Microsoft® Visual Studio® C# آموزش برنامه نویسی

Microsoft® Visual Studio®

Microsoft® .NET

بخش اول

ASP.NET

نوشته شده توسط ایتروز

آدما مثل کتابا می مونن ... بعضی از اونا جلد زرکوب دارن بعضی از اونا جلد ضخیم دارن، بعضی از آدما هم جلد نازک. بعضی از اونا ترجمه می شن، بعضی از اونا همین طوری می مونن. بعضی از آدما تجدید چاپ می شن و بعضی ها هم فراموش می شن . . . از روی بعضی از آدما باید مشق نوشت و بعضی از آدما رو نخونده باید دور انداخت.

Microsoft® Visual Studio® C#

آموزش برنامه نویسی

عنوان

صفحه

عنوان

صفحه

مفهوم برنامه نویسی

قبل از توضیح در رابطه با برنامه نویسی برای درک بهتر مطالب می خواهیم شما را با فرآیند برنامه نویسی آشنا کرده و مفهوم آن را بررسی نمایم. چنانچه می دانیم کامپیوتر ابزاری است که کارها را با سرعت بالا و بر اساس دستورالعملی که برنامه نویس به آن داده انجام می دهد. اگر برنامه ای که برنامه نویس به کامپیوتر داده صحیح باشد نتیجه کار کامپیوتر دارای صحت و دقت بالایی خواهد بود. برای اینکه یک برنامه نوشته شده صحیح برای یک مسئله صحیح باشد باید مراحل مختلفی انجام گیرد.

۱.شناخت صحیح از خواسته مسئله

۲.تجزیه و تحلیل مسئله

۳.طراحی فرآیند حل مسئله

۱-۱ شناخت صحیح از خواسته مسئله

وقتی مسئله ای طرح می شود باید با دقت کافی خواسته های مسئله را بررسی کرد و تعیین کرد که چه کاری باید انجام گیرد.

۱-۲ تحلیل و تجزیه مسئله

در این مرحله باید تمام عواملی که در مسئله دخالت دارند بررسی شوند این عوامل به صورت زیر هستند:

داده ها: ورودی های مسئله و مفروضات مسئله هستند.

مجهول ها: داده ها به همراه مفروضات پس از پردازش به چه نتایجی باید برسند.

خروجی: نتایج عملیات باید شامل چه چیزهایی باشد و به چه صورت در اختیار کاربر قرار گیرد.

۱-۳ طراحی فرآیند حل مسئله

پس از شناخت مسئله و تجزیه و تحلیل آن باید طرح های منطقی برای حل مسئله ارائه کرد و باید بدانیم که برای حل مسئله نمی توان یک راه ارائه کرد بلکه برای حل مسئله راه های متفاوتی پیشنهاد می شود که با بررسی روشهای مختلف باید روشی را انتخاب کرد که از نظر اجرا، سرعت و حجم حافظه بهینه باشد. برای اینکه بتوانیم قبل از نوشن یک برنامه از خطا ها و اشکالات آن آگاه شویم و همچنین در وقت و هزینه صرفه جویی نمایم باید آن را روی کاغذ تجزیه و تحلیل نمایم برای اینکار از الگوریتم استفاده می نمایم.

۱-۴ الگوریتم

کارهایی که ما طبق عادت انجام می دهیم عموما برای انجام آنها الگوریتمی را اجرا می کنیم. برای مثال شما برای رفتن به محل کار از منزل ابتدا: لباس می پوشید ، خود را به ایستگاه اتوبوس می رسانید ، پس از سوار شدن ; در ایستگاه مورد نظر پیاده می شوید ، سوار تاکسی می شوید و در مقابل محل کارتان پیاده می شوید. اگر کمی فکر کنید کارهای دیگری در طول روز انجام می دهید نیز دارای الگوریتم خاصی هستند. واژه الگوریتم از نام دانشمند ایرانی به نام خوارزمی گرفته شده است و تعریف دقیق آن به صورت زیر می باشد:

به مجموعه دستورالعمل هایی که مراحل مختلف انجام کار را به زبان دقیق با جزیات کافی بیان می کند به طوری که از ترتیب مراحل انجام کارها و شرط پایان عملیات در آن کاملا مشخص باشد الگوریتم گفته می شود. اما در الگوریتم قوانینی حاکم است که باید آنها را رعایت کرد:

۱-۴-۱ آغاز

الگوریتم باید از جایی شروع شود، به عبارت دیگر برای انجام هر کاری به یک نقطه شروع نیاز دارد.

۱-۴-۲ زبان دقیق

جملات الگوریتم باید بدون ابهام و دقیق باشد، در مثال" به محل کار" جمله فرد از منزل خارج شده و در ایستگاه اتوبوس سوار اتوبوس می شود جمله ای مبهم است و باید به این صورت باشد:

"در ایستگاه اتوبوس ، سوار اتوبوس شماره ۲۱۲ می شوید"

۱-۴-۳ جزیات کافی

چون جملات الگوریتم را باید ماشین انجام دهد باید با جزیات کافی ذکر شود یعنی در مثال به جای، پس از سوار شدن در ایستگاه مورد نظر پیاده شد باید این گونه نوشت:

"پس از سوار شدن در ایستگاه مورد نظر باید توسط زنگ زدن ، راننده را آگاه کرد که بایستد و سپس پیاده شود"

۱-۴-۴ ترتیب مراحل

در فرآیند اکثر مسائل ترتیب انجام عملیات باید رعایت شود برای مثال سوار شدن به تاکسی باید پس از پیاده شدن در اتوبوس انجام گیرد.

۱-۴-۵ خاتمه پذیر بودن

برای اینکه کاری انجام شود باید در جایی انجام کار تمام شده و مراحل خاتمه پیدا کند به عبارتی الگوریتم باید نقطه پایان داشته باشد.

۱-۵ چرا از الگوریتم استفاده کنیم؟

اگر چه طبق عادت، ما بیشتر کار ها را انجام می دهیم اما همه آنها بر اساس یک الگوریتم صورت می گیرند به دلیل اینکه کارهای عادی بسیار ساده هستند و الگوریتم آنها در ذهن ما جا گرفته است. اما در پروژه هایی که نیاز به فکر دارد و مدتها طول می کشد که برخی از قسمت های آن حل شود باید راه حلهای مسئله را مکتوب نمود. زیرا انسان فرآموش کار است و در صورت نیاز مجدد به آن راه حل، ممکن است آن را فرآموش کرده و یا اینکه برنامه ای که ساخته شد اگر اشکال پیدا کند و طراح اصلی آن حضور نداشته باشد در صورت نبود الگوریتم کار اشکال یابی بسیار پیچیده و سخت خواهد شد.

۱-۶ انواع جملات و دستورالعمل ها به صورت کلی و پایه

دستورالعمل های ورودی: این دستورالعمل ها برای دریافت داده به کار می رود
مثال: a,b,c را بگیر

دستورالعمل های محاسباتی و انتسابی: این دستورالعمل ها برای انتساب یک مقدار به متغیر یا محاسبه یک فرمول به کار می رود
مثال: A را برابر ۲ قرار بده

دستورالعمل های شرطی: این دستورالعمل ها با اگر شروع می شوند و در صورت برقرار بودن شرایط خاصی، عملی صورت می گیرد.
مثال: اگر a>b باشد c را برابر a قرار بده

دستورالعمل های خروجی: دستورالعمل های خروجی برای تعیین نتایج و تحویل آنها مورد استفاده قرار می گیرد.

نوشتن الگوریتم جمع دو عدد یک رقمی

۱. شروع

۲. صفر را در s قرار بده

۳. a و b را بگیر

۴. c=a+b

۵. s را چاپ کن

۶. پایان

مقدمه ای بر سی شارپ

زبان برنامه نویسی سی شارپ به رهبری Anderes Hejlsberg و Scott Wiltamuth در شرکت مایکروسافت بر پایه تکنولوژی .Net طراحی گردید.

اگر شما با برنامه های JAVA , C , C++ آشنایی دارید قادر خواهید بود زبان C# را به سرعت فرآگیرید زیرا سی شارپ مشتق شده از این سه زبان است.

زبان سی شارپ یک زبان رویدادگرا ، کاملا شی گرا، زبان برنامه نویسی ویژال است که برنامه های آن در یک محیط توسعه یافته مجتمع IDE ایجاد می شود و قابلیت برنامه نویسی تحت ویندوز و تحت وب را دارا می باشد.

چند نکته در رابطه با سی شارپ

۱. به حروف بزرگ و کوچک حساس می باشد.
۲. در آخر هر دستور باید یک سیمی کالون (;) گذاشت.
۳. جهت توضیحات در هر خط از // استفاده می شود.
۴. استفاده از کاراکتر های /* در اول و در */ آخر توضیحات می توانند چند خطی باشند.

۱-۲ ویژگی Object Oriented Featuer

روش های برنامه نویسی معمولا به دو دسته کلی (برنامه نویسی تابع گرا) و (برنامه نویسی شی گرا) تقسیم می شوند.

برنامه نویسی تابع گرا: در این روش برنامه نویسی مبتنی بر (Procedure) و تابع می باشد، در این روش معمولا یک تابع اصلی یا بدنه اصلی در برنامه وجود داردکه بقیه توابع یا رویه ها از داخل آن صدا زده می شوند و برنامه از بالا به پایین خط به خط اجرا می گردد و در صورت فراخوانی تابع، برنامه مربوط به تابع خط به خط اجرا شده و کنترل مجداداً به برنامه اصلی بر می گردد.

برنامه نویسی شی گرا: در برنامه نویسی شی گرا به جای تابع از مفاهیم کلاس (class) و خصوصیت (properties) و شی (object) استفاده می شود.

۲-۲ مفاهیم پایه تکنولوژی شی

به اطراف خود نگاه کنید شاهد شی هایی خواهید بود، برخی از این شی ها (کامپیوتر ، میز ، صندلی) حتی در محیطی بزرگتر مثل دنیا هم شی هایی وجود دارد (مردم ، حیوانات ، گیاهان ، هواپیما) و بسیاری دیگر، لازم به ذکر است تفکر انسان بر اساس شی است (تلفن ، خانه ها ، گوشی های همراه) این ها تعداد کمی از بیشمار شی ها هستند که در زندگی روزمره شاهد آن ها هستیم.
می توان شی ها را به دو دسته متحرک و غیر متحرک تقسیم نمود.

شی های متحرک : زنده هستند و می توانند حرکت کنند و کاری انجام دهند
شی های غیر متحرک : در محیط خود حرکت نمی کنند

اما این دو نوع شی (متحرک و غیر متحرک) در برخی از چیزها با هم مشترک هستند مثلا همه آنها دارای صفات (شکل ، رنگ ، وزن) بوده و از خود رفتاری را نشان می دهند مثلا (توپ می غلتد ، بالا و پاین می پرد، بچه گریه می کند، می خوابد، راه می رود، اتومبیل شتاب می گیرد) .
انسان با بررسی صفات و رفتار اشیاء مطالبی در ارتباط با آنها یاد می گیرد. شی های مختلف می توانند صفات و رفتار مشابهی داشته باشند. مثلا بین کودکان و بزرگ سالان.

۲-۳ مدل های شی گرا OOD (Object Oriented Design)

به طور کل می توان گفت روشی که افراد برای توصیف شی ها در دنیای واقعی به کار می برند. طراحی شی گرا یک روش طبیعی و ذاتی در نمایش فرآیند طراحی نرم افزار بنام مدل سازی شی ها توسط صفات، رفتار و رابطه موجود در میان شی ها همانند شی ها در دنیای واقعی است.
همچنین OOD ارتباط ما بین شی ها را مدل سازی می کند. همانند فردی که پیغامی به شخص دیگری می فرستد، شی ها نیز از طریق پیغام ارتباط برقرار می کنند. یک شی حساب بانکی می تواند پیغامی برای کاهش موجودی دریافت کند چرا که مشتری مقداری از پول خود را از حساب برداشت کرده است.

مدل OOD مبادرت به کپسوله سازی (encapsulates) صفات و عملیات (رفتار) در شی ها می کند. سر انجام صفات و رفتار یک شی با هم گره می خورند. شی ها از خصیصه پنهان سازی اطلاعات (Information Hiding) برخوردار هستند. به این معنی که شی ها از نحوه برقراری ارتباط با یک شی دیگر از طریق یک واسط (رابط) مناسب مطلع هستند. اما معمولا از نحوه پیاده سازی شی های دیگر اطلاعی ندارند. معمولا جزیات پیاده سازی در درون خود شی پنهان می باشد.

۲-۴ ویژگی GUI

محیط ویژال استودیو برای اینکه به صورت گرافیکی با برنامه نویس ارتباط برقرار کند دارای ویژگی رابط گرافیکی می باشد که به آن GUI یا Graphical User Interface گفته می شود.

۲-۵ ویژگی IDE

علاوه بر ویژگی GUI محیط ویژال استودیو از جمله محیط های توسعه یافته مجتمع IDE می باشد که به معنی Interface Design Enhandced می باشد و به برنامه نویس امکان می دهد که برنامه های تحت ویندوز خود را بدون نیاز به استفاده از برنامه های کاربردی دیگر ایجاد، اجرا و خطا یابی نماید.

۲-۶ ویژگی RAD

ویژگی دیگر ویژال استودیو RAD است به معنی Rapid Application Development می باشد که برنامه نویس می تواند برنامه کاربردی خود را با استفاده از ویزارد ، ابزارها و... به سرعت توسعه دهد.

۲-۷ ویژگی رسیدگی به خطا (Error Handling)

یکی از جنبه هایی که در نحوه کار با یک زبان برنامه نویسی مد نظر گرفته می شود نحوه کشف ، تصحیح و برخورد با اشتباهات و خطاهایی است که در هنگام طراحی یا اجرای برنامه رخ میدهد.

۲-۸ برنامه نویسی ساختار یافته (Structural Programming)

برنامه هایی که در سی شارپ می توان نوشت ساختار یافته است یعنی اینکه در محیط آن می توان متغیرهایی از نوع متفاوت داشت و برنامه های بزرگ و پیچیده را به قسمت های کوچک تقسیم نمود بطوری که هر قسمت وظیفه خاصی انجام دهد.

انواع داده (Data Type)

در هر زبان برنامه نویسی ساخت یافته، برای پردازش اطلاعات نیاز به استفاده از انواع مختلف داده می باشد.
حال می خواهیم انواع داده ها را برسی نمایم.

نوع	مقدار	محدوده مقادیر	مقدار حافظه مورد استفاده (بایت)
byte	اعداد صحیح	۰ تا ۲۵۵	۱ Byte
sbyte	اعداد صحیح	-۱۲۸ تا ۱۲۷	۱ Byte
short	اعداد صحیح	-۳۲،۷۶۸ تا ۳۲،۷۶۷	۲ Byte
ushort	اعداد صحیح	۰ تا ۶۵،۵۳۵	۲ Byte
int	اعداد صحیح	-۲،۱۴۷،۴۸۳،۶۴۸ تا ۲,۱۴۷،۴۸۳،۶۴۷	۴ Byte
uint	اعداد صحیح	۰ تا ۴،۲۹۴،۹۶۷،۲۹۵	۴ Byte
long	اعداد صحیح	-۹،۲۲۳،۳۷۲،۰۳۶،۸۵۴،۷۷۵،۸۰۸ تا ۹،۲۲۳،۳۷۲،۰۳۶،۸۵۴،۷۷۵،۸۰۷	۸ Byte
ulong	اعداد صحیح	۰ تا ۱۸،۴۴۶،۷۴۴،۰۷۳،۷۰۹،۵۵۱،۶۱۵	۸ Byte
float	اعداد اعشاری	-۳,۴۰۲۸۲۳e۳۸ تا ۳,۴۰۲۸۲۳e۳۸	۴ Byte
double	اعداد اعشاری	-۱,۷۹۷۶۹۳۱۳۴۸۶۲۳۲e۳۰۸ تا ۱,۷۹۷۶۹۳۱۳۴۸۶۲۳۲e۳۰۸	۸ Byte
decimal	اعداد اعشاری	-۷۹۲۲۸۱۶۲۵۱۴۲۶۴۳۳۷۵۹۳۵۴۳۹۵۰۳۳۵ تا ۷۹۲۲۸۱۶۲۵۱۴۲۶۴۳۳۷۵۹۳۵۴۳۹۵۰۳۳۵	۱۶ Byte
char	یونیکد یک کاراکتر	----------	۲ Byte
string	های کاراکتر رشته ای از یونیکد.	----------	۲۰+ Byte
bool	مقادیر منطقی	True or False.	۱ Byte
object	کلاس ها و اشیاء	----------	۸+ Byte

مثال:

C#

```
……
int i = ۰;
double d = ۰;

i = ۱۰;
d = i;       // An implicit conversion

d = ۳,۵;
i = (int) d;  // An explicit conversion, or "cast"

……
```

متغیر ها (Variables)

متغیرها، نگهدارنده هایی هستند که اطلاعات برنامه را به صورت موقت ذخیره می کنند. به عبارت دیگر یک متغیر، مکانی از حافظه را برای یک مقدار نگهداری می کند. مقداری که داخل متغیر قرار می گیرد قابل تغییر است و وقتی مقداری داخل آن قرار گیرد مقدار قبلی از بین می رود. همانطور که گفته شد متغیر ها مکانی از حافظه را رزرو می کنند بنابراین برای شناسایی آن ناحیه از حافظه باید نامی برای آن در نظر گرفت به عبارت دیگر برای هر متغیر باید نامی متناسب با محتوای آن در نظر گرفت و توجه داشته باشید برای دو متغیر نام یکسان انتخاب نشود.
برای نامگذاری متغیر ها می توان:

۱. از ترکیبی از حروف a تا z یا A تا Z استفاده کرد.
۲. ارقام و خط ربط(_) به طوری که اولین کاراکتر آن رقم نباشد.
۳. نام متغیر می تواند هر طولی باشد اما ۳۱ کاراکتر اول آن مورد استفاده قرار می گیرد.

اسامی غیر مجاز: ۱test hight!hight grade:۱ .pcx

اسامی مجاز : s_۱ sum test۲۳ count

۳-۱ تعریف متغیر

[Public یا Private] نوع داده نام متغیر ;

۳-۲ تعریف ثوابت

ثوابت ها، مقادیری هستند که در برنامه وجود دارند ولی قابل تغیر نیستند،برای نامگذاری در ثوابت باید از قانون نامگذاری متغیر ها استفاده کنید. مقداری که برای ثابت تعیین می شود نوع آن را مشخص می کند.
برای تعریف ثوابت ها دو روش وجود دارد:

استفاده از دستور #define

#define < نام ثابت> <مقدار>

استفاده از دستور const

const <نوع داده> <نام ثابت> = <مقدار>;

انواع عملگرها (Operators)

عملگر	نشانه
عملگر های محاسباتی	پایین ترین تقدم + - * ٪ / ++ -- بالاترین تقدم
عملگر های رابطه ای	< > <= >= == != Is as
عملگر های منطقی	& ^ | && || ?:
عملگر های ترکیبی	= += -= *= /= ٪= &= |= ^= <<= >>= ??

نکته: پرانتز تقدم عملگر های خود را بالا می برد

۴-۱ عملگر های محاسباتی

عملگر ❊

```
// cs_operator_mult.cs
using System;
class MainClass
{
   static void Main()
   {
      Console.WriteLine(۵ * ۲);
      Console.WriteLine(-.۵ * .۲);
      Console.WriteLine(-.۵m * .۲m); // decimal type
   }
}
```

خروجی برنامه

```
۱۰
-۰,۱
-۰,۱۰
```

عملگر /

.....

```
// cs_operator_division.cs
using System;
class MainClass
{
   static void Main()
   {
      Console.WriteLine(-۵/۲);
      Console.WriteLine(-۵,۰/۲);
   }
}
```

.....

خروجی برنامه

```
-۲
-۲.۵
```

عملگر ٪

.....

```
// cs_operator_modulus.cs
using System;
class MainClass
{
   static void Main()
   {
      Console.WriteLine(۵ % ۲);       // int
      Console.WriteLine(-۵ % ۲);      // int
      Console.WriteLine(۵,۰ % ۲,۲);   // double
```

```
        Console.WriteLine(۵,۰m % ۲,۲m); // decimal
        Console.WriteLine(-۵,۲ % ۲,۰);  // double
    }
}
```

.....

خروجی برنامه

```
۱
-۱
۰,۶
۰,۶
-۱,۲
```

عملگر +

.....

```
// cs_operator_plus.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(+۵);        // unary plus
        Console.WriteLine(۵ + ۵);     // addition
        Console.WriteLine(۵ + .۵);    // addition
        Console.WriteLine("۵" + "۵"); // string concatenation
        Console.WriteLine(۵,۰ + "۵"); // string concatenation
        // note automatic conversion from double to string
    }
}
```

.....

خروجی برنامه

۵

۱۰

۵,۵

۵۵

۵۵

عملگر –

.....

```
// cs_operator_minus.cs
using System;
class MainClass
{
   static void Main()
   {
      int a = ۵;
      Console.WriteLine(-a);
      Console.WriteLine(a - ۱);
      Console.WriteLine(a - .۵);
   }
}
```

.....

خروجی برنامه

-۵

۴

۴,۵

۲-۴ عملگر های رابطه ای

عملگر <

.....

```
// cs_operator_greater_than.cs
using System;
class MainClass
{
   static void Main()
   {
      Console.WriteLine(١,١ > ١);
      Console.WriteLine(١,١ > ١,١);
   }
}
```

.....

خروجی برنامه

```
True
False
```

عملگر >

.....

```
// cs_operator_less_than.cs
using System;
class MainClass
{
   static void Main()
   {
      Console.WriteLine(١ < ١,١);
      Console.WriteLine(١,١ < ١,١);
   }
}
```

.....

خروجی برنامه

```
True
False
```

عملگر = >

```
// cs_operator_greater_than_or_equal.cs
using System;
class MainClass
{
   static void Main()
   {
      Console.WriteLine(١,١ >= ١);
      Console.WriteLine(١,١ >= ١,١);
   }
}
```

خروجی برنامه

```
True
True
```

عملگر = <

```
// cs_operator_less_than_or_equal.cs
using System;
class MainClass
{
   static void Main()
   {
      Console.WriteLine(١ <= ١,١);
      Console.WriteLine(١,١ <= ١,١);
   }
}
```

خروجی برنامه

```
True
```

True

عملگر = =

.....

```
// cs_operator_equality.cs
using System;
class MainClass
{
   static void Main()
   {
      // Numeric equality: True


Console.WriteLine((۲ + ۲) == ۴);

      // Reference equality: different objects,
      // same boxed value: False.
      object s = ۱;
      object t = ۱;
      Console.WriteLine(s == t);

      // Define some strings:
      string a = "hello";
      string b = String.Copy(a);
      string c = "hello";

      // Compare string values of a constant and an instance: True
      Console.WriteLine(a == b);

      // Compare string references;
      // a is a constant but b is an instance: False.
      Console.WriteLine((object)a == (object)b);

      // Compare string references, both constants
      // have the same value, so string interning
      // points to same reference: True.
      Console.WriteLine((object)a == (object)c);
   }
```

```
}
```

.....

خروجی برنامه

```
True
False
True
False
True
```

عملگر = !

.....

```
// cs_operator_inequality.cs
using System;
class MainClass
{
   static void Main()
   {
      // Numeric inequality:
      Console.WriteLine((٢ + ٢) != ٤);

      // Reference equality: two objects, same boxed value
      object s = ١;
      object t = ١;
      Console.WriteLine(s != t);

      // String equality: same string value, same string objects
      string a = "hello";
      string b = "hello";
```

```
        // compare string values
        Console.WriteLine(a != b);

        // compare string references
        Console.WriteLine((object)a != (object)b);
    }
}
```

.....

خروجی برنامه

```
False
True
False
False
```

عملگر is

.....

```
// cs_keyword_is.cs
// The is operator.
using System;
class Class١
{
}
class Class٢
{
}

class IsTest
{
    static void Test(object o)
    {
        Class١ a;
        Class٢ b;

        if (o is Class١)
```

```
        {
            Console.WriteLine("o is Class۱");
            a = (Class۱)o;
            // Do something with "a."
        }
        else if (o is Class۲)
        {
            Console.WriteLine("o is Class۲");
            b = (Class۲)o;
            // Do something with "b."
        }
        else
        {
            Console.WriteLine("o is neither Class۱ nor Class۲.");



    }
     }
     static void Main()
     {
        Class۱ c۱ = new Class۱();
        Class۲ c۲ = new Class۲();
        Test(c۱);
        Test(c۲);
        Test("a string");
     }
}
```

.....

خروجی برنامه

```
o is Class۱
o is Class۲
o is neither Class۱ nor Class۲.
```

عملگر as

```
// cs_keyword_as.cs
// The as operator.
using System;
class Class۱
{
}

class Class۲
{
}

class MainClass
{
   static void Main()
   {
      object[] objArray = new object[۶];



      objArray[۰] = new Class۱();
      objArray[۱] = new Class۲();
      objArray[۲] = "hello";
      objArray[۳] = ۱۲۳;
      objArray[۴] = ۱۲۳,۴;
      objArray[۵] = null;

      for (int i = ۰; i < objArray.Length; ++i)
      {
         string s = objArray[i] as string;
         Console.Write("{۰}:", i);
         if (s != null)
         {
            Console.WriteLine("'" + s + "'");
         }
         else
         {
            Console.WriteLine("not a string");
```

```
        }
      }
   }
}
```

.....

خروجی برنامه

```
۰:not a string
۱:not a string
۲:'hello'
۳:not a string
۴:not a string
۵:not a string
```

۳-۴ عملگر های منطقی

عملگر &

.....

```
// cs_operator_ampersand.cs
using System;
class MainClass
{
   static void Main()
   {
      Console.WriteLine(true & false); // logical and
```

```
        Console.WriteLine(true & true);  // logical and
        Console.WriteLine("٠x{٠:x}", ٠xf٨ & ٠x٣f);  // bitwise and
    }
}
```

.....

خروجی برنامه

```
False
True
٠x٣٨
```

عملگر & &

.....

```
// cs_operator_logical_and.cs
using System;
class MainClass
{
  static bool Method١()
  {
    Console.WriteLine("Method١ called");
    return false;
  }

  static bool Method٢()
  {
```

```
    Console.WriteLine("Method۲ called");
    return true;
  }

  static void Main()
  {
    Console.WriteLine("regular AND:");
    Console.WriteLine("result is {۰}", Method۱() & Method۲());
    Console.WriteLine("short-circuit AND:");
    Console.WriteLine("result is {۰}", Method۱() && Method۲());
  }
}
```

.....

خروجی برنامه

```
regular AND:
Method۱ called
Method۲ called
result is False
short-circuit AND:
Method۱ called
result is False
```

عملگر |

.....

```
// cs_operator_OR.cs
using System;
class MainClass
{
   static void Main()
   {
      Console.WriteLine(true | false);  // logical or
      Console.WriteLine(false | false); // logical or
      Console.WriteLine("۰x{۰:x}", ۰xf۸ | ۰x۳f);   // bitwise or
   }
}
```

خروجی برنامه

```
True
False
۰xff
```

عملگر ||

```
// cs_operator_short_circuit_OR.cs
using System;
class MainClass
{
   static bool Method۱()
   {
      Console.WriteLine("Method۱ called");
      return true;
   }

   static bool Method۲()
   {
```

```
        Console.WriteLine("Method۲ called");
        return false;
    }

    static void Main()
    {
        Console.WriteLine("regular OR:");
        Console.WriteLine("result is {۰}", Method۱() | Method۲());
        Console.WriteLine("short-circuit OR:");
        Console.WriteLine("result is {۰}", Method۱() || Method۲());
    }
}
```

.....

خروجی برنامه

```
regular OR:
Method۱ called
Method۲ called
result is True
short-circuit OR:
Method۱ called
result is True
```

عملگر ^

.....

```
// cs_operator_bitwise_OR.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(true ^ false);  // logical exclusive-or
        Console.WriteLine(false ^ false); // logical exclusive-or
        // Bitwise exclusive-or:
        Console.WriteLine("۰x{۰:x}", ۰xf۸ ^ ۰x۳f);
    }
```

```
}
```

.....

خروجی برنامه

```
True
False
۰xc۷
```

عملگر :?

.....

```
// cs_operator_conditional.cs
using System;
class MainClass
{
   static double sinc(double x)
   {
      return x != ۰,۰ ? Math.Sin(x)/x : ۱,۰;
   }


   static void Main()
   {
      Console.WriteLine(sinc(۰,۲));
      Console.WriteLine(sinc(۰,۱));
      Console.WriteLine(sinc(۰,۰));
   }
}
```

.....

خروجی برنامه

```
۰,۹۹۳۳۴۶۶۵۳۹۷۵۳۰۶
۰,۹۹۸۳۳۴۱۶۶۴۶۸۲۸۲
۱
```

۳-۴ عملگر های ترکیبی

عملگر =

.....

```
// cs_operator_assignment.cs
using System;
class MainClass
{
   static void Main()
   {
      double x;
      int i;
      i = ۵; // int to int assignment
      x = i; // implicit conversion from int to double
      i = (int)x; // needs cast
      Console.WriteLine("i is {۰}, x is {۱}", i, x);
      object obj = i;
      Console.WriteLine("boxed value = {۰}, type is {۱}",
            obj, obj.GetType());


     i = (int)obj;
      Console.WriteLine("unboxed: {۰}", i);
   }
}
```

.....

خروجی برنامه

```
i is ۵, x is ۵
boxed value = ۵, type is System.Int۳۲
unboxed: ۵
```

عملگر= +

```
// cs_operator_addition_assignment.cs
using System;
class MainClass
{
   static void Main()
   {
      //addition
      int a = ۵;
      a += ۶;
      Console.WriteLine(a);

      //string concatenation
      string s = "Micro";
      s += "soft";
      Console.WriteLine(s);
   }
}
```

خروجی برنامه

```
۱۱
Microsoft
```

عملگر = -

```
// cs_operator_subtraction_assignment.cs
using System;
class MainClass
{
   static void Main()
   {
      int a = ۵;
```

```
        a -= ۶;
        Console.WriteLine(a);
    }
}
```

.....

خروجی برنامه

-۱

عملگر = *

.....

```
// cs_operator_multiplication_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = ۵;
        a *= ۶;
        Console.WriteLine(a);
    }
}
```

.....

خروجی برنامه

۳۰

عملگر = /

.....

```
// cs_operator_division_assignment.cs
using System;
class MainClass
{
```

```
    static void Main()
    {
        int a = ۵;
        a /= ۶;
        Console.WriteLine(a);
        double b = ۵;
        b /= ۶;
        Console.WriteLine(b);
    }
}
```

.....

خروجی برنامه

۰

۰,۸۳۳۳۳۳۳۳۳۳۳۳۳۳

عملگر ٪ =

.....

```
// cs_operator_modulus_assignment.cs
using System;
class MainClass
{
```

```
    static void Main()
    {
        int a = ۵;
        a %= ۳;
        Console.WriteLine(a);
    }
}
```

.....

خروجی برنامه

۲

عملگر = &

.....

```
// cs_operator_and_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = ۰x۰c;
        a &= ۰x۰۶;
        Console.WriteLine("۰x{۰:x۸}", a);
        bool b = true;
        b &= false;
        Console.WriteLine(b);
    }
}
```

.....

خروجی برنامه

```
۰x۰۰۰۰۰۰۴
False
```

عملگر =|

.....

```
// cs_operator_or_assignment.cs
using System;
class MainClass
{
   static void Main()
   {
      int a = ۰x۰c;
      a |= ۰x۰۶;
      Console.WriteLine("۰x{۰:xΛ}", a);
      bool b = true;
      b |= false;
      Console.WriteLine(b);
   }
}
```

.....

خروجی برنامه

```
۰x۰۰۰۰۰۰e
True
```

عملگر =^

.....

```
// cs_operator_xor_assignment.cs
```

```
using System;
class MainClass
{
   static void Main()
   {
      int a = ۰x۰c;
      a ^= ۰x۰۶;
      Console.WriteLine("۰x{۰:xʌ}", a);
      bool b = true;
      b ^= false;
      Console.WriteLine(b);
   }
}
```

.....

خروجی برنامه

```
۰x۰۰۰۰۰۰a
True
```

عملگر = <<

.....

```
// cs_operator_left_shift_assignment.cs
using System;
class MainClass
{
   static void Main()
   {
      int a = ۱۰۰۰;
      a <<= ۴;
      Console.WriteLine(a);
   }
}
```

.....

خروجی برنامه

۱۶۰۰۰

عملگر = >>

.....

```
// cs_operator_right_shift_assignment.cs
using System;
class MainClass
{
   static void Main()
   {
      int a = ۱۰۰۰;
      a >>= ۴;
      Console.WriteLine(a);
   }
}
```

.....

خروجی برنامه

۶۲

عملگر ؟؟

.....

```
// nullable_type_operator.cs
using System;
class MainClass
{
   static int? GetNullableInt()
   {
      return null;
   }
```

```
    static string GetStringValue()
    {
        return null;
    }

    static void Main()
    {
        // ?? operator example.
        int? x = null;

        // y = x, unless x is null, in which case y = -١.
        int y = x ?? -١;

        // Assign i to return value of method, unless
        // return value is null, in which case assign
        // default value of int to i.
        int i = GetNullableInt() ?? default(int);

        string s = GetStringValue();
        // ?? also works with reference types.
        // Display contents of s, unless s is null,
        // in which case display "Unspecified".
        Console.WriteLine(s ?? "Unspecified");
    }
}
```

دستورات

تمامی دستورات به صورت یک ترکیب (syntax) که شامل دستور ، شرط ها و بلاک های دستوری تشکیل شده است.

۵-۱ دستورات شرطی

ساختار دستور شرطی if

همانطور که گفته شد دستور IF یک دستور شرطی می باشد یعنی اگر شرطی برقرار باشد این دستور اجرا می شود فرض کنید می خواهیم شرط را ۲>۱ بگذاریم ، اگر شرط درست باشد دستورات داخل شرط اجرا می شود در غیر این صورت دستورات داخل شرط اجرا نمی شود. حال که ۲ بزرگتر از ۱ است پس شرط اجرا شده و دستورات اجرا می شود اما اگر اینگونه بود ۲>۱ شرط اجرا نمی شد و دستورات داخل آن هم اجرا نمی شدند. و برنامه از شرط رد می شد.

```
IF (شرط)
دستور ;
```

حال می خواهیم نوعی دیگر از دستور IF را برسی نمایم در این نوع ما می توانیم دستور دیگری به شرط اضافه نمایم تا در صورت نادرست بودن شرط آن اجرا شود. یعنی اگر شرط ما ۱>۲ بود در این صورت مثل مثال بالا شرط اجرا می شود و دستور داخل انجام می شود اما اگر شرط این گونه بود ۲>۱ در این صورت از دستور Else استفاده کرده و دستورات داخل آن اجرا می شود.

```
IF (Condition ۱)
{

دستور ;

}
[Elese [IF (Condition ۲)]]
{

دستور ;

}
```

مثال دستور IF

.....

```
if (Condition_۱)
        {
           // Statement_۱;
        }
        else if (Condition_۲)
        {
           // Statement_۲;
        }
        else if (Condition_۳)
        {
           // Statement_۳;
        }
        else
        {
           // Statement_n;
        }
```

.....

```
int x = ۱۲;

if (x > ۱۰)
   if (y > ۲۰)
      Console.Write("Statement_۱");
   else
      Console.Write("Statement_۲");
```

.....

```
int x = ۱۲;
int y = ۱۸;

if (x > ۱۰)
   if (y > ۲۰)
```

```
    Console.Write("Statement_۱");
  else
    Console.Write("Statement_۲");
```

ساختار دستور شرطی Switch

دستور شرطی Switch برای شرط های طولانی می باشد که با ساختار آن آشنا خواهید شد.

```
Switch (عبارت)
{
  case <مقدار یک>:
<دستورات>;
Break;

case <مقدار دو>:
<دستورات>;
Break;

case <مقدار سه>:
<دستورات>;
Break;
}
```

مثال دستور Switch

.....

```
int caseSwitch = ۱;
switch (caseSwitch)
{
   case ۱:
      Console.WriteLine("Case ۱");
      break;
   case ۲:
```

```
        Console.WriteLine("Case ۲");
        break;
    default:
        Console.WriteLine("Default case");
        break;
}
```

۵-۲ دستورات حلقه (Loop)

در برنامه نویسی بارها پیش آمده که بخواهیم اعدادی را پشت سر هم یا به صورت زوج ، فرد ، پنج رقم و ... بنویسیم. برای این کار ساده ترین راه این است که از یک حلقه استفاده کنیم. حلقه دارای نشانه هایی می باشد که آنها را برسی می نمایم و به آنها می پردازیم.

ساختار حلقه تکرار for

```
گام حرکت ; شرط حلقه ; مقدار اولیه = اندیس حلقه
For ( ;;)
{

دستورات;


}
```

مثال دستور For

.....

```
class ForLoopTest
{
    static void Main()
    {
        for (int I = ۱; I <= ۵; i++)
        {
            Console.WriteLine(i);
        }
    }
}
/*
خروجی:
۱
```

۲
۳
۴
۵
*/

ساختار حلقه تکرار while

while (شرط)
{
دستور ;
}

مثال دستور While

.....

```
// statements_while.cs
using System;
class WhileTest
{
  public static void Main()
  {
    int n = ۱;

    while (n < ۶)
    {
      Console.WriteLine("Current value of n is {۰}", n);
      n++;
    }
  }
}
```

خروجی برنامه

Current value of n is ۱
Current value of n is ۲
Current value of n is ۳
Current value of n is ۴
Current value of n is ۵

ساختار حلقه تکرار Do While

```
Do
{
دستور;
} while (شرط);
```

مثال دستور Do While

.....

```
public class TestDoWhile
{
   public static void Main ()
   {
      int x = ۰;
      do
      {
         Console.WriteLine(x);
         x++;
      } while (x < ۵);
   }
}
/*
   Output:
   ۰
   ۱
   ۲
   ۳
   ۴
```

*/

.....

نکته :

Break: موجب خروج از حلقه های تکرار می شود (اگر حلقه تو در تو باشد موجب خروج از حلقه های داخلی می شود)

Continue: موجب انتقال کنترل به انتدای حلقه می شود

کلاس ، شی و متد

برای درک بهتر مطلب موضوع را با یک مثال آغاز می نمایم، یک اتومبیل را در نظر بگیرید معمولا ساخت اتومبیل با ترسیم و نقشه کشی مهندسی شروع می شود، و با ساخت نوع آلیاژهای مختلف تولید می شود. حال شما یک اتومبیل پیش روی خود دارید و فرض کنید می خواهید از نقطه شهر به نقطه دیگر بروید باید چه کار کنید؟ برای به حرکت در آوردن اتومبیل از کلاچ و گاز استفاده می کنید و برای نگه داشتن آن از ترمز، برای آنکه با سرعت بیشتری ادامه دهید پدال گاز را بیشتر فشار می دهید. آیا تمام کسانی که از اتومبیل استفاده می کنند از مکانیزم آن آگاهی دارند ؟ آیا برای استفاده از اتومبیل باید مکانیزم آن را دانست ؟ آیا پدال گاز مکانیزم شتاب گیری را از دید راننده پنهان می کند؟

حال می خواهیم متد ها و کلاس و شی را با مثال فوق برسی نمایم:

متد توصیف کننده مکانیزمی است که وظیفه واقعی خود را انجام می دهد. انجام یک وظیفه در برنامه مستلزم وجود متد است. متد پیچیدگی وظایفی را که قرار است انجام دهد از دید کاربر پنهان می سازد مثل پدال گاز که پیچیدگی مکانیزم شتاب گیری را از دید راننده پنهان می کند.

کلاس همانند نقشه ترسیمی اتومبیل است که طرح پدال گاز هم در آن قرار دارد، یک کلاس می توانید یک یا چند متد داشته باشد که برای انجام وظایف کلاس طراحی شده اند. برای مثال یک کلاس می تواند نشان دهنده یک حساب بانکی و حاوی یک متد برای میزان سپرده در حساب، متد دیگری برای میزان برداشت پول از حساب و متد سومی برای نمایش میزان پول موجود در حساب باشد.

همانطوری که نمی توانید با نقشه ترسیمی اتومبیل رانندگی کنید، نمی توانید کلاسی را مشتق کنید. همانطوری که باید قبل از اینکه بتوانید با اتومبیلی رانندگی کنید، فردی از روی نقشه ترسیمی مبادرت به ساخت اتومبیل کرده باشد، شما هم باید قبل از اینکه برنامه بتواند وظایف توصیفی در کلاس را انجام دهد باید یک شی از کلاس ایجاد کرده باشید. این یکی از دلایل شناخته شدن C# به عنوان یک زبان برنامه نویسی شی گرا است.

تا بدین جا برای مقایسه اتومبیل برای توضیح کلاس ها ، شی ها ، و متد ها استفاده کردیم. علاوه بر قابلیت های اتومبیل، هر اتومبیلی دارای چندین صفت است، صفاتی مانند رنگ، ظرفیت سوخت، سرعت. همانند قابلیت های اتومبیل، این صفات هم بعنوان بخشی از طراحی اتومبیل در نقشه ترسیمی دیده می شوند. همانطوری که رانندگی می کنید ، این صفات همیشه در ارتباط با اتومبیل هستند. هر اتومبیلی حافظ صفات خود می باشد. برای مثال هر اتومبیلی از میزان سوخت باک خود مطلع است اما از میزان سوخت موجود در باک سایر اتومبیل ها مطلع نیست.

یک شی دارای صفاتی است که به همراه شی بوده و در برنامه بکار گرفته می شود. این صفات به عنوان بخشی از کلاس شی تصریح می شوند. برای مثال یک شی حساب بانکی دارای صفت موجودی است که نشان دهنده مقدار پول موجود در حساب می باشد. هر شی حساب بانکی از میزان موجودی خود مطلع

است اما از موجودی سایر حساب ها در بانک اطلاعی ندارد. صفات توسط متغیر های نمونه نمونه کلاس مشخص می شوند. می توانید از خصوصیات یک شی استفاده کنید.

۶-۱ مثال متد ، کلاس و شی

.....

```
class Test
{

   static void Main(string[] args)
   {

      // Keep console window open in debug mode.
      Console.WriteLine("Press any key to exit.");
      Console.ReadKey();
   }
}
/* خروجی:
   Press any key to exit.
 */
```

حال می خواهیم به برسی این برنامه بپردازیم

کلاسی تعریف کردیم با نام test در تابع main شی Console را صدا زده و متد WriteLine را فرآخوانی کرده ایم

۶-۲ اعضای کلاس

اعضای کلاس در سی شارپ

- ثوابت (constants)
- فیلدها (fields)
- متدها (methods)
- خواص (propersties)
- شاخص بند ها (indexers)
- رویدادها (event)
- عملگرها (operators)
- سازنده ها یا مولد ها (constructors)
- مخرب ها (destructor)

اعضای کلاس می توانند به صورت عمومی و خصوصی باشند.

- عمومی (public) : محدودیتی در دستبابی به آن وجود ندارد.
- خصوصی (private) : محدودیت در دستیابی به آن وجو دارد.