه‌ها تنها در زبان **C#** وجود دارند و در صرفه‌جویی حافظه بسیار موثر میباسند. یک آرایه دندانه‌دار، در حقیقت یک آرایه تک بعدی است که هر یک از اعضای آن خود یک آرایه تک بعدی می‌باشند. اندازه این عناصر می‌تواند متفاوت باشد و تفاوت این آرایه‌ها با آرایه‌های چند بعدی در همین جا نمایان می‌شود. استفاده از این آرایه‌ها در مواردی کاربرد دارد که نیازی نیست تا تمامی ابعاد آرایه دارای تعداد عناصر مساوی باشند. بعنوان مثال فرض کنید می‌خواهید آرایه‌ای جهت نگهداری تعداد روزهای ماههای مختلف سال تهیه کنید. در صورتیکه بخواهید از آرایه چند بعدی استفاده نمایید، چون تعداد روزهای تمامی ماههای سال یکسان نیست، مجبورید تا تعداد عناصر تمام بعدهای آرایه را برابر با بزرگترین تعداد روز ماهها، یعنی ۳۱، تعریف نمایید. ولی چون تنها ۶ ماه دارای ۳۱ روز می‌باشند، برای ۶ ماه دیگر تعدادی از عناصر آرایه هیچگاه مورد استفاده قرار نمی‌گیرند و حافظه را به هدر داده‌ایم. اما در صورتیکه برای این مثال از آرایه‌های دندانه‌دار استفاده نماییم، می‌توانیم یک آرایه دندانه‌دار ۱۲ عنصری تعریف نماییم و سپس تعداد عناصر هر یک از اعضای آنرا برابر با تعداد روزهای ماه مورد نظر تعریف کنیم :

با استفاده از آرایه چند بعدی :

```
int[ , ] monthArray = new int[12,31];
```

با استفاده از آرایه دندانه‌دار :

```
int[][] monthArray = new int[12][];
```

در تعریف اول که در آن از آرایه چند بعدی استفاده کردیم، مشاهده می‌کنید که آرایه‌ای دو بعدی تعریف کرده‌ایم که بعد اول آن ۱۲ عضو و بعد دوم آن ۳۱ عضو دارد. این عمل دقیقاً همانند ایجاد یک جدول برای نگهداری روزهای ماههای سال است.

اما در حالت دوم که در آن از آرایه دندانه‌دار بهره برده‌ایم، یک آرایه تعریف نموده‌ایم که بعد اول آن ۱۲ عضو دارد ولی بعد دوم آنرا را تعریف نکرده‌ایم که دارای چند عضو است و هر یک از عناصر بعد اول آرایه می‌تواند دارای تعداد اعضای متفاوتی باشد که با

استفاده از این روش می‌توان به هر یک از ماههای سال تعداد روزهای مورد نظر آن ماه را تخصیص داد و فضایی بلا استفاده ایجاد نخواهیم کرد. توجه نمایید که چون تعداد عناصر بعد دیگر این آرایه معین نشده است در برنامه باید این تعداد عنصر را مشخص نماییم :

```
monthArray[1] = new int[31];
monthArray[10] = new int [30];
```

مشاهده می‌کنید که به هر ماه، تعدا عنصر مورد نیاز خود را تخصیص داده‌ایم. تنها باید به تفاوت اعلان آرایه‌های دندانه‌دار با آرایه‌های چند بعدی توجه نمایید.

دسترسی به عناصر آرایه از طریق اندیس امکان پذیر است. اندیس شماره محل ذخیره‌سازی داده‌های ما می‌باشد که با دادن این شماره می‌توانیم به داده مورد نظر دسترسی پیدا کنیم. در **C#** همانند **C** و **C++** اندیس خانه‌های آرایه از صفر آغاز می‌گردد یعنی اولین خانه آرایه دارای شماره صفر است و عناصر بعدی به ترتیب یک واحد به اندیسشان اضافه می‌گردد. پس شماره اندیس آرایه همیشه یک واحد کمتر از تعداد عناصر آن است، یعنی آرایه‌ای که ۱۰ عضو دارد بزرگترین اندیس خانه‌هایش ۹ می‌باشد. دسترسی به عناصر هر یک از ابعاد آرایه با اندیس امکان پذیر است. معمولاً به بعد اول آرایه سطر و به بعد دوم آن ستون می‌گوید. مثلاً **monthArray[3,7]** عنصر واقع در سطر ۳ و ستون ۷ آرایه را مشخص می‌نماید.(توجه داشته باشید که اندیس دهی آرایه از صفر آغاز می‌شود و بعنوان مثال **intArray[12]** به خانه شماره ۱۲ آرایه اشاره می‌کند اما فراموش نکنید چون اندیس آرایه از صفر آغاز می‌شود خانه شماره ۱۲ آرایه، سیزدهمین داده شما را در خود جای می‌دهد.)

۰	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲

اگر شکل فوق را آرایه‌ای تک بعدی تصور نمایید، مشاهده می‌نمایید که خانه شماره ۵ آرایه حاوی اطلاعات مربوط به ششمین داده ورودی شما می‌باشد.

نکته دیگری که باید در مورد تعریف آرایه‌های این مثال متذکر شوم در مورد آریه‌هائیست که از نوع **string** تعریف می‌شوند. دوستانی که با زبان **C** کار کرده‌اند حتماً می‌دانند که آرایه‌ای از نوع رشته‌ای در **C** وجود ندارد و برای نگهداری چندین رشته در یک آرایه باید از آرایه دو بعدی استفاده کرد. در **C#** این قابلیت فراهم شده تا با استفاده از یک آرایه تک بعدی بتوان چندین رشته را ذخیره نمود بدین صورت که هر یک از عناصر آرایه نک بعدی محلی برای ذخیره‌سازی یک رشته است و همانند زبان **C** نیاز به پردازش‌های گاه پیچیده بر روی آرایه‌های چند بعدی بمنظور کار با رشته‌ها، وجود ندارد. بعنوان یک توضیح کمی اختصاصی عرض می‌کنم که در زبان‌هایی مانند **C**، در صورتیکه می‌خواستید چندین رشته را در آرایه‌ای ذخیره کنید تا بتوانید با اندیس به آنها دسترسی داشته باشید، مجبور به تعریف یک آرایه دو بعدی بودید که با استفاده از تنها اندیس اول آرایه می‌توانستید به عناصر رشته‌ای آرایه دسترسی پیدا کنید، اما در **C#** تنها با استفاده از یک آرایه تک بعدی می‌توان همان کار را انجام داد.

```
string[] stringArray = {"My name is Meysam", "This is C# Persian Blog"}
.....
Console.WriteLine("{0}",stringArray[0]);
.....
```

همانطور که در این مثال ملاحظه می‌کنید، آرایه‌ای از نوع رشته تعریف شده و دو عنصر به آن تخصیص داده شده است و در جایی در متن برنامه با استفاده از اندیس از اولین عنصر این آرایه برای نمایش در خروجی استفاده گردیده است. خروجی این برنامه به شکل زیر است :

```
My name is Meysam
```

درس سوم – دستورالعمل‌های کنترلی و شرطی

در این درس با دستورالعمل‌های کنترل و انتخاب در **C#** آشنا می‌شوید. هدف این درس عبارتست از :

- یادگیری دستور ***if***
- یادگیری دستور ***switch***
- نحوه بکارگیری دستور ***break*** در دستور ***switch***
- درک صحیح از نحوه بکارگیری دستور ***goto***

*بررسی دستور **if** و انواع مختلف آن*

در درسهای گذشته، برنامه‌هایی که مشاهده می‌کردید از چندین خط دستور تشکیل شده بودند که یکی پس از دیگری اجرا می‌شدند و سپس برنامه خاتمه می‌یافت. در این برنامه‌ها هیچ عمل تصمیم‌گیری صورت نمی‌گرفت و تنها دستورات برنامه به ترتیب اجرا می‌شدند. مطالب این درس نحوه تصمیم‌گیری در یک برنامه را به شما نشان می‌دهد.

اولین دستور تصمیم‌گیری که ما آنرا بررسی می‌نماییم، دستورالعمل ***if*** است. این دستور دارای سه فرم کلی : تصمیم‌گیری ساده، تصمیم‌گیری دوگانه، تصمیم‌گیری چندگانه می‌باشد.

*مثال ۱-۳ – فرم‌های دستورالعمل **if***

```
using System;

class IfSelect
{
  public static void Main()
  {
   string myInput;
   int myInt;
   Console.Write("Please enter a number: ");
   myInput = Console.ReadLine();
   myInt = Int32.Parse(myInput);
   // تصمیم‌گیری ساده و اجرای عمل داخل دو کروشه
   if (myInt > 0)
    {
     Console.WriteLine("Your number {0} is greater than zero.", myInt);
    }
   //تصمیم‌گیری ساده و اجرای عمل بدون استفاده از دو کروشه
   if (myInt < 0)
     Console.WriteLine("Your number {0} is less than zero.", myInt);
   // تصمیم‌گیری دوگانه
   if (myInt != 0)
    {
     Console.WriteLine("Your number {0} is not equal to zero.", myInt);
    }
   else
    {
     Console.WriteLine("Your number {0} is equal to zero.", myInt);
    }
   // تصمیم‌گیری چندگانه
   if (myInt < 0 || myInt == 0)
    {
```

```
    Console.WriteLine("Your number {0} is less than or equal to zero.",  myInt);
   }
  else if (myInt > 0 && myInt <= 10)
  {
   Console.WriteLine("Your number {0} is between 1 and 10.", myInt);
  }
  else if (myInt > 10 && myInt <= 20)
  {
   Console.WriteLine("Your number {0} is between 11 and 20.", myInt);
  }
  else if (myInt > 20 && myInt <= 30)
  {
   Console.WriteLine("Your number {0} is between 21 and 30.", myInt);
  }
 else
 {
  Console.WriteLine("Your number {0} is greater than 30.", myInt);
 }
 } //Main() پایان متد
} //IfSelect پایان کلاس
```

برنامه ۱-۳ از یک متغیر **myInt** برای دریافت ورودی از کاربر استفاده می‌نماید، سپس با استفاده از یک سری دستورات کنترلی، که همان دستور **if** در اینجاست، عملیات خاصی را بسته به نوع ورودی انجام می‌دهد. در ابتدای ایـن برنامـه عبـارت **Please enter a umber:** در خروجی چاپ می‌شود. دستور **Console.ReadLine()** منتظر می‌ماند تا کاربر ورودی وارد کـرده و سپس کلید **Enter** را فشار دهد. همانطور که در قبل نیز اشاره کرده‌ایم، دستور **Console.ReadLine()** عبارت ورودی را به فرم رشته دریافت می‌نماید پس مقدار ورودی کاربر در اینجا که یک عدد است به فرم رشته‌ای در متغیر **myInput** کـه از نـوع رشته‌ای تعریف شده است قرار می‌گیرد. اما میدانیم که برای اجرای محاسبات و یا تصمیم‌گیری بر روی اعداد نمی‌توان از آنهـا در فرم رشته‌ای استفاده کرد و باید آنها را بصورت عددی مورد استفاده قرار داد. به همین منظور باید متغیر **myInput** را به نحوی به مقدار عددی تبدیل نماییم. برای این منظور از عبارت **Int32.Parse()** استفاده می‌نماییم. این دستور مقدار رشته‌ای متغیر داخل پرانتزش را به مقدار عددی تبدیل کرده و آنرا به متغیر دیگری از نوع عددی تخصیص می‌دهد. در این مثال نیز همانطور کـه دیده می‌شود، **myInput** که تز نوع رشته‌ای است در داخل پرانتز قرار گرفته و این مقدار برابر با **myInt** که از نوع **int** اسـت قرار گرفته است. با این کار مقدار عددی رشته ورودی کاربر به متغیر **myInt** تخصیص داده می‌شود. (توضیح کامل‌تری در مـورد **Int32** و سایر تبدیلات مشابه به آن در درسهای آینده و در قسمت نوع‌های پیشرفته مورد بررسی قـرار مـی‌گیرنـد.)حـال مـا متغیری از نوع مورد نظر در دست داریم و می‌توانیم با استفاده از دستور **if** بر روی آن پردازش انجام داده و تصمیم‌گیری نماییم.

دستور if

اولین دستور بصورت ***if (boolean expression) {statements}*** آورده شده است. دسـتور **if** بـا اسـتفاده از کلمـه کلیدی **if** آغاز می‌شود. سپس یک عبارت منطقی درون یک زوج پرانتز قرار مـی‌گیـرد . پـس از بررسـی ایـن عبـارات منطقـی دستورالعمل/دستورالعمل‌های داخل کروشه اجرا می‌شوند. همانطور که مشاهده می‌نمایید، دستور **if** یک عبارت منطقی را بررسی می‌کند. در صورتیکه مقدار این عبارات **true** باشد دستورهای داخل بلوک خود را اجرا مـی‌نمایـد(قـبلا توضـیح داده شـد کـه دستورهایی که داخل یک زوج کروشه {} قرار می‌گیرند در اصطلاح یک بلوک نامیده می‌شوند.) و در صورتیکه مقدار آن برابر بـا **false** باشد اجرای برنامه به بعد از بلوک **if** منتقل می‌شود. در این مثال همانطور که ملاحظه می‌نمایید، عبارت منطقی دسـتور **if** بشکل **if(myInt > 0)** است. در صورتیکه مقدار **myInt** بزرگتر از عدد صفر باشد، دستور داخل بلوک **if** اجرا می‌شـود و در غیر اینصورت اجرای برنامه به بعد از بلوک **if** منتقل می‌گردد.

دومین دستور **if** دراین برنامه بسیار شبیه به دستور اول است، با این تفاوت که در این دستور، دستور اجرایی **if** درون یک بلـوک قرار نگرفته است. در صورتیکه بخواهیم با استفاده از دستور **if** تنها یک دستورالعمل اجرا شود، نیازی به استفاده از بلوک برای آن دستورالعمل نمی‌باشد. استفاده از بلوک تنها زمانی ضروری است که بخواهیم از چندین دستور استفاده نماییم.

دستور ***if-else***

در بیشتر موارد از تصمیم‌گیری‌های دوگانه یا چندگانه استفاده می‌شود. در این نوع تصمیم‌گیری‌ها، دو یا چند شرط مختلف بررسی می‌شوند و در صورت **true** بودن یکی از آنها عمل مربوط به آن اجرا می‌گردد. سومین دستور **if** در این برنامه نشان دهنده یـک تصمیم‌گیری دوگانه است. در این حالت درصورتیکه عبارت منطقی دستور **if** برابر با **true** باشد دستور بعد از **if** اجرا می‌شـود و در غیر اینصورت دستور بعد از **else** به اجرا در می‌آید. در حقیقت در این حالت می‌گوئیم " اگر شرط **if** صحیح است دسـتورات مربوط به **if** را انجام بده و درغیر اینصورت دستورات **else** را اجرا کن".

فرم کلی دستور **if-else** بصورت زیر است :

```
if (boolean expression)
  {statements}
else
  {statements}
```

که در آن **boolean expression** عبارت منطقی است که صحت آن مورد بررسی قرار می‌گیرد و **statements** دستور یا دستوراتی است که اجرا می‌گردند.

دستور ***if-else if ... else*** *یا* ***if*** *تودرتو*

در صورتیکه نیاز باشد تا چندین حالت منطقی مورد بررسی قرار گیرد و دستورات مربوط بـه یکـی از آنهـا اجـرا شـود، از فـرم تصمیم‌گیری چندگانه استفاده می‌نماییم. این نوع استفاده از دستور **if** در اصطلاح به **if** تودرتـو **(Nested If)** معـروف اسـت چراکه در آن از چندین دستور **if** مرتبط به یکدیگر استفاده شده است. چهارمین دستور **if** در مثال ۱-۳ استفاده از **if** تودرتو را نشان می‌دهد. در این حالت نیز دستور با کلمه کلیدی **if** آغاز می‌گردد. شرطی بررسی شده و در صورت **true** بـودن دسـتورات مربوط به آن اجرا می‌گردد. اما اگر مقدار این عبارت منطقی **false** بود آنگاه شرطهای فرعـی دیگـری بررسـی مـی‌شـوند.ایـن شرطهای فرعی با استفاده از **else if** مورد بررسی قرار می‌گیرند. هر یک از این شرطها دارای عبارات منطقـی مربـوط بـه خـود هستند که در صورت **true** بودن عبارت منطقی دستورات مربوط به آنها اجرا می‌گردد و در غیـر اینـصورت شـرط بعـدی مـورد بررسی قرار می‌گیرد. باید توجه کنید که در ساختار **if** تودرتو تنها یکی از حالتها اتفاق می‌افتد و تنها یکی از شرطها مقدار **true** را بازمی‌گرداند.

فرم کلی **if** تودرتو بشکل زیر است :

```
if (boolean expression)
  {statements}
else if (boolean expression)
  {statements}
...
else
  {statements}
```

عملگرهای ***OR*** *و* ***AND*** *(|| و &&)*

نکته دیگری که باید در اینجا بدان اشاره کرد، نوع شرطی است که در عبارت منطقی دستور **if** آخر مورد استفاده قرار گرفته است. در این عبارت منطقی از عملگر **||** استفاده شده است که بیانگر **OR** منطقی است. عملگر **OR** زمانی مقدار **true** بازمی‌گردانـد که *حداقل یکی* از عملوندهای آن دارای مقدار **true** باشد. بعنوان مثـال در عبـارت (**myInt < 0 || myInt == 0**)، در صورتیکه مقدار متغیر **myInt** کوچکتر *یا* مساوی با صفر باشد، مقدار عبارت برابر با **true** است. نکته قابل توجه آنـست کـه در زبان **C#**، همانطور که در درس دوم به آن اشاره شد، دو نوع عملگر **OR** وجود دارد. یکی **OR** منطقـی کـه بـا **||** نمـایش داده می‌شود و دیگری **OR** معمولی که با **|** نشان داده می‌شود. تفاوت بین این دو نوع **OR** در آنست که **OR** معمولی هـر دو عملگـر

خود را بررسی می‌نماید اما **OR** منطقی تنها در صورتیکه عملگر اول آن مقدار **false** داشته باشد به بررسی عملگر دوم خود می‌پردازد.

عبارت منطقی (**myInt > 0 && myInt <= 10**) حاوی عملگر **AND** شرطی (**&&**) می‌باشد. این عبارت در صورتی مقدار **true** بازمی‌گرداند که هر دو عملوند **AND** دارای مقدار **true** باشند. یعنی در صورتیکه **myInt** هم بزرگتر از صفر باشد و هم کوچکتر از ۱۰، مقدار عبارت برابر با **true** می‌گردد. در مورد **AND** نیز همانند **OR** دو نوع عملگر وجود دارد. یکی **AND** معمولی (**&**) و دیگری **AND** شرطی (**&&**). تفاوت این دو نیز در آنست که **AND** معمولی (**&**) همیشه هر دو عملوند خود را بررسی می‌نماید ولی **AND** شرطی (**&&**) تنها هنگامی به بررسی عملوند دوم خود می‌پردازد که مقدار اولین عملوندش برابر با **true** باشد. عملگرهای منطقی (**||** و **&&**) را در اصطلاح عملگرهای میانبر (**short-circuit**) می‌نامند چراکه تنها در صورت لزوم عملوند دوم خود را بررسی می‌نمایند و از اینرو سریعتر اجرا می‌شوند.

*بررسی دستور **switch***

همانند دستور **if**، دستور **switch** نیز امکان تصمیم‌گیری را در یک برنامه فراهم می‌نماید.

*مثال ۲-۳ – دستورالعمل **switch***

```
using System;

class SwitchSelect
 {
  public static void Main()
  {
   string myInput;
   int myInt;
   begin:
   Console.Write("Please enter a number between 1 and 3: ");
   myInput = Console.ReadLine();
   myInt = Int32.Parse(myInput);
   // دستور switch بهمراه متغیری از نوع صحیح
   switch (myInt)
   {
    case 1:
      Console.WriteLine("Your number is {0}.", myInt);
      break;
    case 2:
      Console.WriteLine("Your number is {0}.", myInt);
      break;
    case 3:
      Console.WriteLine("Your number is {0}.", myInt);
      break;
    default:
      Console.WriteLine("Your number {0} is not between 1 and 3.", myInt);
      break;
   } //switch پایان بلوک
   decide:
   Console.Write("Type \"continue\" to go on or \"quit\" to stop: ");
   myInput = Console.ReadLine();
   // دستور switch بهمراه متغیری از نوع رشته‌ای
   switch (myInput)
   {
```

```
    case "continue":
      goto begin;
    case "quit":
      Console.WriteLine("Bye.");
      break;
    default:
      Console.WriteLine("Your input {0} is incorrect.", myInput);
      goto decide;
  } //switch پایان بلوک
 } //Main() پایان متد
} //SwitchSelect پایان کلاس
```

مثال ۲-۳ دو مورد استفاده از دستور **switch** را نشان می‌دهد. دستور **switch** بوسیله کلمه کلیدی **switch** آغاز شده و به دنبال آن عبارت دستور **switch** قرار می‌گیرد. عبارت دستور **switch** می‌تواند یکی از انواع زیر باشد : **sbyte, byte**, **short, ushort, int, uint, long, ulong, char, string, enum** (نوع **enum** در مبحث جداگانه‌ای مورد بررسی قرار خواهد گرفت.) در اولین دستور **switch** در مثال ۲-۳، عبارت دستور **switch** از نوع عددی صحیح **(int)** می‌باشد.

به دنبال دستور و عبارت **switch**، بلوک **switch** قرار می‌گیرد که در آن گزینه‌هایی قرار دارند که جهت منطبق بودن با مقدار عبارت **switch** مورد بررسی قرار می‌گیرند. هر یک از این گزینه‌ها با استفاده از کلمه کلیدی **case** مشخص می‌شوند. پس از کلمه کلیدی **case** خود گزینه قرار می‌گیرد و به دنبال آن ":" و سپس دستوری که باید اجرا شود. بعنوان مثال به اولین دستور **switch** در این برنامه توجه نمایید. در اینجا عبارت دستور **switch** از نوع **int** است. هدف از استفاده از دستور **switch** آنست که از بین گزینه‌های موجود در بلوک **switch**، گزینه‌ای را که مقدارش با مقدار عبارت **switch** برابر است پیدا شده و عمل مرتبط با آن گزینه اجرا شود. در این مثال مقدار متغیر **myInt** بررسی می‌شود. سپس اگر این مقدار با یکی از مقادیر گزینه‌های داخل بلوک **switch** برابر بود، دستور یا عمل مربوط به آن گزینه اجرا می‌گردد. توجه نمایید که در این مثال منظور ما از گزینه همان عدد پس از **case** است و منظور از دستور عبارتی است که پس از **":"** قرار گرفته است. بعنوان مثال، در دستور زیر :

```
case 1:
    Console.WriteLine("Your number is {0}.", myInt);
```

عدد ۱، گزینه مورد نظر ما و دستور **Console.WriteLine(...)**، عمل مورد نظر است. در صورتیکه مقدار **myInt** برابر با عدد ۱ باشد آنگاه دستور مربوط به **case 1** اجرا می‌شود که همان **Console.WriteLine("Your number is {0}.", myInt);** است. پس از منطبق شدن مقدار عبارت **switch** با یکی از **case** ها، بلوک **switch** باید خاتمه یابد که این عمل بوسیله استفاده از کلمه کلیدی **break**، اجرای برنامه را به اولین خط بعد از بلوک **switch** منتقل می‌نماید.

همانطور که در ساختار دستور **switch** مشاهده می‌نمایید، علاوه بر **case** و **break**، دستور دیگری نیز در داخل بلوک وجود دارد. این دستور یعنی **default**، برای زمانی مورد استفاده قرار می‌گیرد که هیچ یک از گزینه‌های بلوک **switch** با عبارت دستور **switch** منطبق نباشند. به عبارت دیگر درصورتیکه مقدار عبارت **switch** با هیچ یک از گزینه‌های **case** برابر نباشد، دستور مربوط به **default** اجرا می‌گردد. استفاده از این دستور در ساختار بلوک **switch** اختیاری است. همچنین قرار دادن دستور **break** پس از دستور **default** نیز اختیاری می‌باشد.

همانطور که قبلاً نیز گفته شد پس از هر دستور **case**، به منظور خاتمه دادن اجرای بلوک **switch** باید از یک **break** استفاده نمود. دو استثنا برای این موضوع وجود دارد. اول اینکه دو دستور **case** بدون وجود کد و دستورالعملی در بین آنها، پشت سر هم قرار گیرند و دیگری در زمانیکه از دستور **goto** استفاده شده باشد.