دانشگاه جامع علمی کاربردی
مرکز بوکان

جزوه آموزشی

برنامه سازی شیئ گرا

Visual C#

کارشناسی فناوری اطلاعات

گردآوري و تدوين: رحمن رحمن نژاد

سال تحصیلی ۱۳۹۷-۱۳۹۶

دوره مهندسی فناوری اطلاعات - فناوری اطلاعات

	نظری	عملی
واحد	۱	۱
ساعت	۱۶	۳۲

نام درس: برنامه سازی شیء گرا

پیش نیاز: -

الف) سر فصل آموزشی و رئوس مطالب:

ردیف	سرفصل و ریز محتوا	زمان یادگیری (ساعت)	
		نظری	عملی
۱	مدلها، Inheritance , Abstraction , Encapsulation, Classes Objects Pakages , Interfaccs, Gencralization , Polymorphism	۲	۴
۲	آشنایی با کلیات یک زبان شی گرا (مانند تشرش یا C#): دستورات و عبارات، انواع داده‌ها، تعریف متغیرها، انواع متغیرها، توضیحات انواع عملگرهای زبان، ارایه‌ها، دستورات شرطی، حلقه‌ها و ..... - کار با Objects : ایجاد اشیاء، استفاده از New، مدیریت حافظه، مکانیسم‌های مختلف دسترسی به Class، فراخوانی متدها، ارجاع به اشیا و	۴	۸
۳	تعریف Classes: ایجاد متغیرهای Instance، ثابت‌ها و متغیرهای Class، ایجاد متدها ، کلمه کلیدی this، محافظت در برابر دسترسی به اجزاء Class، سازنده‌ها، مخرب‌ها و سربار گذاری، متدهای Static، ایجاد کنترل‌ها و تعاملات	۶	۱۰
۴	مباحث پیشرفته تر: استثناء و پردازش استثناء (Exeception)، بسته‌ها و واسطها، استریم‌ها (Stream) و چند ریسمانی (Multithread)	۴	۱۰

ب) منبع درسی:

۱. آموزش برنامه نویسی کاربردی سی شارپ، محمدرضا مهدیان، به آوران

۲. الگوهای طراحی برنامه نویسی شیئ گرا در C# ، وحید نصیری، ناقوس

۳. *An introduction to Object- Oriented programming, Timothy Budd, Addison wasley,* ۲۰۰۱

۴. *Introduction to programming Using Java: An Object – Oreinted Approach, David M. Amow and Gerald Weiss, Addison Wesley,* ۱۹۹۸

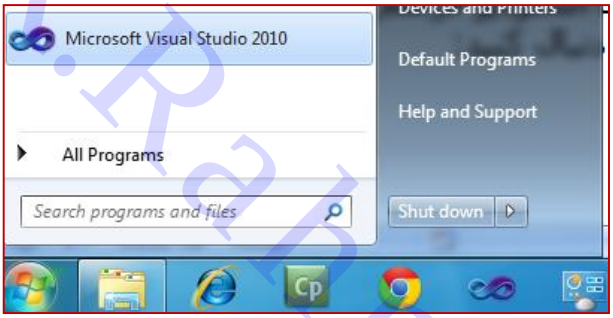
۵. *Java ۲: The Complete Refrence, Third Edition, patraick Naughton and Herbert schildt, Osborne publishing,* ۱۹۹۹.

۶. *Object- Oriented programming with jave :An Introduction, Davide Barnes, Prentice Hall,* ۲۰۰۰.

۷. *Thinking in Java, Bruce Eckel, prentice Hall PTR,* ۲۰۰۳.

فهرست مطالب

فصل اول – آشنایی با Visual Studio



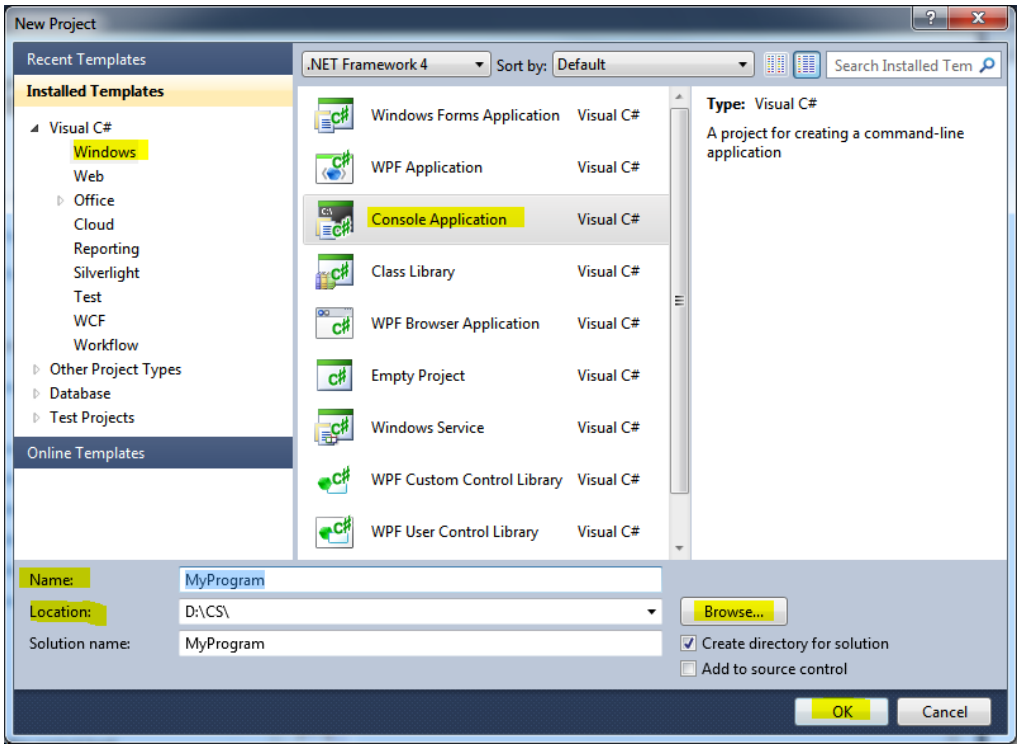
محیط برنامه نویسی برای C# توسط مجموعه ویژوال استودیو فراهم می شود.

بعد از نصب برنامه برای برنامه نویسی مراحل زیر را دنبال کنید:

۱) برنامه Visual Studio را اجرا نمایید.

۲) با صفحه شروع به صورت زیر روبرو می شود:

۳) روی گزینه New Project کلیک نمایید



۴) تنظیمات زیر را در هر قسمت انجام دهید:

- بالا سمت چپ گزینه Windows
- بخش میانی گزینه Console Application
- Name : نامی بصورت لاتین برای نام برنامه وارد کنید
- Location: محل ذخیره پروژه را مشخص کنید. با زدن دکمه Browse مسیر درایو و پوشه مورد نظر را انتخاب نمایید.
- در نهایت دکمه Ok را بزنید تا پنجره کدنویسی ظاهر شود

۵) دستورات برنامه را داخل متد Main بین علائم {} بنویسید

هر برنامه با یک نام کلاس شروع شده و در داخل نام کلاس برنامه اصلی نوشته می شود.

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

namespace MyProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            // دستورات را اینجا بنویسید
        }
    }
}
```

دستور نمایش در خروجی

دستور نمایش خروجی روی صفحه بصورت زیر می باشد:

System.Console.Write("متن خروجی");

System.Console.WriteLine("متن خروجی");

دستور دومی (WriteLine) بعد از نوشتن خروجی مکان نما را به خط بعد می برد و خروجی پس از آن در خط جدید نوشته می شود.

مثال: برنامه نمایش «پیام خوش آمد به C#» روی صفحه بصورت زیر است:

```
class ClassName{
    static void Main(){
        System.Console.Write("Welcome to C# !");
    }
}
```

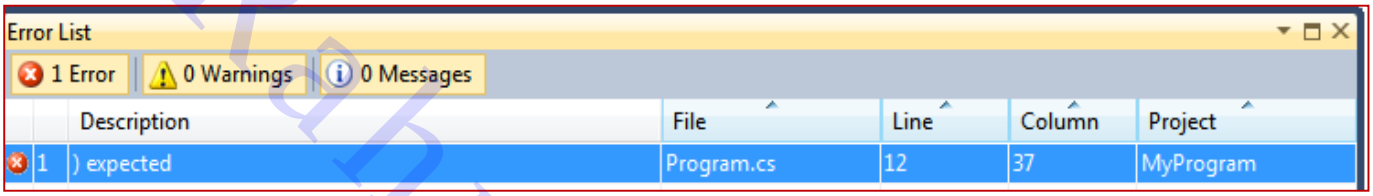
☜نکته ۱: متن خروجی باید داخل علامت " " نوشته شود.

☜نکته ۲: زبان C# نسبت حروف کوچک و بزرگ حساس است، پس باید به همان صورت تعریف شده کلمات را نوشت. یعنی System با system فرق دارد.

جهت بررسی برنامه از نظر درستی دستورات F6 را بزنید. اگر برنامه خطایی داشته باشد در پنجره Error List خطاها را نشان می دهد. مثلاً در دستور زیر

```
Console.WriteLine("Salam";
```

علامت) را فراموش کرده ایم و با پیام خطای زیر روبرو می شویم:



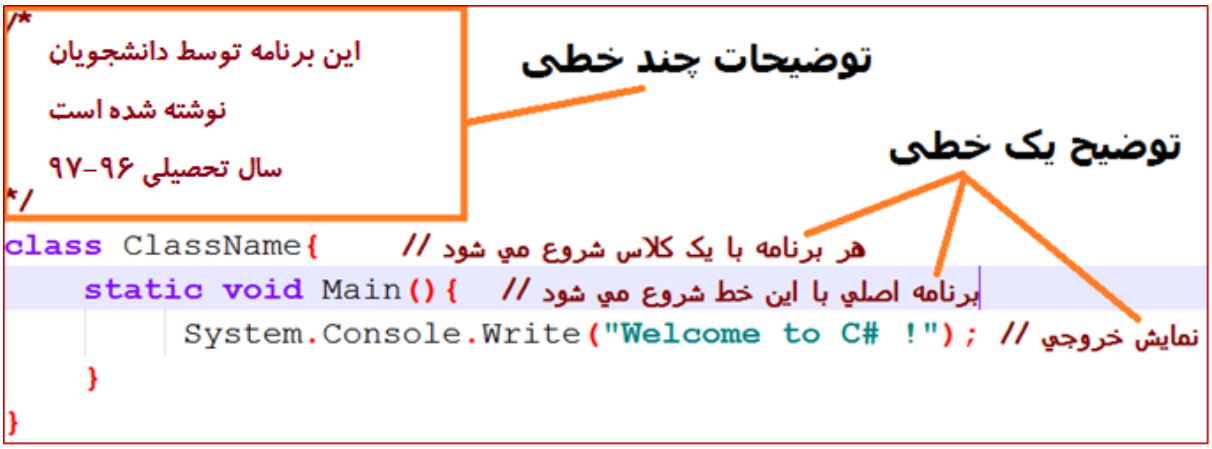
۶) بعد از اطمینان از درستی برنامه، برای اجرا Ctrl+F5 یا F5 را بزنید.

تمرین : دستورات زیر را نوشته و با نام car ذخیره کرده و آنرا ترجمه و اجرا کنید

```
class Car{     // Car: نام کلاس
    static void Main(){
        System.Console.Write(" peykan ");
        System.Console.WriteLine(" bar ");// بعد از نوشتن به خط بعد برو
        System.Console.WriteLine(); // يک خط خالي
        System.Console.WriteLine("___/''''][___]");
        System.Console.WriteLine("'-@-------@--'");
    }
}
```

توضیحات برنامه(Comment): برای توضیح برنامه می توان شرح دلخواهی نوشت که در اجرا تأثیری ندارد و فقط برای خوانایی برنامه و کمک به برنامه نویس و شرح دستورات استفاده می گردد. به دو صورت توضیحات نوشته می شوند:

- یک خطی: بعد از علامت //
- چند خطی: بین علامتهای /* (برای شروع) و */ (برای پایان)



فضای نامی(Name Space): بصورت زیر تعریف می شود:

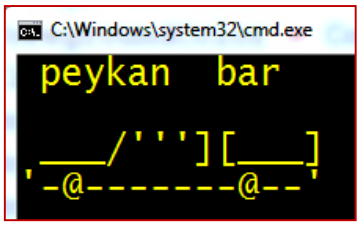
```
using فضای نامی ;
```

مثلاً با نوشتن دستور **using System;** در ابتدای برنامه می توان فضای نامی System را معرفی کرد و در دستورات نمایش خروجی، کلمه System را از دستور زیر حذف کرد:

```
System.Console.Write();
```

به این مثال دقت کنید:

```
using System;
class Car{
    static void Main(){
        Console.Write(" peykan ");
        Console.WriteLine(" bar ");
        Console.WriteLine();
        Console.WriteLine("___/''''][___]");
        Console.WriteLine("'-@-------@--'");
    }
}
```



پاک کردن صفحه: برای

پاک کردن تمام نوشته های قبلی روی صفحه نمایش از دستور زیر استفاده کنید:

```
Console.Clear( );
```

کار با رنگها

الف) رنگ پس زمینه صفحه خروجی

```
Console.BackgroundColor = ConsoleColor.نام رنگ;
```

الف) رنگ پیش زمینه(رنگ نوشته ها) صفحه خروجی

```
Console.ForegroundColor= ConsoleColor.نام رنگ;
```

```
using System;
class Car{
    static void Main(){
        Console.BackgroundColor = ConsoleColor.Black; // رنگ پس زمینه سیاه
        Console.ForegroundColor = ConsoleColor.Yellow ; // رنگ نوشته ها زرد
        Console.Clear();// پاک کردن صفحه
        Console.Write(" peykan ");
        Console.WriteLine(" bar ");
        Console.WriteLine();
        Console.WriteLine("___/''''][___]");
        Console.WriteLine("'-@-------@--'");
    }
}
```

☜**نکته:** بعد از تنظیم رنگها برای اعمال رنگ روی کل پس زمینه صفحه، باید صفحه را پاک کرد (با دستور **Console.Clear();**)

نکته: جدول نام رنگها در C# بصورت زیر می باشد:

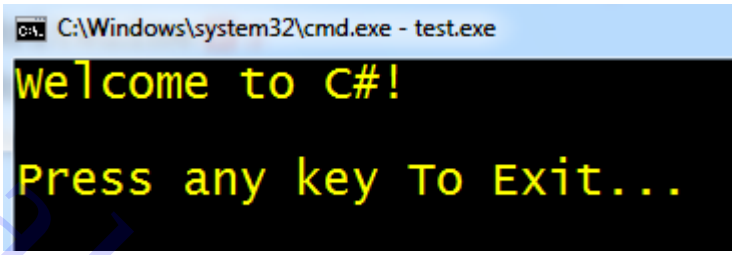
نام رنگ در ConsoleColor	نام رنگ	نمونه رنگ
Black	مشکی	Black
DarkBlue	سورمه ای	DarkBlue
DarkGreen	سبز تیره	DarkGreen
DarkCyan	فیروزه ای تیره	DarkCyan
DarkRed	قرمزتیره	DarkRed
DarkMagenta	بنفش	DarkMagenta
DarkYellow	زرد تیره	DarkYellow
DarkGray	خاکستری تیره	DarkGray
Blue	آبی	Blue
Green	سبز	Green
Cyan	فیروزه‌ای	Cyan
Red	قرمز	Red
Magenta	صورتی	Magenta
Yellow	زرد	Yellow
White	سفید	White
Gray	خاکستری	Gray

نکته: اگر برنامه را با کلید F5 اجرا کنید، بلافاصله بعد از اجرا، پنجره آن بسته می شود و فرصت دیدن نتیجه را نخواهیم داشت. برای اینکه برنامه را وادار کنیم بعد از فشار دادن کلیدی بسته شود از دستور زیر در آخر برنامه کمک می گیریم:

Console.ReadKey( );

```
using System;
class Test{
    static void Main(){
        Console.Clear();// پاک کردن صفحه
        Console.WriteLine("Welcome to C#!");
        Console.WriteLine();
        Console.WriteLine("Press any key To Exit..."); // پیام به کاربر
        Console.ReadKey(); // انتظار براي دريافت كليدي قبل از بسته شدن پنجره اجرا
    }
}
```

خروجی بعد از اجرا بصورت زیر است و تا کلیدی فشار ندهیم منتظر می ماند:



پخش صدا

برای ایجاد یک صدا یا صوت در برنامه از متد زیر از کلاس Console استفاده می شود:

*Console.**Beep**(مدت زمان بر حسب میلی ثانیه ,فرکانس بر حسب هرتز);*

نکته: اگر داخل پرانتز خالی باشد، صوتی با مدت یک ثانیه پخش می کند.

*Console.**Beep**( ); → پخش صوتی با مدت یک ثانیه*

*Console.**Beep**(700 , 2000); → پخش صوتی با مدت دو ثانیه و فرکانس ۷۰۰ هرتز*

دقت کنید که ۲۰۰۰ میلی ثانیه معادل ۲ ثانیه است و محدوده فرکانس قابل شنیدن برای انسان ۲۰۰ تا ۱۰۰۰۰ هرتز می باشد.

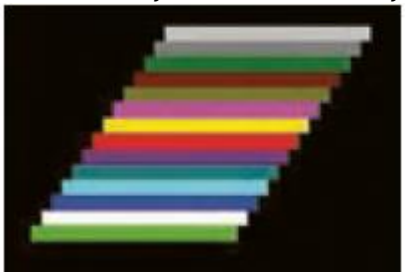
نکته: با استفاده از دستور زیر و با دادن شماره ستون (فاصله از چپ با شروع از صفر)، و سطر (فاصله از بالا با شروع از صفر)، می توان مکان نما را به محل دلخواه در پنجره کنسول منتقل کرد:

SetCursorPosition(left, top);

Console.SetCursorPosition(29 , 9); → انتقال مکان نما به محل سطر ۹ و ستون ۲۹

مقدار ستون(left) و سطر (top) معمولاً در بازه زیر هستند.

$$0 \le left < Console.BufferWidth = 80, \quad 0 \le top < Console.BufferHeight = 300$$



تمرین: برنامه ای بنویسید که پله های رنگی مطابق شکل زیر را ترسیم کند.

بخشی از دستورات را در قطعه زیر می بینید

```
Console.BackgroundColor = ConsoleColor.Red; // رنگ زمینه
Console.SetCursorPosition(20, 2); // رفتن به ستون و سطر مورد نظر
Console.Write("          "); // نمایش کادر رنگی با فاصله
```

برای ترسیم شکلهای دلخواه از کلید Alt استفاده کنید به این ترتیب که پس از انتخاب کد مناسب با استفاده از کلید Alt کد مورد نظر را به کمک قسمت ماشین حساب صفحه کلید وارد نموده و سپس کلید Alt را رها کنید. چند نمونه را در کد زیر می بینید:

```
Console.WriteLine("Alt + Codes: ");
Console.WriteLine("1-6:         ☺ ☻ ♥ ♦ ♣ ♠");
Console.WriteLine("11,12,14-20:  ♂ ♀ ♫ ☼ ► ◄ ↕ ‼ ¶ ");
Console.WriteLine("21-31:       § ▬ ↨ ↑ ↓ → ← ∟ ↔ ▲ ▼");
Console.WriteLine("169-175:     ⌐ ¬ ½ ¼ ¡ « »");

Console.WriteLine("176-190:     ░ ▒ ▓ │ ┤ ╡ ╢ ╖ ╕ ╣ ║ ╗ ╝ ╜ ┐");

Console.WriteLine("191-200:     └ ┴ ┬ ├ ─ ┼ ╞ ╟ ╚");

Console.WriteLine("201-210:     ╔ ╩ ╦ ╠ ═ ╬ ╧ ╨ ╤ ╥");

Console.WriteLine("211-223:     ╙ ╘ ╒ ╓ ╫ ╪ ┘ ┌ █ ▄ ▌ ▐ ▀");
Console.WriteLine();
```

فصل دوم-آشنایی با انواع داده ها و متغیرها

تصور کنید که سر سفره غذا نشسته ایم؛ برای هر نوع غذا و خوراکی ظرف مخصوص خود در نظر گرفته شده است. در برنامه نویسی هم برای هر نوع اطلاعاتی که می خواهیم برنامه پردازش کند، باید ظرف مخصوص آنرا برای ذخیره استفاده نماییم. در برنامه نویسی به این ظرفها، متغیر می گویند.

متغیر(Variable): مکانی نامدار از حافظه اصلی(RAM) برای نگهداری موقت اطلاعات مورد نیاز پردازش در کامپیوتر می باشد. مقدار متغیر در طول برنامه ممکن است تغییر کند.

برای استفاده از متغیرها در برنامه نخست باید آنها را معرفی کنیم. برای معرفی متغیرها دو مشخصه را باید تعیین نماییم:

- نوع متغیر: تعیین می کنید که چه نوع اطلاعاتی در آن ذخیره می کنیم(عددی، رشته کارکتر و ...).
- نام متغیر: نامی که برای ظرف خود انتخاب می کنیم که البته باید از قوانین خاص نامگذاری پیروی کند.

درنامگذاری متغیرها،رعایت موارد زیر الزامی است:

۱) استفاده ازحروف الفبا،اعداد و کاراکتر زیرخط، مجاز است.
۲) نام متغیر نمیتواند با عدد شروع شود.
۳) نام انتخابی نمیتواند با کلمات کلیدی یا رزرو شده باشد.
۴) استفاده از علامت فاصله و خط تیره و ... در نام متغیر مجاز نیست.

در جدول روبرو چند نمونه نامگذاری صحیح و نادرست را مشاهده می کنید:

نام متغیر	درست یا غلط	دلیل نادرست بودن
ABC	درست	
Ali23	درست	
P2p	درست	
15M	غلط	با عدد شروع شده است
My_Age	درست	
_New	درست	
Ab_	درست	
My var	غلط	فاصله قابل قبول نیست
s@lam	غلط	علامت @ قابل قبول
class	غلط	این کلمه رزرو شده است

درانتخاب نام متغیرها، بهتر است نکات زیر رعایت شود:

- نام بامعنی و با توجه به کاربرد متغیر در برنامه انتخاب شود. مانند Score برای نمره

- از نامهای مخفف استفاده نکنید چون خواندن آنها مشکل است
- اولین حرف نام متغیر را با حروف کوچک شروع کنید و اگر نام متغیر از چند کلمه تشکیل شده، برای خوانایی، حرف اول کلمات بعدی را با حروف بزرگ بنویسید. به این روشِ نوشتن نام، کوهان شتری(Camel Case)میگویند. مانند secondsPerHour برای ذخیره تعداد ثانیه در هر ساعت.

قالب کلی تعریف متغیر بصورت زیر است:

نام متغیر نوع متغیر;

نوع متغیر در واقع سه ویژگی را مشخص می کند:

۱) *گنجایش یا ظرفیت متغیر*
۲) *نوع اطلاعاتی که می توان در متغیر ذخیره نمود*
۳) *چه عملیاتی روی آن قابل انجام است*

انواع داده ها در C# طبق جدول زیر است:

	نوع داده	کاربرد نوع داده	مقدار حافظه (بایت)	کمترین مقدار	بیشترین مقدار
عددی	sbyte	اعداد صحیح	1	−128	127
	byte	اعداد صحیح مثبت	1	0	255
	short	اعداد صحیح	2	−32768	32767
	ushort	اعداد صحیح مثبت	2	0	65535
	int	اعداد صحیح	4	−2147483648	2147483647
	uint	اعداد صحیح مثبت	4	0	4294967295
	long	اعداد صحیح	8	−9223372036854778508	9223372036854778507
	ulong	اعداد صحیح مثبت	8	0	18446744073709551615
	float	اعداد اعشاری	4	-3.402823×10^{38}	3.402823×10^{38}
	double	اعداد اعشاری با دقت زیاد	8	$-1.79769313486232 \times 10^{308}$	$1.79769313486232 \times 10^{308}$
	decimal	اعداد صحیح بزرگ اعداد اعشاری با دقت بسیار زیاد	16	$-79228162514264337593543950335$ -7.9×10^{28}	$79228162514264337593543950335$ $+7.9 \times 10^{28}$
غیرعددی	bool	مقدار منطقی	1	false	true
	char	یک حرف یا علامت (کراکتر)	2	0 کد کراکتر مطابق با سیستم Unicode	65535 کد کراکتر مطابق با سیستم Unicode
	string	رشته		–	–
	object	آدرس یک داده		–	–

مثال: متغیری برای ذخیره نمره درس برنامه سازی تعریف کنید

float nomre;

در اینجا، چون نمره عددی اعشاری است(مثل ۱۷,۵)، پس نوع داده float انتخاب شده است و نام آن به دلخواه nomre تعیین شده است.

اعداد صحیح

برای اعداد صحیح و بدون ممیز نوع داده های زیر استفاده می شود:

sbyte (– یا +) ,byte (+) ,

short(– یا +), ushort (+) ,

int (– یا +) , uint (+) ,

long (– یا +) , ulong (+)

و برای اعداد اعشاری میتوانید از نوع داده های زیر استفاده کنید:

double(اعشاری با دقت زیاد) , float(تا دقت ۷ رقم)

جدول زیر مثالهایی از انواع متغیرها را بر اساس کاربرد آنها نشان می دهد.

کاربرد	محدوده مقادیر	تعریف متغیر
موجودی حساب	عدد دلخواه	**long** Mowjodi ;
تعداد روزهای یک ماه	۱ تا ۳۱	**byte** Days;
نمره این درس	۰ تا ۲۰ اعشاری	**float** Nomre ;
نام خود	رشته کاراکتر	**string** MyName;
پلاک ماشین	رشته ای با طول ۱۱	**string** Pelak;
شماره ملی	عدد بزرگ	**long** melli ;
تلفن منزل بدون پیش شماره	۴۶۲******	**int** Tel ;
موبایل	رشته ای با طول ۱۱	**string** Mobile;
وضعیت یک لامپ	روشن(True) یا خاموش(False)	**bool** Lamp;

نکته: برخی داده دارای ماهیت عددی هستند، اما پردازش عددی و محاسبات روی آنها انجام نمی شود و فقط جنبه نمایشی دارند. برای این نوع داده ها می توان از string به جای نوع داده صحیح(int , ...) استفاده نمود. به مثال زیر دقت کنید:

string mobile; → برای ذخیره صفر اول بهتر است به این شکل تعریف شود

نحوه مقدار دهی به متغیر: ابتدا متغیر را از نوع مناسب تعریف کنید سپس توسط دستور زیر مقدار دهی کنید

تعریف→ ; نام متغیر نوع متغیر

مقدار دهی→; مقدار= نام متغیر

sbyte dama1, dama2;→ *دو متغیر برای برای ذخیره دما (بصورت مثبت و منفی)*

```
dama1 = -2;
dama2 =  5;
```

نمایش مقدار متغیر:

```
Console.Write(نام متغیر);
Console.Write(dama1);
```

نمایش پیام همراه با مقدار متغیر:

```
Console.Write("پیام" + نام متغیر);
Console.Write(" دمای بوکان = " + dama1);
```

تعریف متغیر همزمان با مقدار دهی:

```
نوع متغیر  نام متغیر = مقدار;
```

byte *sen = 16;→ تعریف متغیر برای ذخیره سن خود و مقداردهی همزمان*

مثال: به برنامه زیر دقت کنید:

Program.cs

MyProgram.Program Main(string[] args)

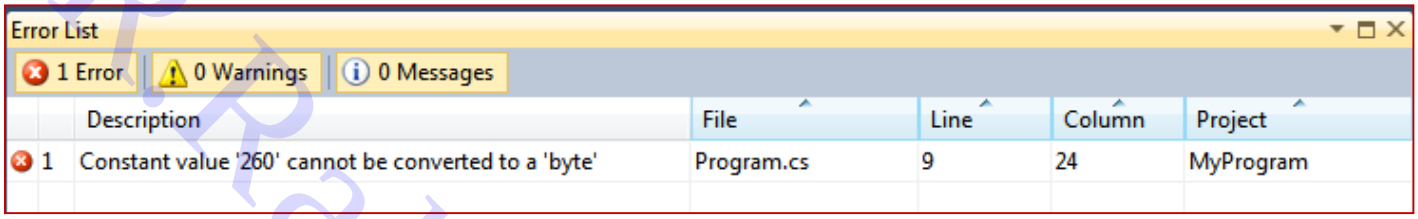
```
using System;
namespace MyProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            sbyte dama1, dama2;// تعریف دو متغیر برای دما
            byte sen = 16; // متغیر برای سن خود مقدار دهی همزمان
            dama1 = -2; // مقدار دهی دمای شب
            dama2 = 5;  // مقدار دهی دمای روز
            Console.WriteLine("shab = " + dama1); // نمایش دمای شب
            Console.WriteLine("ruz = " + dama2);  // نمایش دمای روز
            Console.WriteLine("sene man = " + sen); // نمایش سن خود
        }
    }
}
```

C:\Windows\system32\cmd.exe

```
shab = -2
ruz = 5
sene man = 16
Press any key to continue . . .
```

☜ نکته: اگر مقداری در متغیر ذخیره کنید که خارج از ظرفیت متغیر باشد، با پیام خطا مواجه می شوید:

byte money = 260; → خطا: فقط مقادیر ۰ تا ۲۵۵ را می توان در این نوع داده ذخیره کرد



نکته: برای نمایش اعداد در مبنای ۱۶ باید قبل از عدد مورد نظر 0x یا 0X قرار دهید.

byte test = 0X1B; → 1B : ذخیره مقدار مبنای ۱۶

با دستور فوق مقدار عددی 1B در مبنای ۱۶ که معادل عدد ۲۷ می باشد در متغیر test ذخیره می شود.

$$(1B) = (1\times16)+11=27$$

مثال: ذخیره عدد پی(π) :

```
double  pi= 3.1415916589;
```

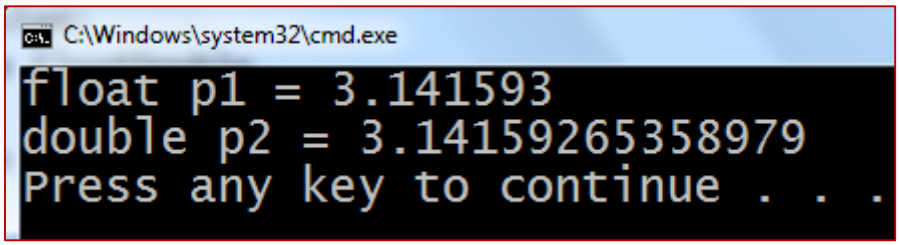
ذخیره نمره درس:

```
float  nomre = 17.5f;
```

حرف f در انتهای نمره یعنی آنرا از نوع float در نظر بگیرد.

مثال: تفاوت دو نوع داده float و double در ذخیره مقدار:

```
using System;
namespace FloatDouble
{
    class Program
    {
        static void Main(string[] args)
        {
            float p1 = 3.14159265358793238f; //float عدد اعشاری
            double p2 = 3.1415926535897932 38;//double عدد اعشاری
            Console.WriteLine("float p1 = " + p1);
            Console.WriteLine("double p2 = " + p2);
        }
    }
}
```



می بینیم که اعداد float با دقت شش رقم اعشاری و اعداد double با دقت ۱۴ رقم اعشاری ذخیره می شوند.

کار با اعداد اعشاری

به دو شیوه عدد اعشاری را نمایش می دهند:

- شیوه نماد علمی: عدد از دو بخش تشکیل شده است که با علامت ضرب ×جدا می شوند. بخش سمت چپ رقمی بین ۱ تا ۹ است(مانتیس) و بخش دوم بصورت توانی از ۱۰(نما)

نما	علامت	مانتیس
توانی از ۱۰	×	رقمی بین ۱ تا ۹

نحوه تبدیل: یک رقم سمت چپ نقطه ممیز قرار گیرد و بقیه بعد از علامت ممیز (.) و سپس علامت × و توان ۱۰ به تعداد رقمهای دیگر قسمت صحیح که به قسمت اعشاری منتقل شده اند:

شش رقم صحیح به پشت ممیز منتقل شده اند → 4.38×10^{6} ← **4380000**

چهار رقم به پشت ممیز منتقل شده اند → 4.875825×10^{4} ← **4875**8.25

رقم ۲ به تعداد سه مکان به چپ منتقل شده است → 2.5×10^{-3} ← **0.0025**

- ممیز شناور: در زبان C# برای نمایش نماد علمی به جای 10× حرف E نوشته می شود.

4380000 → 4.38×10^{6} → 4.38**E6**

48758.25 → 4.875825×10^{4} → 4.875825**E4**

0.0025 → 2.5×10^{-3} → 2.5**E-3**

```
static void Main(string[] args)
{
    double d = 4.875825E4; //نمایش عدد اعشاری با نماد علمی
    Console.WriteLine(d);
}
```

خروجی:

```
C:\Windows\system32
48758.25
Press any key to continue .
```

نوع داده منطقی(bool)

در صورتی که بخواهیم دو وضعیت مختلف را نشان دهیم، این نوع داده می تواند مفید باشد و فقط برای ذخیره مقدار درست(true) یا نادرست(false) استفاده می شود. مثلاً برای روشن بودن لامپ(مقدار true) و برای خاموش بودن(مقدار false) را در نظر می گیریم:

bool lamp = false; →false :*وضعیت لامپ خاموش*

bool q1 = true ; →*مقدار درست برای جواب یک سوال*

نوع داده کارکتری(char)

برای ذخیره یک کارکتر، حرف، رقم، علامت و ... (روی کلیدهای صفحه کلید) و همچنین کارکترهای خاصی که با نگه داشتن کلید Alt و تایپ کد آن در قسمت عددی صفحه کلید تولید می شوند، بکار می رود. مقدار کارکتری باید داخل علامت ‘ ’ ذخیره شود.

char harf = **‘A’**;→A *ذخیره حرف*

char fasele = ‘ ’; →*ذخیره علامت فاصله*

می توان به جای کارکتر، کد آنرا در مبنای ۱۶ ذخیره کرد:مثلاً کد کارکتر حرف A عدد ۶۵ است که در مبنای ۱۶ عدد ۴۱ می باشد:

$A : (65)_{10} \rightarrow (41)_{16}$

char harf = **‘\u0041’**; →A *کد کارکتر حرف*

داخل علامت‘ ’، علامت \u را نوشته و به دنبال آن کد کارکتر را در مبنای ۱۶ بصورت ۴ رقمی می نویسیم.

نوع داده رشته ای(string)

برای ذخیره تنها یک کارکتر از char استفاده می کنیم. اما برای ذخیره چندین کارکتر مثل نام خود، از string کمک می گیریم. مقدار رشته داخل علامت " " قرار می گیرد.

string name; →متغیر رشته ای

name = “Hiwa” ;→ذخیره مقدار

اعمال رشته

- الحاق یا چسباندن رشته ها به هم توسط علامت + :

```
string name = “Hiwa”;
string family = ”Kamyar”;
Console.Write(name+ “ ” +family); →Hiwa Kamyar    خروجی با یک فاصله بین آنها
```

دریافت رشته از ورودی

برای اینکه کاربر بتواند ورودی را از صفحه کلید دریافت کند(در حین اجرای برنامه)، از دستور زیر استفاده می شود:

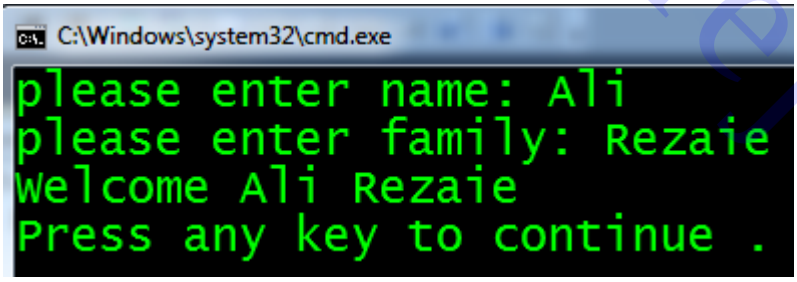
متغیر = Console.**ReadLine**( );→دریافت رشته و ذخیره در متغیر

نکته: برای خوانایی خروجی برنامه و راهنمایی کاربر برای دریافت داده ها، بهتر است قبل از هر دستور ورودی، به کمک دستور Write پیامی را نمایش دهیم.

مثال: دو متغیر برای ذخیره نام و نام خانوادگی تعریف کرده و سپس مقدار آنها را از ورودی دریافت کنید و همراه پیام خوش آمد، مقدار آنها روی صفحه نشان دهید

```
using System;
namespace NameFamily
{
    class Program
    {
        static void Main(string[] args)
        {
            string name;  // متغیر نام
            string family; // متغیر نام خانوادگی
            Console.Write("please enter name: "); // پیام دریافت نام
            name = Console.ReadLine(); // دریافت نام
            Console.Write("please enter family: ");// پیام دریافت فامیلی
            family = Console.ReadLine(); // دریافت نام خانوادگی
            // نمایش همراه با پیام خوش آمد
            Console.WriteLine("Welcome " + name + " " + family);
        }
    }
}
```

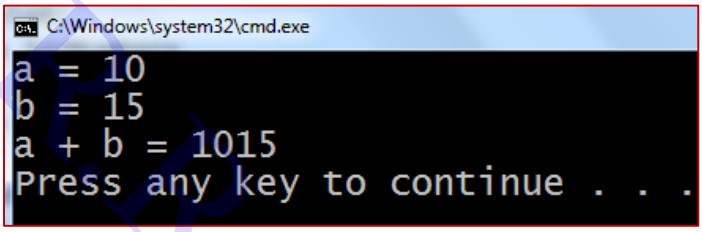
خروجی:



مثال: برنامه زیر را در نظر بگیرید:

```
using System;
namespace AddNumbers
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 10, b = 15;
            Console.WriteLine("a = " + a);
            Console.WriteLine("b = " + b);
            Console.WriteLine("a + b = " + a + b);
        }
    }
}
```

خروجی برنامه بصورت زیر خواهد بود:



می بینیم به جای جمع دو مقدار، آنها را کنار هم نمایش داده است!

دلیل: چون دستور زیر آنها را بصورت رشته درنظر گرفته و آنها را الحاق می کند(کنار هم می نویسد):

```
Console.WriteLine("a + b = " + a + b);
```

می توان دستور آخر را به صورت زیر بازنویسی کرد که خروجی مورد انتظار تولید می شود:

```
Console.WriteLine("a + b = " + (a + b));
```

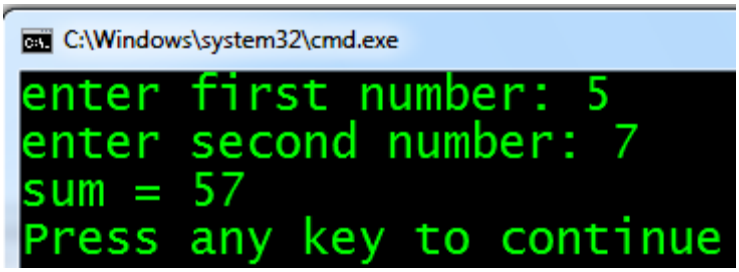
خروجی:

```
a = 10
b = 15
a + b = 25
```

مثال ۲: دو عدد بخوانید و مجموع آنها را نمایش دهید

```
using System;
namespace TwoNumbers
{
    class Program
    {
        static void Main(string[] args)
        {
            string a,b; // دو متغیر
            Console.Write("enter first number: "); // پیام
            a = Console.ReadLine(); // خواندن عدد اول
            Console.Write("enter second number: "); // پیام
            b = Console.ReadLine(); // خواندن عدد دوم
            Console.WriteLine("sum = " + a + b); // نمایش نتیجه
        }
    }
}
```

خروجی برنامه بصورت زیر است:



توضیح: می بینیم که به جای حاصلجمع ۵+۷ که عدد ۱۲ است، خروجی ۵۷ را نشان داده است. چرا؟

دلیل: چون علامت + رشته ها را کنار هم قرار می دهد و جمع نمی کند.

سوال: برای جمع چکار کنیم؟

قدم اول: باید متغیرهای خود را عددی تعریف کنیم:

```
int a,b;
```

قدم دوم: چون Console.ReadLine() ورودی را بصورت رشته دریافت می کند، باید قبل از ذخیره در متغیرهای عددی، به عدد تبدیل شوند. این کار با متد Parse() و دستور زیر انجام می شود:

```
متغیر عدد صحیح = int.Parse(مقدار رشته ای);
```

نکته: تبدیل داده رشته ای(ارقام رشته ای) به عدد اعشاری float یا double بصورت زیر خواهد بود:

```
متغیر عدد float = float.Parse(مقدار رشته ای);
متغیر عدد double = double.Parse(مقدار رشته ای);
```

حال برنامه قبل را باز نویسی می کنیم:

```
class Program
{
    static void Main(string[] args)
    {
        int a, b,c; // متغیرهای عددی
        string input; // متغیر ورودی
        Console.Write("enter first number: "); // پیام
        input = Console.ReadLine(); // خواندن عدد اول
        a = int.Parse(input); // تبدیل ورودی رشته ای به عدد
        Console.Write("enter second number: "); // پیام
        input = Console.ReadLine(); // خواندن عدد دوم
        b = int.Parse(input); // تبدیل ورودی رشته ای به عدد
        c = a + b; // جمع اعداد
        Console.WriteLine("sum = " + c); // نمایش نتیجه
    }
}
```

خروجی:

```
C:\Windows\system32\cmd.exe
enter first number: 5
enter second number: 7
sum = 12
Press any key to continue
```

تمرین: نام و نمره دو درس خود را بخوانید و سپس نام دروس، نمره هریک و مجموع نمرات آنها را نمایش دهید.

```
C:\Windows\system32\cmd.exe
name darse 1: Barname sazi (1)
nomre darse 1: 15.5
name darse 2: farsi
nomre darse 2: 18
Barname sazi (1) : 15.5
farsi : 18
majmue = 33.5
Press any key to continue . . .
```

فصل سوم – عبارتهای محاسباتی

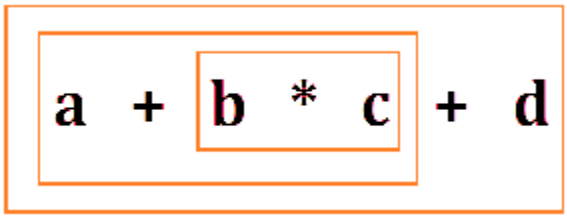
عملگر: به هریک از علائمی که بیانگر عمل خاصی مثل محاسبات (+، -، *، /، %) هستند، عملگر می گویند.
عملوند: به اعداد یا مقادیری که عمل مورد نظر روی آنها انجام می گیرد، عملوند می گوییم.

- برخی عملگرها، دو تایی هستند؛ یعنی دو عملوند نیاز دارند. مانند عملگر جمع (a+b) که دو عملوند a و b دارد.
- برخی عملگرها یکتایی هستند؛ یعنی فقط یک عملوند لازم دارند. مانند عملگر قرینه(a-) که عمل منفی کردن(-) روی یک عملوند (a) انجام می شود.

عبارت: یک عبارت از تعدادی عملگر و عملوند تشکیل شده است و یک نتیجه را بدست می دهد که ممکن است عددی یا غیر عددی باشد. مانند: **a+b*c**

اولویت عملگرها: ترتیب اجرای عملگرها را مشخص می کند. اگر ترتیب اجرای دو عملگر در یک عبارت یکسان باشد، به ترتیب چپ به راست انجام خواهد شد.
مثال: در عبارت **a+b*c+d** ابتدا **c*d** انجام می شود (* اولویت بالاتری دارد). سپس از بین دو جمع، جمع سمت چپ انجام می شود یعنی **a** با حاصل **c*d** جمع می شود و در نهایت حاصل این مرحله با **d** جمع خواهد شد.

4 + 5 * 6 + 3 = 4 + (30) + 3 = (34) + 3 = 37



عملگرهای ریاضی یا حسابی

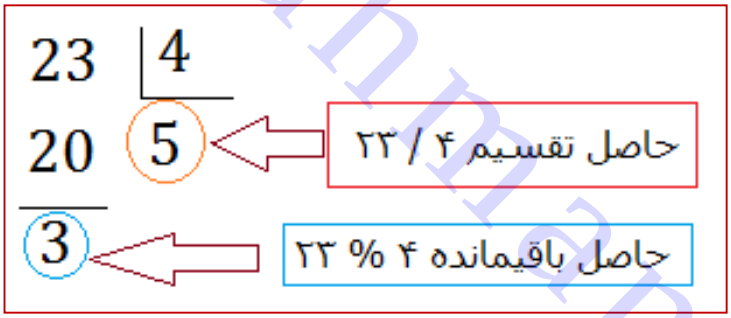
اولویت	نام عملگر	نشانه	مثال	نوع عملگر
۱	قرینه	-	-5	یکتایی
۲	ضرب	*	12 * 36	دو تایی
	تقسیم	/	25/4	
	باقیمانده تقسیم	%	23 % 5	
۳	جمع	+	75 + 14	دو تایی
	تفریق	-	29 – 36	

عملکرد عملگرهای جمع و تفریق و ضرب مانند عملکرد آنها در ریاضیات است اما عملگر تقسیم با توجه به نوع عملوندهایش میتواند تقسیم صحیح و بدون ممیز و یا تقسیم اعشاری و ممیزی انجام دهد. مثلاً تقسیم دو عدد صحیح 9/2 مقدار 4 را نتیجه می دهد ولی تقسیم بین اعداد اعشاری 9/2.0 یا 9.0/2 یا 9.0/2.0 نتیجه 4.5 را بدست می آورد.

در جدول فوق، عملگر جدیدی می بینید که باقیمانده تقسیم را بدست می آورد. مثلاً 9%2 نتیجه عدد یک است چون باقیمانده تقسیم ۹ بر ۲ برابر ۱ است.

به این مثالها دقت کنید:

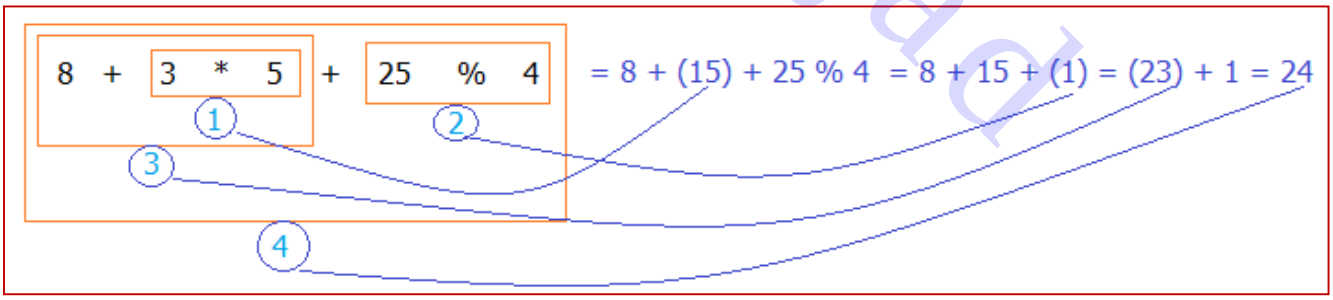
23 / 4 = 5
23 % 4 = 3



نکته: از پرانتز برای تغییر اولویت می توان استفاده کرد. مثلاً در عبارت زیر هرچند اولویت ضرب بیشتر از جمع است، ولی چون جمع داخل پرانتز است، زودتر انجام می شود.

5 * (4+6) = 5* (10) = 50

به مثال زیر در مورد اولویت اجرای عملگرها دقت کنید. ترتیب اولویتها با شماره (۱) تا (۴) مشخص شده و نتیجه هر مرحله نیز داخل پرانتزها نشان داده شده است.



نتایج عبارات محاسباتی معمولاً در یک متغیر نتیجه ذخیره می شوند مثلاً در عبارت زیر نتیجه در متغیر A ذخیره می شود:

A = 8 + 3 * 5 + 25 % 4 → A = 24

قوانین زیر، باید در هنگام انتساب یک عبارت به یک متغیر، رعایت شود و گرنه با پیام خطای مترجم مواجه میشویم.

۱) اگر حاصل یک عبارت عددصحیح باشد بسته به اندازه و بزرگی عدد، میتواند در یک متغیرِ نوعِ صحیح که گنجایش آن مساوی یا بزرگتر از حاصل عبارت باشد جای گیرد. مثلاً حاصل عبارت 175 / 31 که برابر 5 است در متغیر صحیح از نوع های زیر قرار گیرد:

sbyte,byte , short , ushort , int , uint , long , ulong

و حاصل 2541389 / 10 که عدد 254138 می باشد، فقط در متغیرهای صحیح از نوع های زیر قابل ذخیره سازی است

int,uint , long , ulong

۲) اگرحاصل یک عبارت از نوع صحیح باشد میتواند در یک متغیر نوع اعشاری نیز ذخیره شود البته اعداد صحیح بزرگ(long) فقط با دقت ۷ رقم اعشار در متغیر نوع float و با دقت ۱۵ رقم در متغیر نوع double ذخیره شده و رقمهای اضافی گرد می شوند.

```
using System;
namespace IntegerinFloat
{
    class Program
    {
        static void Main(string[] args)
        {
            float number1 = 1234567890 / 10;
            double number2 = 1234567890 / 10;
            Console.WriteLine("1234567890 / 10 --> float : " + number1);
            Console.WriteLine("1234567890 / 10 --> double : " + number2);
        }
    }
}
```

```
C:\Windows\system32\cmd.exe
1234567890 / 10 --> float : 1.234568E+08
1234567890 / 10 --> double : 123456789
```

می بینیم که نتیجه تقسیم بصورت float در قالب نماد علمی ذخیره شده است:

1.234568E+08→ 1.234568×10⁸→ 123456800

یعنی در مقایسه با جواب اصلی تقسیم(۱۲۳۴۵۶۷۸۹) از رقم ۷ به بعد گرد شده است.

۳) اگر حاصل عبارت مقداری اعشاری(float یا double) باشد، نمی تواند در متغیر از نوع صحیح قرار بگیرد.

```
int  a;
a = 7.5 / 2 = 3.75 ;
```

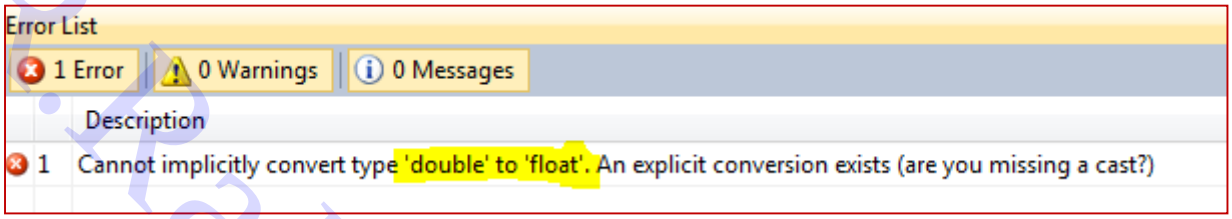
→نتیجه تقسیم(۳,۷۵) را نمی توان در متغیر صحیح ذخیره کرد

۴) اگر حاصل عبارت مقداری اعشاری از نوع double باشد، فقط در متغیر از نوع double قابل ذخیره سازی است.

به برنامه زیر دقت کنید:

```
using System;
namespace doubleFloat
{
    class Program
    {
        static void Main(string[] args)
        {
            double    d;
            float f2;
            d = 219.5 / 14;
            f2 = 219.5 / 14;
            Console.WriteLine("double: " + d);
            Console.WriteLine("float : " + f2);
        }
    }
}
```

با اجرای آن با خطای زیر مواجه می شویم:

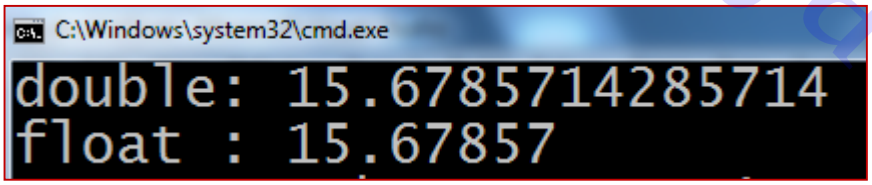


دلیل خطا: در خط چهارم برنامه، بصورت پیش فرض اعداد را بصورت double در نظر می گیرد و نتیجه در متغیر f2 که از نوع float است قابل ذخیره سازی نیست.

راه حل: در انتهای عدد اعشاری 219.5 باید حرف f را قرار داد تا آنرا بصورت float در نظر بگیرد.

```
double d;
float f2;
d = 219.5 / 14;
f2 = 219.5f / 14;
Console.WriteLine("double: " + d);
Console.WriteLine("float : " + f2);
```

حال خروجی بصورت زیر خواهد بود.



مثال: قد(برحسب متر) و وزن (برحسب کیلوگرم) را از ورودی خوانده و تناسب قد به وزن را با محاسبه نسبت زیر بررسی کنید. وزن را بر قد به توان ۲ تقسیم کنید:

نسبت = وزن (کیلوگرم) / (قد بر حسب متر)^2

- *نسبت کمتر از ۲۰: لاغر*
- *نسبت بیشتر از ۲۵ : چاق*
- *نسبت بین ۲۰ تا ۲۵: نرمال*

همچنین با توجه به نسبت فوق، با جایگذاری به ترتیب مقادیر ۲۰ و ۲۵ برای نسبت، وزن حداقل و حداکثر نرمال را نیز محاسبه نمایید.

وزن حداقل (کیلوگرم) = ۲۰ (قد بر حسب متر)^2*

وزن حداکثر (کیلوگرم) = ۲۵ (قد بر حسب متر)^2*

```
static void Main(string[] args)
{
    float wazn, qad, nesbat; // متغیرهای وزن، قد و نسبت
    string input;            //دریافت ورودی
    float min, max;          // وزن حداقل و حداکثر نرمال
    Console.Write("wazn(kilogram): ");
    input = Console.ReadLine();     // ورودی وزن
    wazn = float.Parse(input);      //تبدیل ورودی رشته ای به اعشاری
    Console.Write("Qad (meter): ");
    input = Console.ReadLine();     //ورودی قد
    qad = float.Parse(input);       //تبدیل ورودی رشته ای به اعشاری
    nesbat = wazn / (qad * qad);    // محاسبه نسبت
    min = 20 * (qad * qad);         // وزن حداقل نرمال
    max = 25 * (qad * qad);         // وزن حداکثر نرمال
    Console.WriteLine("nesbat : " + nesbat); // نمایش نسبت
    Console.WriteLine("min : " + min);       // نمایش وزن حداقل
    Console.WriteLine("max : " + max);       // نمایش وزن حداکثر
}
```

خروجی نمونه بصورت زیر است:

یعنی اگر قد یک نفر ۱,۷۲ باشد، محدوده وزن نرمال او بین حدود ۵۹ تا ۷۴ خواهد بود.

```
C:\Windows\system
wazn(kilogram): 65
Qad (meter): 1.72
nesbat : 21.97133
min : 59.168
max : 73.96
Press any key to continue
```

عملگرهای افزایشی و کاهشی

در زبان C# عملگرهای یک عملوندی افزایشی(++) و کاهشی (--) برای افزایش و کاهش یک واحد وجود دارند. به مثال زیر دقت کنید:

```
int  a = 5;
a++;  → a= a + 1; →a = 6:یعنی یکواحد افزایش
a -- ;→ a = a – 1; →a = 4  یعنی یکواحد کاهش:
```

توضیح اینکه عملگر ++ و -- یکواحد به متغیر خود اضافه یا کم می کنند و به دو شکل پسوندی(بعد از متغیر) و پیشوندی(قبل از متغیر)، استفاده می شوند:

متغیر ها	به شکل پسوندی	به شکل پیشوندی	معادل دستور
a = 5 ;	a ++ ; → a = 6	++ a ; → a = 6	a = a + 1 ;
b = 10 ;	b -- ; → b = 9	-- b ; → b = 9	b = b - 1 ;

تفاوت دو حالت پسوندی و پیشوندی ++ و --

در حالت پسوندی اول استفاده می شود(شرکت در انتساب و انتقال به متغیر چپ) و سپس عمل جمع(تفریق) انجام می شود، اما در پیشوندی؛ نخست عمل جمع(تفریق) انجام شده و سپس استفاده می شود. در هر دو حالت تأثیر روی عملوند ++ و -- یکسان است.

متغیر ها	به شکل پسوندی		به شکل پیشوندی	
a = 5 ;	b = a ++ ;	→ b = 5 , a = 6	b = ++ a ;	→ a = 6 , b = 6
	b = a -- ;	→ b = 5 , a = 4	b = -- a ;	→ a = 4 , b = 4

مثال : در برنامه زیر دو حالت پسوندی و پیشوندی نشان داده شده است.

```
using System;
namespace PlusPlus
{
    class Program
    {
        static void Main(string[] args)
        {
            int a ,b;    // تعریف متغیرها
            a = 5;       // مقداردهی با ۵
            Console.WriteLine(" a = " + a);
            b = a ++;    // عملگر افزایشی پسوندی
            Console.Write(" b = a ++    --> ");
            Console.WriteLine("a = " + a + " , b = " + b);
            a = 5;
            b = ++ a;    // عملگر افزایشی پیشوندی
            Console.Write(" b = ++ a    --> ");
            Console.WriteLine("a = " + a + " , b = " + b);
            Console.WriteLine();
        }
    }
}
```

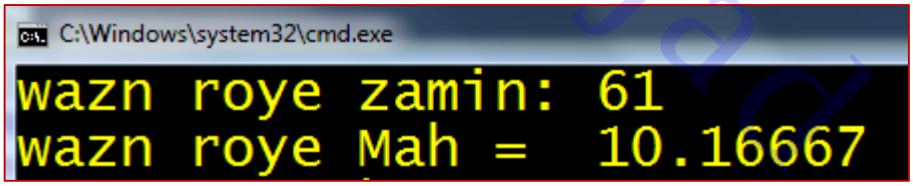
```
C:\Windows\system32\cmd.exe
 a = 5
 b = a ++     --> a = 6 , b = 5
 b = ++ a     --> a = 6 , b = 6
```

در حالت پسوندی می بینیم که اول مقدار a به b منتقل شده و سپس افزایش صورت می گیرد، اما در حالت پیشوندی، نخست افزایش یکواحدی a انجام شده و سپس مقدار آن به b منتقل می گردد.

تمرین عملی : نیروی جاذبه در کره ماه $\frac{1}{6}$ (یک ششم) جاذبه زمین است. برنامه ای بنویسید که وزن یک شخص در روی زمین را سؤال کرده و سپس وزن وی در کره ماه را حساب کند. برای محاسبه وزن روی ماه، وزن خود روی زمین را در $\frac{1}{6}$ ضرب می کنیم:

```
using System;
namespace MoonWeight
{
    class Program
    {
        static void Main(string[] args)
        {
            float waznZamin, waznMah;          // تعریف متغیرها
            string vorodi;                     // متغیر ورودی
            Console.Write("wazn roye zamin: "); // نمایش پیام ورودی
            vorodi = Console.ReadLine();       // خواندن وزن روی زمین
            waznZamin = float.Parse(vorodi); // تبدیل ورودی رشته ای به اعشاری
            waznMah = (waznZamin * 1) / 6;  // تبدیل وزن روی زمین به وزن روی ماه
            Console.WriteLine("wazn roye Mah =  " + waznMah); // نمایش وزن روی ماه
        }
    }
}
```

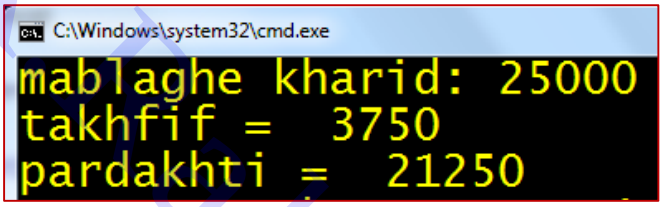
درخروجی برنامه می بینیم که با وارد کردن وزن ۶۱ کیلوگرم روی زمین، وزن ۱۰,۱۶ را روی ماه خواهیم داشت:



تمرین عملی:یک فروشگاه پوشاک،اجناس خود را با ۱۵ درصد تخفیف، حراج کرده است، برنامه ای بنویسید که مبلغ قبل از تخفیف جنس را از ورودی دریافت کند و سپس قیمت بعد از تخفیف را محاسبه و نمایش دهد.

```
using System;
namespace Shop
{
    class Program
    {
        static void Main(string[] args)
        {
            int mablagh, takhfif, pardakhti;    // متغیرهای لازم
            string vorodi;                      // متغیر ورودی
            Console.Write("mablaghe kharid: "); // نمایش پیام ورودی
            vorodi = Console.ReadLine();        // خواندن مبلغ خرید
            mablagh = int.Parse(vorodi);     // تبدیل ورودی رشته ای به اعشاری
            takhfif = (mablagh * 15) / 100;         // محاسبه تخفیف
            pardakhti = mablagh - takhfif;     // محاسبه خالص پرداختی
            Console.WriteLine("takhfif =  " + takhfif);    // نمایش تخفیف
            Console.WriteLine("pardakhti =  " + pardakhti); // نمایش خالص پرداختی
        }
    }
}
```

برای محاسبه تخفیف ۱۵٪، مبلغ خرید را در ۱۵ ضرب کرده و بر ۱۰۰ تقسیم می کنیم. خروجی برنامه را به ازای مبلغ خرید ۲۵۰۰۰ و تخفیف ۱۵٪مشاهده می کنید:

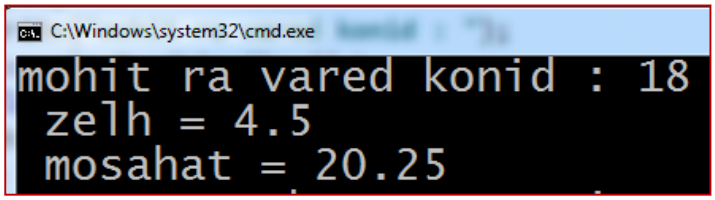


تمرین: مبلغ تخفیف و خرید دو گوشی موبایل را از ورودی بخوانید و مبلغ پرداختی را محاسبه و نمایش دهید.

مثال: فرض کنید اندازه دور(محیط) یک باغ مربع شکل را می دانیم، برنامه ای بنویسید که محیط را سؤال کند و سپس اندازه ضلع و اندازه مساحت باغ را محاسبه نماید.

```
static void Main(string[] args)
{
    float mohit, zelh, mosahat; // متغیرهای محیط و مساحت و ضلع
    string input;
    Console.Write("mohit ra vared konid : ");
    input = Console.ReadLine(); // ورودی محیط
    mohit = float.Parse(input); // تبدیل به عدد اعشاری
    zelh = mohit / 4;  // محاسبه اندازه ضلع
    mosahat = zelh * zelh; // محاسبه مساحت
    Console.WriteLine(" zelh = " + zelh);
    Console.WriteLine(" mosahat = " + mosahat);
}
```

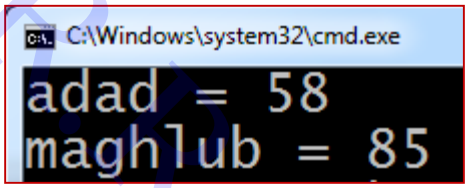
خروجی برنامه به ازای مقدار محیط ۱۸ بصورت زیر است:



مثال: برنامه ای بنویسید که مقلوب یک عدد صحیح دو رقمی دریافتی از کاربر را نمایش دهد

```
static void Main(string[] args) // برنامه مقلوب عدد دو رقمی
{
    int n = 58; // عدد دو رقمی دلخواه
    int r1, r2,m; // متغیرهای رقم اول و دوم و نیز مقلوب
    r1 = n % 10;  // بدست آوردن رقم یکان
    r2 = n / 10;  // بدست آوردن رقم دهگان
    m = r1 * 10 + r2; // عدد مقلوب(با تعویض رقم یکان و دهگان)ا
    Console.WriteLine("adad = " + n);
    Console.WriteLine("maghlub = " + m);
}
```

خروجی برنامه به ازای عدد ۵۸ بصورت زیر است:



مثال: فرض کنید شما حسابدار یک اداره هستید، حقوق ماهانه یک کارمند را از ورودی دریافت کرده و بر اساس اطلاعات زیر، خالص پرداختی او را نمایش دهید.

الف) کسورات(کم کننده):

- مالیات : ٪۵
- بیمه : ٪۳
- وام: ٪۱۰

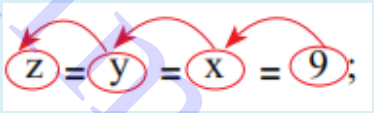
ب) اضافات:

- پاداش: ٪۴
- عیدی : مبلغ ۵۰۰۰۰۰

```
static void Main(string[] args)
{
    int salary,pardakhti; // حقوق ماهانه، پرداختی
    int m,b,w; // مالیات، بیمه، وام
    int p, eidi = 50000;// پاداش، عیدی
    string vorodi;
    Console.Write("Hoghoq ra wared konid: ");
    vorodi = Console.ReadLine(); // دریافت ورودی
    salary = int.Parse(vorodi); // تبدیل به عدد
    m = (salary * 5) / 100; // محاسبه مالیات
    b = (salary * 3) / 100; // محاسبه بیمه
    w = (salary * 10) / 100; // محاسبه وام
    p = (salary * 4) / 100; // محاسبه پاداش
    pardakhti = salary - m - b - w + p + eidi; // محاسبه پرداختی
    Console.WriteLine("wam = "+w); // نمایش نتایج
    Console.WriteLine("bime = " + b);
    Console.WriteLine("maliat = " + m);
    Console.WriteLine("padash = " + p);
    Console.WriteLine("eidi = " + eidi);
    Console.WriteLine("pardakhti = " + pardakhti);
}
```

عملگر انتساب

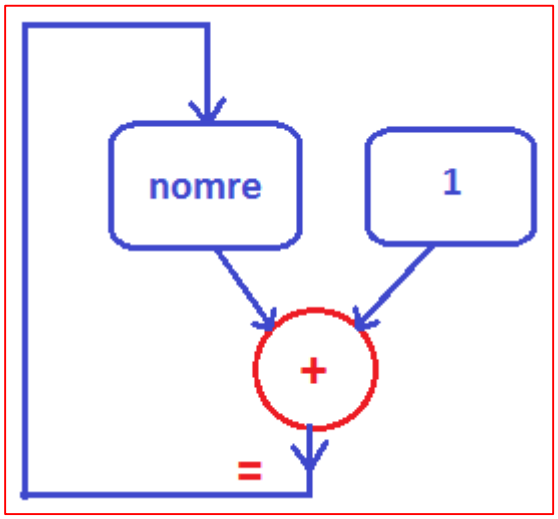
توسط عملگر انتساب(=)، محتوای یک متغیر را تغییر می دهیم یا مقداری در آن ذخیره می کنیم. دقت کنید که ظرف مقصد همیشه در سمت چپ انتساب قرار می گیرد:

قالب عمل انتساب	متغیر = مقدار ;
تعریف دو متغیر صحیح	`int x , y ;`
ذخیره مقدار ۶ در متغیر x	`x = 6 ;`
کپی مقدار متغیر x در متغیر y، پس y هم برابر با ۶ می شود، اما مقدار x تغییر نمی کند	`y = x ;`
تعریف متغیر s از نوع string و ذخیره مقدار رشته ای "salam" در آن	`string s = "salam";`
انتساب چندگانه: همزمان همه متغیرهای x , y , z مقدار ۹ را خواهند داشت	`int z = y = x = 9;` 

فرض کنید بخواهید به نمره درس خود، یک نمره اضافه کنید. از دستور زیر می توان کمک گرفت:

```
float  nomre = 17.5f;
nomre = nomre +1;
```

برای اجرای این دستور، ابتدا سمت راست انتساب(علامت =) محاسبه شده و نتیجه که ۱۸,۵ است در متغیر سمت چپ قرار خواهد گرفت.



دستور جایگزین برای این حالت، استفاده از عملگرهای انتساب به همراه عملگرهای ریاضی است:

```
nomre  +=  1;
```

این دستور دقیقاً معادل دستوری است که قبلاً نوشتیم. در این حالت، عملگر ریاضی را قبل از علامت = نوشته و سپس مقداری که می خواهیم در محاسبه شرکت کند، می نویسیم. به این نمونه ها دقت کنید:

```
int  x = 15;
```

شرح	نتیجه	دستور معادل	دستور ترکیب انتساب با عملگرهای ریاضی
افزایش متغیر به مقدار ۵	x = 20	x = x +5	x +=5
کاهش متغیر به مقدار ۲	x = 13	x = x - 2	x -= 2
ضرب متغیر در مقدار ۳	x = 45	x = x * 3	x *= 3
تقسیم متغیر بر مقدار ۳	x = 5	x = x / 3	x /= 3
باقیمانده تقسیم متغیر بر مقدار ۲	x = 1	x = x % 2	x %= 2

با توجه به جدول فوق در می یابیم که نخست عمل مورد نظر انجام شده و سپس نتیجه در متغیر سمت چپ قرار می گیرد.

مثلاً اگر بخواهیم به نمره خود مقدار ۲ را اضافه کنیم باید بنویسیم :

```
nomre += 2;
```

تمرین: به نظر شما فرق سه دستور زیر چیست؟

```
nomre  ++ ;
nomre  += 1 ;
nomre = nomre +1 ;
```

توجه: چون نتیجه کار در متغیر چپ ذخیره می شود، سمت چپ این عملگرهای ترکیبی حتماً باید متغیر(ظرف) باشد نه عدد ثابت:

nomre += 2; →درست

2 += nomre ; →نادرست

مثال: فرض کنید می خواهید مبلغی را از حساب خود به حساب یک نفر دیگر انتقال دهید. مبلغ موجودی خود، موجودی طرف مقابل و مبلغ انتقال را دریافت کرده و انتقال را انجام دهید.

```
static void Main(string[] args)
{
    int mowjudiMan, mowjudiTo,mabglagh;
    string input;
    Console.Write("Mowjudi khodam: ");
    input = Console.ReadLine(); // دریافت موجودی خود
    mowjudiMan = int.Parse(input); // تبدیل به عدد

    Console.Write("Mowjudi to: ");
    input = Console.ReadLine(); // دریافت موجودی طرف
    mowjudiTo = int.Parse(input); // تبدیل به عدد

    Console.Write("mablaghe enteghal: ");
    input = Console.ReadLine(); // دریافت مبلغ انتقال
    mabglagh = int.Parse(input); // تبدیل به عدد

    mowjudiMan -= mabglagh; // کم کردن مبلغ انتقال از حساب خود
    mowjudiTo += mabglagh; //افزودن مبلغ انتقال به حساب طرف
    Console.WriteLine("Mowjudi khodam=  "+mowjudiMan); // نمایش موجودی خود
    Console.WriteLine("Mowjudi to=  " + mowjudiTo); // موجودی طرف مقابل
}
```

نمونه خروجی برنامه:

```
Mowjudi khodam: 120000
Mowjudi to: 30000
mablaghe enteghal: 45000
Mowjudi khodam=  75000
Mowjudi to=  75000
```

مثال: برنامه ای بنویسید که دو عدد از ورودی دریافت نماید و مقادیر درون دو متغیر را جابجا کند :

```
static void Main(string[] args) // برنامه جابجایی دو متغیر
{
    int a = 5; // متغیر اول
    int b = 7; // متغیر دوم
    int c;   // ظرف کمکی
    c = a;  // c <-- a
    a = b;
    b = c;
    Console.WriteLine("a = " + a + " , b = " + b);
}
```

تمرین: نتیجه اجرای دستورات زیر چیست؟(مقدار نهایی x , y)

```
int x = 5;
int y = 10;
x += y;
y = x - y;
x -= y;
```

فصل چهارم- دستورهای شرطی

عبارت منطقی(شرط): عبارتی است که نتیجه آن یکی از دو حالت درست(true) یا نادرست(false) باشد. مثلاً : (6<5)یک عبارت منطقی با نتیجه false است، چون ۶ از ۵ کوچکتر نیست.

عملگرهای مقایسه ای: برای مقایسه عملوندها از این عملگرها استفاده می شود و نتیجه آنها درست یا نادرست می باشد.

در جدول روبرو عملگرهای مقایسه را مشاهده می کنید:

عملگر	کاربرد	مثال	نتیجه
<	کوچکتر	5< 6	درست
<=	کوچکتر یا مساوی	5<=5	درست
>	بزرگتر	12>12	نادرست
>=	بزرگتر یا مساوی	4>=9	نادرست
!=	مخالف	3 !=4	درست
==	تساوی	5 == 5	درست

نکته : حاصل عبارت های منطقی را می توانید بر روی صفحه خروجی نمایش دهید و یا در داخل متغیرهایی از نوع bool ذخیره کنید.

عملگرهای منطقی: برای ترکیب شرطها بکار می روند و نتیجه آنها درست یا نادرست است.

عملگر	کاربرد	مثال(a = 5, b = 7)	نتیجه
&&	ترکیب منطقی And «و»: در صورت درست بودن هر دو طرف درست است و گرنه نادرست	(a<5) && (b==7)	نادرست
||	ترکیب منطقی OR «یا»: اگر حداقل یکی از طرفین درست باشد، درست است و گرنه نادرست	(a<5) || (b==7)	درست
!	نقیض یا برعکس(Not)	! (a<5)	درست
^	«یا» انحصاری Xor: اگر دو شرط مخالف هم باشند، جواب درست است.	(a<5) ^ (b==7)	درست

جدول زیر، نمونه هایی از ترکیب شرط را نشان می دهد

مثال عملگرهای منطقی					
A	**B**	**A || B**	**A && B**	**A ^ B**	**! A**
false	**false**	false	false	false	true
true	**false**	true	false	true	false
false	**true**	true	false	true	true
true	**true**	true	true	false	false

مثال: قد(برحسب متر) و وزن (برحسب کیلوگرم) را از ورودی خوانده و تناسب قد به وزن را با محاسبه نسبت زیر بررسی کنید. وزن را بر قد به توان ۲ تقسیم کنید:

نسبت = وزن (کیلوگرم) / (قد بر حسب متر)^2

- *نسبت کمتر از ۲۰: لاغر*
- *نسبت بیشتر از ۲۵ : چاق*
- *نسبت بین ۲۰ تا ۲۵: نرمال*

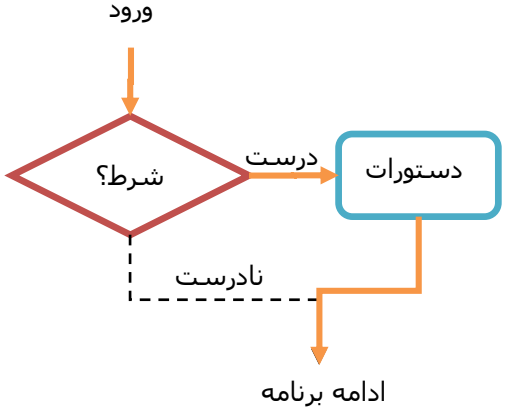
```
using System;
namespace Fitness
{
    class Program
    {
        static void Main(string[] args) // برنامه تناسب وزن
        {
            float wazn, qad, nesbat; // متغیرهای وزن، قد و نسبت
            string input;
            Console.Write("wazn(kilo gram): ");
            input = Console.ReadLine(); // ورودی وزن
            wazn = float.Parse(input);
            Console.Write("Qad (meter): ");
            input = Console.ReadLine(); //ورودی قد
            qad = float.Parse(input);
            nesbat = wazn / (qad * qad); // محاسبه نسبت
            Console.WriteLine("nesbat : " + nesbat ); // نمایش نسبت
            Console.WriteLine("Chaq : " + (nesbat > 25) ); // شرط چاقی
            Console.WriteLine("Laghar : " + (nesbat < 20)); // شرط لاغری
            Console.WriteLine("Motanaseb : " + ((nesbat >= 20) && (nesbat < 25)));// عادی
        }
    }
}
```

```
C:\Windows\system32\cmd.exe
wazn(kilo gram): 62
Qad (meter): 1.68
nesbat : 21.96712
Chaq : False
Laghar : False
Motanaseb : True
```

می بینیم که اگر وزن ما ۶۲ کیلوگرم و قد ۱,۶۸ باشد، نسبت مقدار ۲۱,۹۶ بوده و در محدوده نرمال می باشد(true) و محدوده چاق و لاغر false شده است.

۱- دستور شرطی if

حالت معمولی و ساده دستورif بصورت زیر می باشد که در این حالت اگر شرط برقرار باشد، دستور مورد نظر انجام می شود و در غیر اینصورت اتفاقی نمی افتد

if (شرط مورد نظر) دستور مورد نظر; *مثال:* *if (فردا برف نبارد) مسابقه فوتبال برگزار می گردد*	

در برنامه نویسی مواردی پیش می آید که بخواهیم دستور یا دستوراتی، هنگامی که شرط خاصی برقرار است، توسط برنامه به اجرا در آید. این مورد در زندگی روزمره نیز دیده می شود؛ به عنوان مثال " اگر فردا برف نبارد، مسابقه فوتبال برگزار می گردد." شرط مورد نظر نباریدن برف است و عملی که قرار است انجام شود برگزاری مسابقه می باشد.

☜ اگر بخواهیم در صورت برقراری شرط، چندین دستور اجرا شوند، از شکل زیر استفاده می کنیم یعنی دستورات را بین {} قرار می دهیم. در صورتیکه تنها یک دستور داشته باشیم، نوشتن {} اجباری نیست.

```
if(شرطموردنظر){
دستورشماره۱;
دستورشماره ۲;
دستورشماره۳;
دستورشماره.....;
}
```

مثال: دو عدد را در نظر گرفته و مشخص کنید که آیا اولی بزرگتر است؟

```
using System;
namespace SimpleIf
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 17, b = 15;
            if (a > b)
                Console.WriteLine(" a is bigger than b.");
        }
    }
}
```

در این مثال، چون مقدار a از b بزرگتر است، پیام نمایش داده می شود:

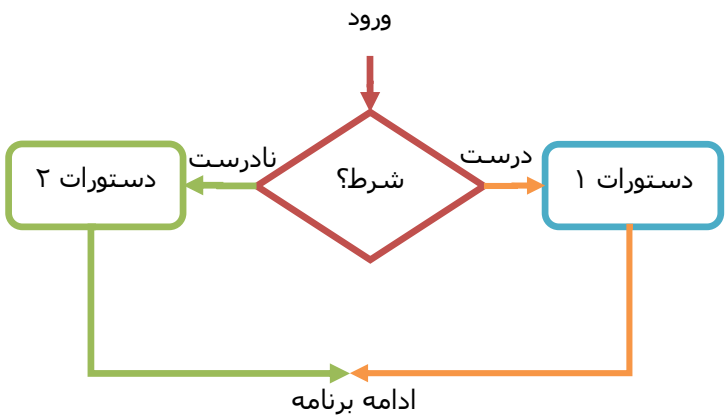
اگر مقدار a را مثلاً به عدد ۱۰ تغییر دهیم، پیامی نمایش داده نمی شود: چون شرط درست نیست.

۲-دستور شرطی کامل بصورت if-else

گاهی اوقات نیاز داریم که در صورت برقرار بودن شرط خاصی یک سری از دستورات اجرا و در صورت برقرار نبودن آن شرط، دسته ای دیگر از دستورات اجرا گردند. به عنوان مثال "*اگر فردا برف نبارد، مسابقه فوتبال برگزار می گردد، در غیر اینصورت مسابقه برگزار نخواهد شد.*"

☜در این حالت از دستور، اگر شرط درست باشد، دستورات ۱ و در غیر اینصورت دستورات ۲ اجرا می شوند. دقت کنید از بین دستورات ۱ و ۲ فقط یکی اجرا خواهد شد.

```
if (شرط مورد نظر) {
دستورات ۱
}
else{
دستورات ۲
}
```



مثال: دو عدد را در نظر گرفته و مشخص کنید که آیا اولی بزرگتر است یا دومی؟

```
using System;
namespace SimpleIf
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 17, b = 15;
            if (a > b)
                Console.WriteLine(" a is bigger than b.");
            else
                Console.WriteLine(" b is bigger than a.");
        }
    }
}
```

در این مثال، چون شرط برقرار است، دستور مقابل if اجرا می شود و پیام زیر نمایش داده می شود:

a is bigger than b.

اگر مقدار a مثلاً ۱۰ باشد، چون شرط درست نیست، دستور مقابل else اجرا می گردد یعنی پیام زیر نمایش داده می شود:

b is bigger than a.

نکات مهم:

- در مقابل شرط دستور if نباید علامت ; گذاشت، بلکه در آخر دستور متناظر آن باید علامت ; را قرار داد.

```
if (عبارت منطقی)  ← نقطه ویرگول ندارد
دستور;
```

- در مقایسه تساوی دستور if حتماً باید از علامت == برای مقایسه مساوی بودن استفاده نمود و قرار دادن یک علامت = باعث بروز خطا می شود.

if (pass == "12345") → *شیوه صحیح مقایسه تساوی*

if (pass = "12345") → *شیوه غلط مقایسه تساوی*

مثال: برنامه ای بنویسید که رمز ورودی را درخواست کرده و آنرا با رمز دلخواه خود مثلاً ۱۲۳۴۵ مقایسه نموده و در صورت درست یا نادرست بودن، پیام مناسب نمایش دهد:

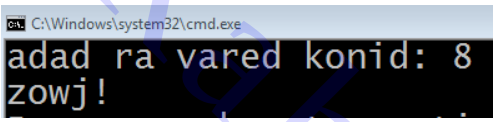
```
using System;
namespace Password
{
    class Program
    {
        static void Main(string[] args) // برنامه تشخیص رمز
        {
            string pass;
            Console.Write("ramz ra vared konid: ");
            pass = Console.ReadLine(); // خواندن رمز
            if (pass == "12345") // بررسی مساوی بودن رمز با ۱۲۳۴۵
                Console.WriteLine("ramz dorost ast!");
            else
                Console.WriteLine("ramz eshtebah ast!!");

        }
    }
}
```

مثال: برنامه ای طراحی کنید که عددی از ورودی خوانده و زوج و فرد بودن آنها را بررسی کند.

```
using System;
namespace OddEven
{
    class Program // برنامه تشخیص عدد زوج و فرد
    {
        static void Main(string[] args)
        {
            string input; // متغیر ورودی
            int n, r; // متغیر عدد و باقیمانده
            Console.Write("adad ra vared konid: ");
            input = Console.ReadLine(); // خواندن عدد
            n = int.Parse(input); // تبدیل ورودی به عدد صحیح
            r = n % 2; // محاسبه باقیمانده بر ۲
            if (r == 0) // بررسی شرط زوج بودن
                Console.WriteLine("zowj!");
            else
                Console.WriteLine("fard!");
        }
    }
}
```

زوج بودن به این معنا است که عدد بر ۲ بخش پذیر باشد؛ یعنی باقیمانده آن عدد بر ۲ برابر صفر باشد. پس اگر باقیمانده عدد بر ۲ صفر شد(با استفاده از عملگر %)، عدد زوج و گرنه فرد است.

با اجرای برنامه و وارد کردن اعداد مختلف، زوج و فرد بودن را مشخص می کند:



بلاک: به تعدادی دستور که داخل علامت های { } قرار داشته باشند بلاک گفته می شود.

- بلاک می تواند خالی(یعنی بین علامت های { } ، هیچ دستوری وجود نداشته باشد) یا فقط شامل یک دستور یا چند دستور باشد .
- برای خوانا و واضح شدن یک بلاک، دستورهای داخل بلاک را با تورفتگی می نویسیم تا به راحتی در برنامه مشخص شوند

```
{
        دستور ;
        دستور ;
        دستور ;
}
```

مثال: فرض کنید می خواهید مبلغی را از حساب خود به حساب یک نفر دیگر انتقال دهید. مبلغ موجودی خود، موجودی طرف مقابل و مبلغ انتقال را دریافت کرده و انتقال را انجام دهید. کنترل کافی بودن موجودی را هم انجام دهید.

نمونه خروجی اگر موجودی کافی باشد:

```
Mowjudi khodam: 120000
Mowjudi to: 30000
mablaghe enteghal: 45000
Mowjudi khodam=  75000
Mowjudi to=  75000
Press any key to continue . . .
```

اگر موجودی کافی نباشد:

```
Mowjudi khodam: 120000
Mowjudi to: 30000
mablaghe enteghal: 150000
Mowjudi kafi nist!!
Press any key to continue . . .
```

```
static void Main(string[] args)
{
    int mowjudiMan, mowjudiTo,mabglagh;
    string input;
    Console.Write("Mowjudi khodam: ");
    input = Console.ReadLine(); // دریافت ورودی
    mowjudiMan = int.Parse(input); // تبدیل به عدد

    Console.Write("Mowjudi to: ");
    input = Console.ReadLine(); // دریافت ورودی
    mowjudiTo = int.Parse(input); // تبدیل به عدد

    Console.Write("mablaghe enteghal: ");
    input = Console.ReadLine(); // دریافت ورودی
    mabglagh = int.Parse(input); // تبدیل به عدد
    if (mowjudiMan >= mabglagh) // اگر موجودی از مبلغ انتقال بیشتر است
    {
        mowjudiMan -= mabglagh; // کم کردن مبلغ انتقال از حساب خود
        mowjudiTo += mabglagh; //افزودن مبلغ انتقال به حساب طرف
        Console.BackgroundColor = ConsoleColor.Green;
        Console.WriteLine("Mowjudi khodam= " + mowjudiMan);
        Console.WriteLine("Mowjudi to=  " + mowjudiTo);
        Console.BackgroundColor = ConsoleColor.White;
    }
    else{
        Console.BackgroundColor = ConsoleColor.Red;
        Console.WriteLine("Mowjudi Kafi nist!!");
        Console.BackgroundColor = ConsoleColor.White;
    }
}
```

مثال: نمره یک درس را خوانده و معتبر بودن(عدد اعشاری در محدوده بین صفر تا ۲۰) و سپس قبولی و مردودی را مشخص نمایید.

اگر کاربر هنگام ورود نمره، اشتباهی عدد وارد نکند(مثلاً به جای نمره 15 مقدار 15t وارد شود)، برنامه با خطا مواجه شده و متوقف می گردد. برای بررسی این موضوع به جای متد ( )Parse، از متد ( )TryParse به شکل زیر استفاده می کنیم.

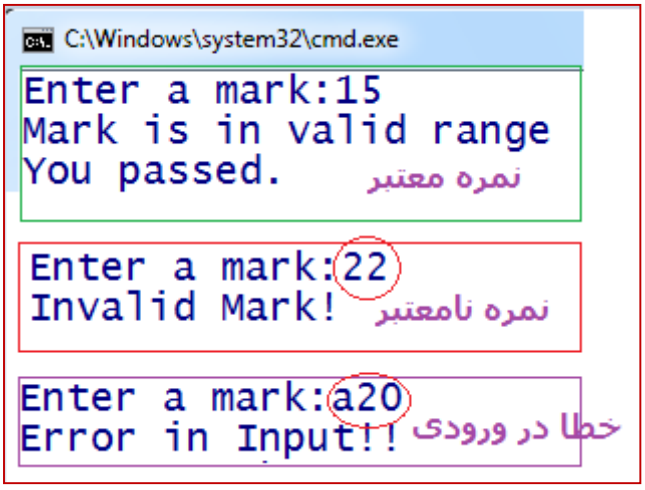
(متغیر عددی نتیجه out , متغیر ورودی رشته ای) TryParse .نوع داده مورد نظر

نتیجه عبارت فوق مقداری منطقی است(جواب درست یا نادرست) پس در دستور if می توان نتیجه آنرا بررسی نمود:

if (float.TryParse(input, out number)) // بررسی ورود عدد و تبدیل به عدد اعشاری

```
class Program
{
    static void Main(string[] args)
    {
        float number ; // متغیر نمره
        string input ;  // متغیر دریافت ورودی
        Console. Write ("Enter a mark:" );
        input = Console.ReadLine ( ); // خواندن نمره
        //number = float.Parse(input); // تبدیل به عدد اعشاری
        if (float.TryParse(input, out number)) // بررسی وارد شدن عدد و تبدیل به اعشاری
        {
            if ((number < 0) || (number > 20)) // اگر نمره کمتر از ۰ و بیشتر از ۲۰ باشد
                Console.WriteLine("Invalid Mark!"); // نمره نامعتبر
            else  // در غیر این صورت
            {
                Console.WriteLine("Mark is in valid range"); // نمره معتبر
                if (number < 10) // بررسی مردودی
                    Console.WriteLine("You failed, try harder next.");
                else // قبولی
                    Console.WriteLine("You passed.");
            }
        }
        else
            Console.WriteLine("Error in Input!!"); // خطا در ورودی
    }
}
```

چند نمونه مقدار ورودی و نتیجه آنها:



مثال: دو عدد را از ورودی خوانده و بیشترین آنها را نمایش دهید

```
static void Main(string[] args)
{
    int a,b,max;
    string input;  // متغیر دریافت ورودی
    Console.Write("enter a:");
    input = Console.ReadLine(); //a خواندن عدد
    a = int.Parse(input);
    Console.Write("enter b:");
    input = Console.ReadLine(); //b خواندن عدد
    b = int.Parse(input);
    if (a > b)
        max = a;
    else
        max = b;
    Console.WriteLine("max= ",max);
}
```

عملگر ؟

در زبان C#، برای پیاده سازی حالتی که بر اساس شرط، از بین دو مقدار یکی را انتخاب می کنیم، حالت ساده شده ای وجود دارد که به عملگر شرطی ؟ معروف است و جایگزینی برای دستور if-else می باشد.

```
if (a > b)
    max = a;
else
    max = b;
```

```
max = (a > b) ? a : b;
```

قالب کلی دستور به شکل زیر است:

*متغیر = **(شرط)** ? مقدار در صورت درست بودن شرط : مقدار در صورت نادرست بودن شرط*

در دستور زیر، اگر a > b باشد، a و در غیر این صورت مقدار b انتخاب شده و در متغیر max قرار می گیرد.

```
max = (a > b) ? a : b;
```

تمرین۱: دستور زیر چکاری انجام می دهد؟

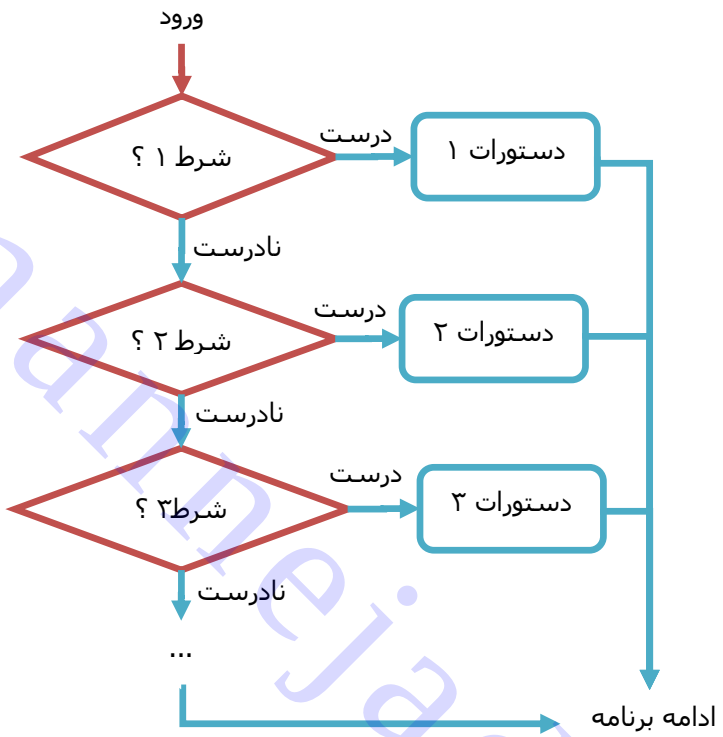
```
m = (m < 0) ? -m : m;
```

تمرین ۲: عددی را خوانده و به کمک دستورات if-else، مثبت، منفی یا صفر بودن آنرا با نوشتن پیغام مربوطه مشخص نمایید.

تمرین ۳: عددی را خوانده و به کمک عملگر شرطی ؟ وضعیت زوج و فرد بودن آنرا تعیین و جواب را نمایش دهید.

۳- if...else پیچیده و بصورت ترکیبی: هر گاه بخواهیم چندین شرط را بررسی کرده و بر اساس آنها دستورات متناظر را اجرا نماییم از این حالت استفاده می کنیم. اگر شرط اول درست بود، دستورات قسمت اول اجرا می شوند؛ در غیر این صورت به سراغ بررسی شرط دوم و ... می رود.

```
if( شرط ۱ ) {
        دستورات ۱
}
else if( شرط ۲ ){
        دستورات ۲
}
else if( شرط ۳ ){
        دستورات ۳
}
```



مثال: فرض کنید می خواهیم عددی(شماره فصل سال) دریافت کنیم و بر اساس آن نام فصل را بنویسیم. در این حالت لازم است چهار شرط را برای بررسی هر شماره فصل پیاده سازی کنیم.

```
using System;
namespace Season
{
    class Program
    {
        static void Main(string[] args)
        {
            int n;  // شماره فصل
            string input; // متغیر ورودی
            Console.Write("shomare fasl: ");
            input = Console.ReadLine(); // ورودی
            n = int.Parse(input); // تبدیل ورودی به عدد صحیح
            if(n == 1)  // اگر شماره برابر با ۱ بود
                Console.WriteLine ("Bahar");
            else if (n == 2)
                Console.WriteLine("Tabestan");
            else if (n == 3)
                Console.WriteLine("Payeez");
            else if (n == 4)
                Console.WriteLine("Zamestan");
            else  // اگر هیچ کدام از حالات فوق نبود
                Console.WriteLine("khata!!");
        }
    }
}
```

اگر مثلاً عدد ۳ را وارد کنیم، مقدار Payeez نمایش داده می شود و اگر مقدار ۶ وارد شود، پیام Khata!! نمایش می یابد.

تمرین: مانند مثال فوق، شماره روز هفته را از ورودی دریافت و نام روز را نمایش دهید.
(0: شنبه، ۱: یکشنبه، ...، ۶:جمعه)

دستور switch

switch بیانگر شرط در برنامه نویسی C# اما به شکل دیگری است. استفاده از دستورات if و else برای حالاتی مناسب تر است که ما دو احتمال بیشتر نداشته باشیم. حال اگر بخواهیم تعداد حالات شرط را افزایش دهیم از دستور switchاستفاده می کنیم. البته با استفاده از دستورات if و else نیز می توانیم حالات بیش از دو مورد را بسنجیم اما با دستور switch کد برنامه خواناتر است.

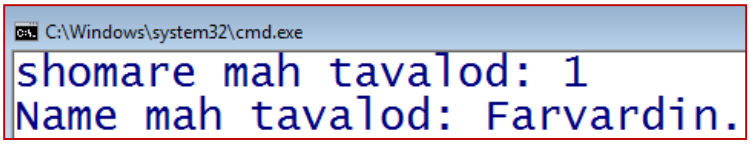
```
switch (عبارت)
{
    case مقدار۱:
        دستور۱;
        break;
    case : مقدار۲
        دستور۲;
        break;
    .
    .
    .
    default:
        دستورهای دیگر;
        break;
}
```

ساختار کلی دستور switch

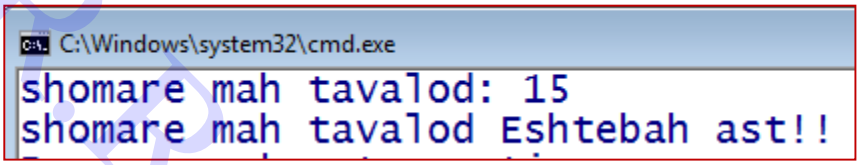
فرض کنیم می خواهیم برنامه ای بنویسیم که در ابتدای برنامه از کاربر می خواهیم که عدد مربوط به ماهی که در آن متولد شده است را وارد کند و برنامه نام ماه انتخابی را برای کاربر به نمایش درآورد.
ساختار switch همانند ساختار if است و به صورت switch( ){} نوشته می شود که داخل پرانتز باید پارامتر ورودی که در اینجا همان شماره ماه است وارد شود و داخل دو علامت {} باید case ها و یا "مواردی" را تعریف کنیم که بر اساس آن، نام ماه انتخابی مشخص شده و سپس روی صفحه مانیتور نمایش داده شود.

```
using System;
namespace BirthDay
{
    class Program
    {
        static void Main(string[] args)
        {
            int n;  //شماره ماه
            string input; //ورودی
            Console.Write("shomare mah tavalod: ");
            input = Console.ReadLine(); //دریافت ورودی
            n = int.Parse(input); // تبدیل ورودی به عدد صحیح
            switch (n) { //بررسی ورودی
                case 1:
                    Console.WriteLine("Name mah tavalod: Farvardin."); break;
                case 2:
                    Console.WriteLine("Name mah tavalod: Ordibehesht.");break;
                case 3:
                    Console.WriteLine("Name mah tavalod: Khordad.");break;
                case 4:
                    Console.WriteLine("Name mah tavalod: Tir.");break;
                case 5:
                    Console.WriteLine("Name mah tavalod: Mordad.");break;
                case 6:
                    Console.WriteLine("Name mah tavalod: Shahrivar.");break;
                case 7:
                    Console.WriteLine("Name mah tavalod: Mehr.");break;
                case 8:
                    Console.WriteLine("Name mah tavalod: Aban.");break;
                case 9:
                    Console.WriteLine("Name mah tavalod: Azar.");break;
                case 10:
                    Console.WriteLine("Name mah tavalod: Day.");break;
                case 11:
                    Console.WriteLine("Name mah tavalod: Bahman.");break;
                case 12:
                    Console.WriteLine("Name mah tavalod: Esfand.");break;
                default:
                    Console.WriteLine("shomare mah tavalod Eshtebah ast!!"); break;
            }
        }
    }
}
```

حال می توانیم برنامه خود را اجرا کنیم. یک عدد از ۱ تا ۱۲ را وارد می کنیم. به طور مثال عدد ۱ را وارد می کنیم سپس دکمه Enter را می زنیم.



و به ازای عدد ۱۵: شماره ماه اشتباه است!



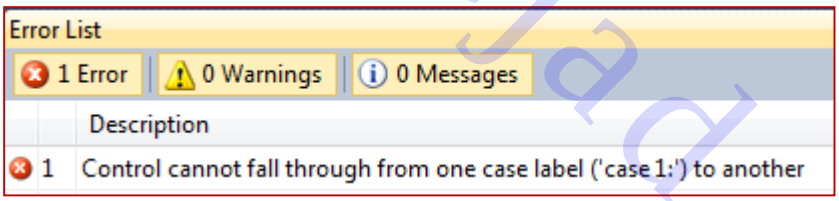
در اولین قدم، دستور switch می سنجد که ببیند آیا داده متغیرش معادل با case 1 می باشد یا خیر. اگر درست باشد، دستور متناظر آن انجام می شود. سپس دستور break; به برنامه می گوید که دیگر نیازی به اجرای دستورات بعدی نبوده و بلافاصله باید به آخر برنامه برود. اگر شرط اول درست نبود به سراغ موارد دیگر می رود.

سوال: اگر دستورات break را نداشته باشیم چه اتفاقی خواهد افتاد؟

برای همین منظور می توانیم دستورات break را به صورت Comment یا توضیحات در آوریم که در این صورت کد ما مثلاً در case 1 به شکل زیر در خواهد آمد:

```
case 1:
    Console.WriteLine("Name mah tavalod: Farvardin."); //break;
```

در این صورت با پیام خطای زیر مواجه می شویم:



این خطا به این معناست که کنترل برنامه از یک case روی دیگری نمی تواند قرار بگیرد! به عبارتی دیگر با نداشتن دستور break دستورات case ها مجزا نخواهند بود.

نکته ۱ : نوع عبارتی که داخل پرانتز دستور switch نوشته می شود، نمی تواند اعشاری باشد. اما عبارتهای حرفی، رشته ای و انواع داده صحیح می تواند استفاده گردد.

نکته ۲ : در صورتیکه بخواهیم حالتی را کنترل کنیم که هیچ کدام از موارد case صدق نمی کنند، دستورات را در قسمت default می نویسیم. مثلاً ماه با شماره غیر از ۱ تا ۱۲ وجود ندارد. پس در آخر موارد case می توان نوشت:

```
default: // حالتی که هیچ کدام از موارد صدق نکنند
    Console.WriteLine("shomare mah tavalod Eshtebah ast!!"); break;
```

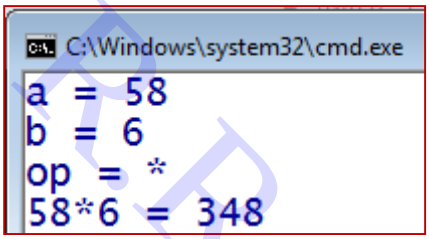
تمرین ۱: نام روز هفته را نیز علاوه بر دستور if-else با switch بنویسید.

تمرین ۲: یک عدد یک رقمی را دریافت کرده و به کمک دستور switch معادل حرفی آنرا نمایش دهید. مثلاً با وارد کردن عدد ۷ بنویسد «هفت».

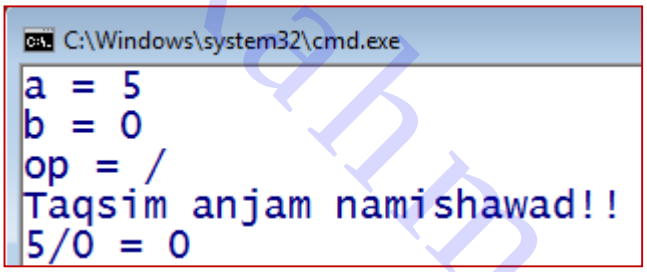
مثال: به کمک دستور switch یک ماشین حساب ساده با دو عملوند و ۵ عمل اصلی (+, -, *, /, %) طراحی کنید.

```
using System;
namespace Calculator
{
    class Program
    {
        static void Main(string[] args)
        {
            int a, b; // عملوند
            float r; // نتیجه
            string op; // عملگر
            string input; // ورودی
            Console.Write("a = ");
            input = Console.ReadLine();
            a = int.Parse(input); //a خواندن

            Console.Write("b = ");
            input = Console.ReadLine();
            b = int.Parse(input);//b خواندن

            Console.Write("op = ");
            op = Console.ReadLine();// خواندن عملگر
            r = 0;
            switch (op) { // انجام عمل بر اساس عملگر
                case "+": r = a + b; break;
                case "-": r = a - b; break;
                case "*": r = a * b; break;
                case "/":
                    if (b == 0){ // بررسی اینکه مخرج صفر نباشد
                        Console.WriteLine("Taqsim anjam namishawad!!");
                        r = 0;
                    }
                    else
                        r = a / b;
                    break;
                case "%":
                    if (b == 0){
                        Console.WriteLine(" anjam namishawad!!");
                        r = 0;
                    }
                    else
                        r = a % b;
                    break;
            } //switch پایان
            Console.WriteLine(a + op + b + " = " + r);
        }
    }
}
```

در خروجی زیر عمل ضرب ۶*۵۸ را می بینید

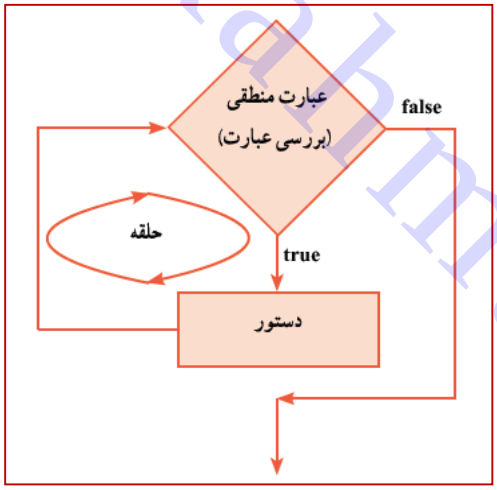


و به ازای تقسیم ۵/۰ ، انجام نشدن عمل به دلیل تقسیم بر صفر را نشان می دهد.



فصل پنجم – دستورات تکرار(حلقه ها)

مفهوم حلقه: حلقه ها برای اجرای یک یا چند دستورالعمل به تعداد دفعات مشخص یا تا زمان رسیدن به شرط خاصی به کار برده می شوند. حلقه های زبان C# عبارتند از for ، while و do-while



حلقه while: در حلقه while شرط خود و یا همان نقطه پایانی حلقه را داخل پرانتز نوشته و در داخل دو علامت {} که مقابل while() قرار می گیرد دستورات حلقه و نیز مقدار افزایش(کاهش) متغیر را تعریف کنیم. ساختار این حلقه مطابق زیر است:

```
while (عبارت منطقی)
    دستور ;
```

توجه کنید که در آخر خط دستور while علامت ; قرار ندهید:

```
while (عبارت منطقی)
    دستور ;
```
نقطه ویرگول ندارد

مثال: اعداد ۱ تا ۱۰۰ را نمایش دهید

```
using System;
namespace while1
{
    class Program
    {
        static void Main(string[] args) //نمایش اعداد ۱ تا ۱۰۰
        {
            int x = 1;  // نقطه شروع
            while (x <= 100) // تا زمانیکه به ۱۰۰ نرسیده است ادامه بده
            {
                Console.WriteLine("x = " + x); // نمایش عدد
                x++; // عدد بعدی
            }
        }
    }
}
```

مثال ۲: نمایش اعداد زوج بین ۱ تا ۱۰۰

```
using System;
namespace WhileEven
{
    class Program
    {
        static void Main(string[] args) //نمایش اعداد زوج ۱ تا ۱۰۰
        {
            int x = 2;  // نقطه شروع
            while (x <= 100) // تا زمانیکه به ۱۰۰ نرسیده است ادامه بده
            {
                Console.WriteLine("x = " + x); // نمایش عدد
                x += 2; // عدد زوج بعدی
            }
        }
    }
}
```

تمرین ۱: نمایش اعداد فرد کمتر از ۱۰۰

تمرین ۲: نمایش اعداد مضرب ۵ کمتر از ۱۰۰۰

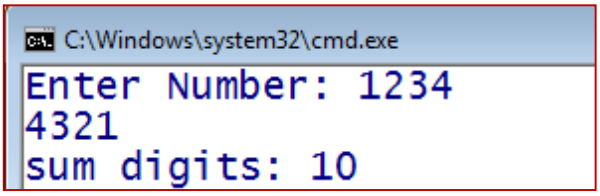
تمرین ۳: برنامه ای طراحی کنید که مجموع اعداد ۱ تا ۱۰۰ را بدست آورده و نمایش دهد.

مثال: برنامه ای بنویسید که ارقام یک عدد مفروض را بصورت معکوس چاپ نموده و مجموع آنها را نیز نمایش دهد.

حل: برای جدا کردن هر رقم، می توان از باقیمانده عدد بر ۱۰ استفاده نمود. و هر بار عدد مربوطه را بر ۱۰ تقسیم نماییم تا به صفر برسد

عدد n	تقسیم بر ۱۰	باقیمانده بر ۱۰ r	مجموع sum
1234	123	4	0+4 = 4
	12	3	4+3 = 7
	1	2	7+2 = 9
	0	1	9+1=10

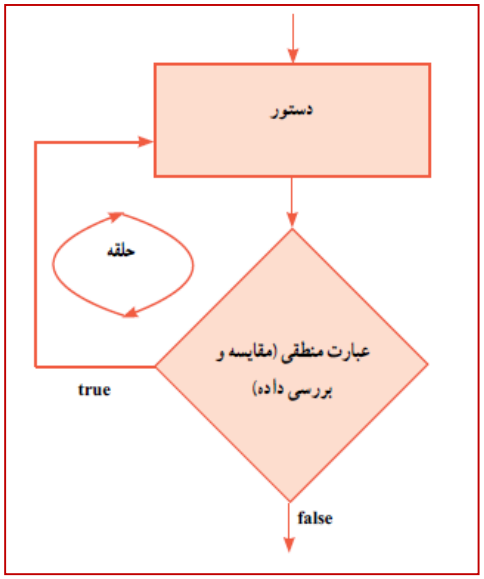
نمونه خروجی برای عدد ۱۲۳۴:



```
using System;
namespace SumDigits
{
    class Program
    {
        static void Main(string[] args)// نمایش ارقام عدد بصورت برعکس
        {
            int n; // عدد ورودی
            int r; // هر رقم جدا شده
            int sum = 0; // مجموع رقم
            string input; //ورودی
            Console.Write("Enter Number: ");
            input = Console.ReadLine();// خواندن عدد
            n = int.Parse(input); // تبدیل ورودی به عدد
            while (n > 0) // تا زمانیکه عدد صفر نیست
            {
                r = n % 10; // بدست آوردن رقم یکان
                Console.Write(r); // نمایش رقم
                sum += r; // جمع رقم
                n = n / 10; // ادامه با بقیه رقمها
            }
            Console.WriteLine();
            Console.WriteLine("sum digits: " + sum);// نمایش جمع رقم
        }
    }
}
```

حلقه تکرار do-while

در این نوع حلقه تکرار که مشابه حلقه while می باشد، شرط در آخر حلقه تست می شود و حتی اگر شرط هم درست نباشد، حداقل یکبار دستورات داخل آن اجرا می شوند؛ چون قبل از بررسی شرط دستورات نوشته شده اند.



مثال: می خواهیم برنامه ای بنویسیم که نام کاربری و رمز عبور را سؤال نماید و اگر کاربر اطلاعات خواسته شده را به درستی وارد نکرد، دوباره سؤال شود.

```
using System;
namespace DoWhileLogin2
{
    class Program
    {
        static void Main(string[] args)
        {
            string pass, username;
            bool ok = false; // بررسی درستی رمز
            do     // حلقه تکرار
            {
                Console.Write("Enter username: ");
                username = Console.ReadLine();  // خواندن نام کاربر
                Console.Write("Enter password: ");
                pass = Console.ReadLine();   // خواندن رمز
                if ((pass == "1234") && (username == "admin")) // بررسی درستی
                {
                    ok = true;  // نام کاربر و رمز درست است
                    Console.WriteLine("Welcome to program! "); // پیام خوش آمد
                }
                else // اگر نام کاربر یا رمز درست نباشد
                {
                    Console.WriteLine("Invalid username or password: "); // اشتباه
                    Console.WriteLine("try again... "); // سعی مجدد
                    Console.WriteLine(); // خط خالی
                }

            } while (! ok); // تا زمانیکه پاسخ درست نیست
        }
    }
}
```

در این مثال نام کاربر admin و رمز ۱۲۳۴ تعین شده است.

نمونه ای از خروجی که تا نام کاربر و رمز درست وارد نشود، ادامه می دهد:

```
Enter username: 1
Enter password: 2
Invalid username or password:
try again...

Enter username: 123
Enter password: 123
Invalid username or password:
try again...

Enter username: admin
Enter password: 1234
Welcome to program!
```

تمرین: تغییراتی در برنامه بدهید که فقط ۳ بار کاربر مجاز به وارد کردن نام و رمز باشد.

(راهنمایی: متغیری عددی تعریف کنید و هر بار رمز وارد شود، یکی اضافه شود، سپس مقدار آنرا بررسی کنید)

مثال: می خواهیم یک بازی حدس عدد، ایجاد کنیم. این بازی بین دو بازیکن به شرح زیر صورت می گیرد. بازیکن اول عددی(بین ۱ تا ۱۰۰) را برای خود در نظر می گیرد و بازیکن دوم باید آن عدد را حدس بزند. بازیکن اول در طول بازی، راهنمایی لازم را در اختیار بازیکن دوم قرار می دهد تا عدد بالاتر یا پایین تری را حدس بزند. تعداد حدسها را هم نشان دهید.

```
using System;
namespace GuessNumber
{
    class Program
    {
        static void Main(string[] args)
        {
            string input; // ورودی
            int number = 0, guess, secret; // تعداد، حدس کاربر و عدد سرّی
            Console.Write("Player 1: Think a number (1-100): ");
            input = Console.ReadLine();
            secret = int.Parse(input); // خواندن عدد سری
            Console.Clear(); // پاک کردن صفحه
            do //حلقه تکرار
            {
                Console.Write("Player 2: Guess the number (1-100): ");
                input = Console.ReadLine();
                guess = int.Parse(input); // خواندن حدس کاربر
                number++; // شمارش حدس
                Console.WriteLine("\t Guesses = " + number ); // نمایش حدسها
                if (guess < secret) // اگر عدد حدس زده شده کوچکتر بود
                    Console.WriteLine("\t" + guess + " < " + " ? ");
                else if (guess > secret) // اگر عدد حدس زده شده بزرگتر بود
                    Console.WriteLine("\t" + guess + " > " + " ? ");
                else // اگر عدد حدس زده شده مساوی بود
                    Console.WriteLine("\t Well done!, it's correct!");
            } while (guess != secret); // تا زمانیکه حدس درست نباشد
        }
    }
}
```

تمرین: تعداد حدسها را محدود نمایید. مثلاً اگر بیش از ۷ بار حدس زد و پیدا نکرد بازنده است.

```
Player 2: Guess the number (1-100): 25
          Guesses = 1
         25 <  ?
Player 2: Guess the number (1-100): 90
          Guesses = 2
         90 >  ?
Player 2: Guess the number (1-100): 70
          Guesses = 3
         70 <  ?
Player 2: Guess the number (1-100): 80
          Guesses = 4
         80 >  ?
Player 2: Guess the number (1-100): 75
          Guesses = 5
          Well done!, it?s correct!
```

مقایسه دو حلقه while و do-while:

نوع حلقه	محل قرار گیری شرط	اگر شرط از اول برقرار نباشد	ساختار حلقه

while	قبل از دستورات حلقه	هیچ یک از دستورات اصلاً اجرا نمی شوند	while(شرط){
			دستورات;
			}

do- while	بعد از دستورات حلقه	دستورات حداقل یکبار اجرا می شوند	do { دستورات; } while(شرط);

مثال: برنامه ای بنویسید که نمرات درس برنامه سازی دانش آموزان یک کلاس ۱۵ نفری را سؤال نماید و سپس اطلاعات زیر را نمایش دهد:

الف) بالاترین نمره کلاس

ب) کمترین نمره کلاس

ج) فاصله بین کمترین و بیشترین نمره

د) مجموع نمرات کلاس

ه) میانگین یا معدل نمرات کلاس

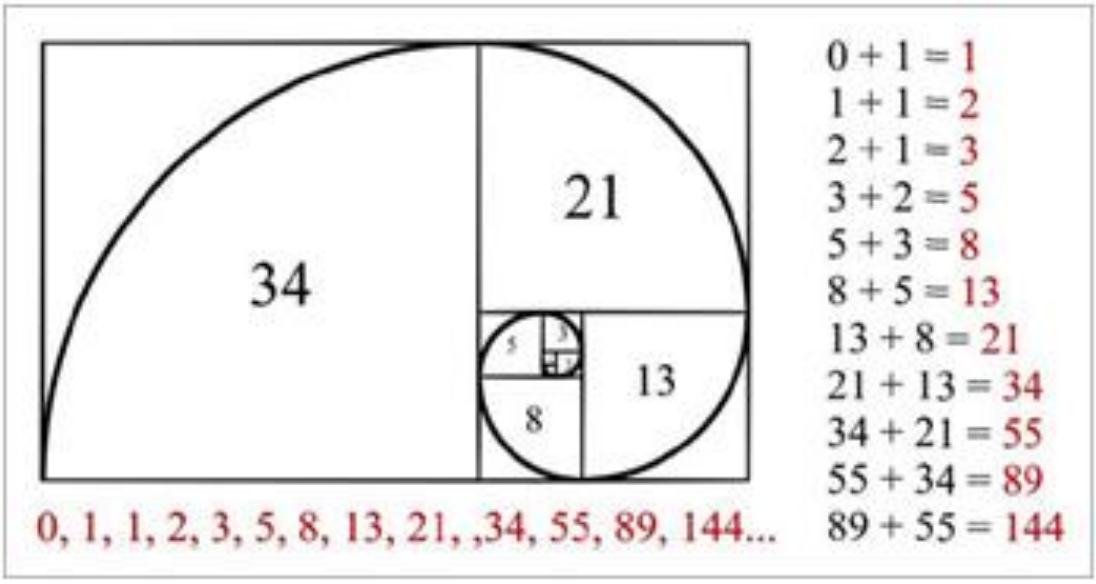
حل: برای پیدا کردن بیشترین(max) و کمترین(min) نمره، می توان فرض نمود که اولین نمره هم بیشترین است و هم کمترین. سپس بعد از خواندن نمرات بعدی، اگر از max بیشتر بود، آنرا با نمره جدید جایگزین می کنیم. و اگر از min کمتر بود، نمره جدید min خواهد شد. برای محاسبه میانگین هم کافی است نمرات را جمع نموده و بر تعداد تقسیم کنیم.

```
enter Student score[1]: 14
enter Student score[2]: 11.5
enter Student score[3]: 15
enter Student score[4]: 16.25
enter Student score[5]: 17
enter Student score[6]: 13
enter Student score[7]: 18
enter Student score[8]: 19.5
enter Student score[9]: 17
enter Student score[10]: 17.5
enter Student score[11]: 14
enter Student score[12]: 16
enter Student score[13]: 18
enter Student score[14]: 15
enter Student score[15]: 12
sum = 233.75
mean = 15.58333
max = 19.5
min = 11.5
max-min = 8
```

```
using System;
namespace StudentScores
{
    class Program
    {
        static void Main(string[] args)
        {
            string input; // ورودی
            float score,max,min, sum = 0, mean;// نمره ، بیشترین، کمترین، مجموع و میانگین
            int number = 1; // تعداد نمرات
            Console.Write("enter Student score[" + number + "]: ");
            input = Console.ReadLine();
            score = float.Parse(input); // خواندن اولین نمره
            max = score; min = score; // فرض: اولین نمره بیشترین و کمترین است
            sum += score; // جمع کردن نمره
            while (number < 15)
            {
                number ++; // شمارش نمره
                Console.Write("enter Student score[" + number + "]: ");
                input = Console.ReadLine();
                score = float.Parse(input); // خواندن نمره بعدی
                sum += score; // جمع کردن نمره
                if (score > max) max = score;  // به روزرسانی بزرگترین
                if (score < min) min = score; // به روزرسانی کوچکترین
            }
            mean = sum / number; // محاسبه میانگین
            Console.WriteLine("sum = "+ sum); // نمایش مجموع نمرات
            Console.WriteLine("mean = " + mean); // نمایش میانگین نمرات
            Console.WriteLine("max = " + max); // نمایش بیشترین نمره
            Console.WriteLine("min = " + min); // نمایش کمترین نمره
            Console.WriteLine("max-min = " + (max-min)); // نمایش اختلاف بیشترین و کمترین
        }
    }
}
```

مثال: سری فیبوناتچی: یک سری عدد پشت سرهم که هر عدد برابر مجموع دو عدد قبلی است:

1 , 1 , 2 , 3, 5, 8 , 13 , 21 , 34, 55 , ...



برنامه زیر این سری را تا کمتر از عدد ۱۰۰ نمایش می دهد:

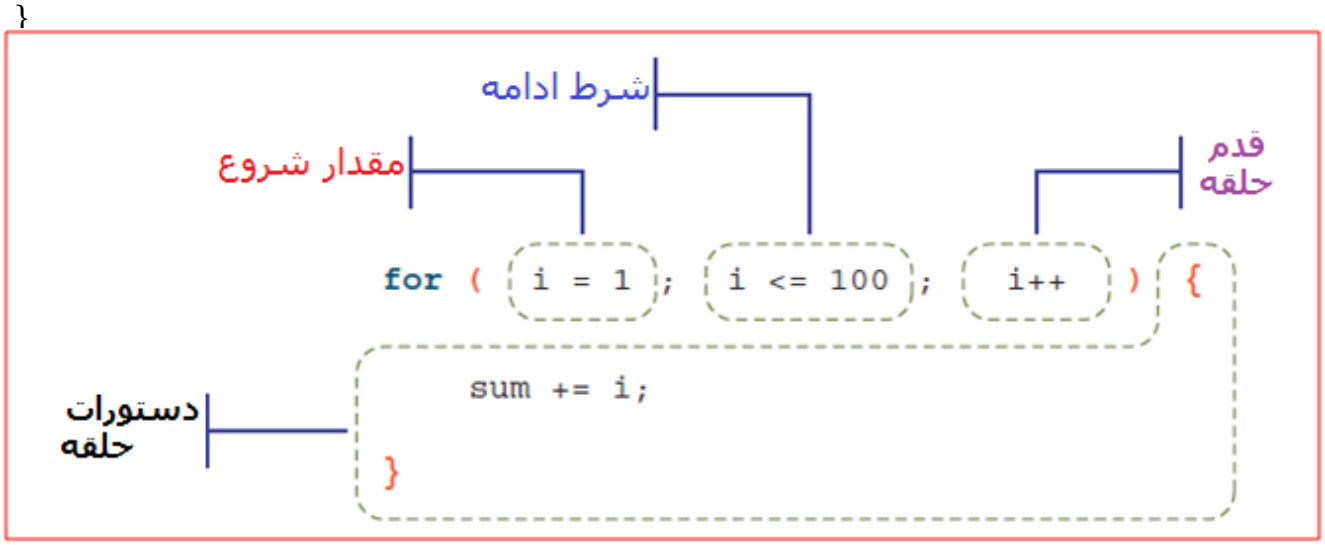
```
using System;
namespace fibonacci
{
    class Program
    {
        static void Main(string[] args)
        {
                /* ۱۶۰ سری فیبوناتچی: خودآزمایی ۵ صفحه
                      یک سری عدد پشت سرهم که هر عدد برابر مجموع دو عدد قبلی است
                      1 , 1 , 2 , 3,  5, 8 , 13 , 21 ,  34, 55 , ...
                 */
                int a = 1, b = 1, c = 0;
                while (a < 100)  // ۱۰۰ تا عدد
                {
                        Console.WriteLine(a); // نمایش عدد سری
                        c = a + b;  // جمع دو عدد قبلی
                        a = b;  // جایگزینی عدد اول با دوم
                        b = c;  // جایگزینی عدد دوم با عدد جدید
                }
        }
    }
}
```

حلقه تکرار for

دستورات حلقه در یک برنامه کامپیوتری، برنامه را مجبور می کنند تا برای تعداد دفعاتی که ما برای آن تعریف می کنیم دستورات خاصی از برنامه را اجرا کنند. حلقه for برای حالتی کاربرد دارد که تعداد تکرار از قبل مشخص باشد.

ساختار این حلقه بصورت زیر است:

```
for ( مقدار شروع ; شرط ادامه ; اندازه قدمهای حلقه) {
        دستوراتی که باید چند بار اجرا شوند
}
```



هر حلقه for دارای یک شمارنده است که تعداد تکرار را می شمارد. این شمارنده متغیری مثلاً از نوع عدد صحیح است.

داخل پرانتز مقابل for باید سه قسمت را بگنجانیم که باعلامت ; از هم جدا می شوند:

- مقدار شروع : باید نقطه آغازین حلقه باشد، یعنی شمارنده حلقه از چند شروع کند.
- شرط ادامه : یک شرط است(اغلب مقایسه شمارنده با یک مقدار نهایی) که تا زمانیکه درست است، حلقه ادامه خواهد یافت و به محض اینکه نادرست شود برنامه از حلقه خارج خواهد شد.
- اندازه قدمهای حلقه :باید مشخص کنیم که مقدار شمارنده نقطه آغازین در هر بار تکرار حلقه چقدر تغییر کند.

مثال ۱: برنامه ای طراحی کنید که مجموع اعداد یک تا ۱۰۰ را بدست آورد.

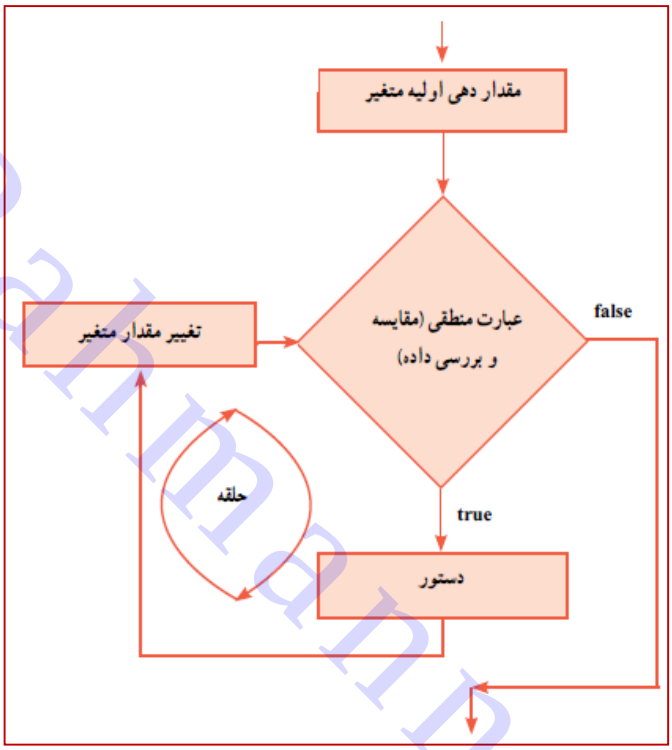
حل: باید حلقه ای طراحی کنیم که از یک تا ۱۰۰ حرکت کند پس سه قسمت حلقه باید بصورت زیر باشد:

- مقدار شروع: **i=1**
- شرط ادامه: **i<=100** ، یعنی تا زمانیکه شمارنده به ۱۰۰ نرسیده است
- اندازه قدم حلقه: **i++** ، یعنی هربار یک واحد به شمارنده اضافه شود.

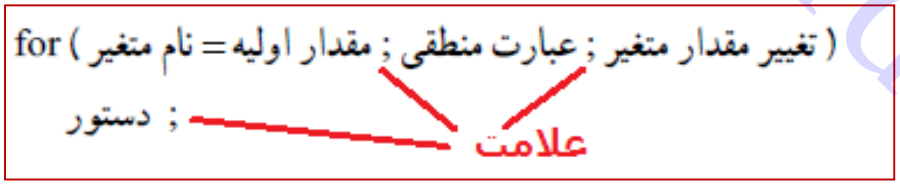
در قسمت دستورات حلقه هم متغیر sum که مجموع را نگه می دارد، در هر بار گردش و تکرار حلقه مقدار i را با خود جمع می کند: **sum += i**

```
using System;
namespace Sum100
{
    class Program
    {
        static void Main(string[] args)
        {
            int i;        // شمارنده حلقه
            int sum = 0;  // متغیر مجموع
            for (i = 1; i <= 100; i++) // حلقه تکرار
            {
                sum += i;  // دستور حلقه: جمع اعداد ۱ تا ۱۰۰
            }
            Console.WriteLine("sum = " + sum); // نمایش نتیجه
        }
    }
}
```

نحوه اجرای حلقه for: نخست مقدار شروع تعین می شود، سپس شرط بررسی می گردد، تا زمانیکه شرط درست است، دستورات داخل حلقه اجرا شده و مقدار شمارنده در قسمت اندازه قدم حلقه، تغییر می کند. اگر مقدار شمارنده در شرط صدق نکند، کار حلقه تمام می شود.



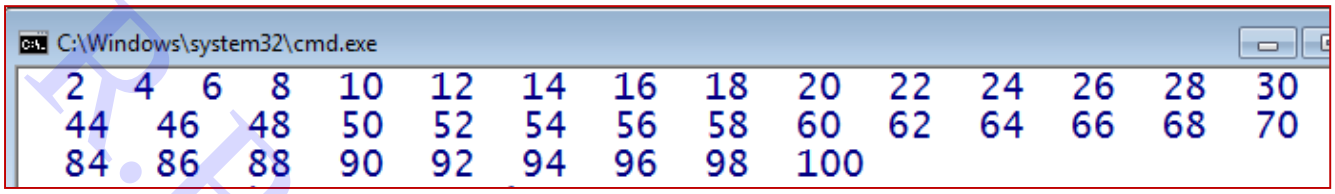
نکته: علامت ; را بین سه قسمت داخل پرانتز دستور for فراموش نکنید و در ضمن نباید بعد از پرانتز آخر، علامت ; را به منزله پایان دستور نوشت، بلکه این علامت باید در آخر دستورات حلقه نوشته شود.



مثال۲: اعداد زوج کمتر از ۱۰۰ را نمایش دهید

```
using System;
namespace EvenNumbers
{
    class Program  // نمایش اعداد زوج کمتر از ۱۰۰
    {
        static void Main(string[] args)
        {
            int i; // شمارنده حلقه
            for (i = 2; i <= 100; i += 2) // شروع: ۲؛ تا ۱۰۰ ادامه بده؛ هربار ۲ واحد افزایش
            {
                Console.Write("  " + i); // نمایش هر عدد
            }
            Console.WriteLine(); // یک خط خالی
        }
    }
}
```

بخشی از خروجی برنامه:



تمرین ۱: اعداد فرد کمتر از ۱۰۰

تمرین ۲: نمایش اعداد مضرب ۵، بین ۱۰۰ تا ۱۰۰۰

تمرین ۳: نمایش اعداد زوج بین ۳ تا ۲۱ بصورت نزولی مانند زیر:

20 , 18, 16, 14 , … , 4,2

حلقه کاهشی: اگر مقدار شروع از مقدار پایان کمتر باشد، اندازه قدمهای حلقه کاهشی(کم کردن) خواهد بود. در این صورت حلقه کاهشی داریم. مثلاً اگر بخواهیم اعداد ۱ تا ۱۰۰ را بصورت نزولی (از بزرگ به کوچک) نمایش دهیم باید بنویسیم:

دقت کنید که شرط ادامه حلقه این است که به صفر نرسد(i>0) و قدم حلقه بصورت i-- می باشد.

```
for(i  = 100 ; i > 0 ; i--){
        Console.Write(“  ” + i);
}
```

نکته : در حلقه کاهشی شرط برعکس می شود:

for(i = 100 ; i > 0 ; i--) → حلقه کاهشی

for(i = 1 ; i <=100 ; i++) → حلقه افزایشی

دستور break در حلقه for:

هرگاه بخواهیم در اثر شرطی خاص اجرای حلقه متوقف شود، از دستور **break;** استفاده می شود. به مثال زیر برای کنترل نمره دقت کنید که تعدادی نمره را خوانده و بیشترین نمره را پیدا می کند.

اگر نمره اشتباه (کمتر از صفر یا بیشتر از ۲۰) وارد گردد، کار حلقه متوقف شود

```
using System;
namespace MaxNumberBreak
{
    class Program
    {
        static void Main(string[] args)
        {
            string input; // ورودی
            float score, max;// نمره ، بیشترین
            int number = 1; // تعداد نمرات
            Console.Write("enter Student score[" + number + "]: ");
            input = Console.ReadLine();
            score = float.Parse(input); // خواندن اولین نمره
            max = score; // فرض: اولین نمره بیشترین است
            for (number = 2; number <= 10; number++)
            {
                Console.Write("enter Student score[" + number + "]: ");
                input = Console.ReadLine();
                score = float.Parse(input); // خواندن نمره بعدی
                if ((score < 0) || (score > 20)) // اگر نمره بین صفر تا ۲۰ نباشد
                {
                    Console.Write("Error in input!!"); // پیام خطای ورودی
                    break;    // خروج از حلقه
                }
                if (score > max) max = score;  // به روزرسانی بزرگترین
            }
            Console.WriteLine("max = " + max); // نمایش بیشترین نمره
        }
    }
}
```

مثلاً در خروجی زیر بعد از وارد کردن ۳ نمره، مقدار نمره ورودی ۲۴ باعث خروج از حلقه شده است (قرار بود ۱۰ نمره وارد شود) و فقط بیشترین مقدار(max) این سه نمره محاسبه شده است.

```
C:\Windows\system32\cmd.exe
enter Student score[1]: 15
enter Student score[2]: 17
enter Student score[3]: 12
enter Student score[4]: 24
Error in input!!
max = 17
```

نکته : نوشتن همه قسمتهای حلقه for اجباری نیست، اما اگر قسمتی خالی باشد، باید علامت ; فراموش نشود. به مثالهای زیر دقت کنید:

int i = 1; for(; i <=100 ; i++)	در اینجا، قسمت مقدار گذاری اولیه، بیرون حلقه انجام شده و آن قسمت از حلقه خالی است
for(; ;) Console.Write("*");	در اینجا، تمام قسمتهای حلقه، خالی است و دستور چاپ(نمایش علامت *) بی نهایت بار اجرا می گردد.

حلقه های تودرتو

اگر داخل یک حلقه، حلقه دیگری بکار ببریم، حلقه تودرتو داریم. به ازای هر گردش حلقه بیرونی، حلقه داخلی دوباره از اول تا آخر اجرا می شود. مثلاً فرض کنید در یک مسابقه دو میدانی، باید ۱۰ دور زمین را با دو طی کنیم و بعد از هر بار دور میدان، ۵ بار طناب بزنیم، در این صورت چند بار در مجموع طناب می زنیم؟

```
for(i = 1 ; i<=10 ; i++){   به تعداد ۱۰ مرتبه
دویدن دور میدان
        for(j = 1 ; j<=5 ; j++){   به تعداد ۵ بار
        طناب زدن
        }
}
```

مثال: به کمک حلقه تودرتو جدول ضرب را نمایش دهید. فاصله اعداد در خروجی را به الگوی نمایش رشته ها، ۵ فاصله درنظر بگیرید.

```
namespace Jadvalzarb
{
    class Program
    {
        const int SIZE = 10; // تعداد سطر و ستون بصورت ثابت
        static void Main(string[] args)
        {
            for (int i = 1; i <= SIZE; i++) // تعداد سطر
            {
                for (int j= 1; j <= SIZE; j++)// تعداد ستون
                {
                    Console.Write("{0,5}", i * j);//نمایش حاصل ضرب با ۵ فاصله
                }
                Console.WriteLine();// بعد از هر سطر انتقال به خط بعد
            }
        }
    }
}
```

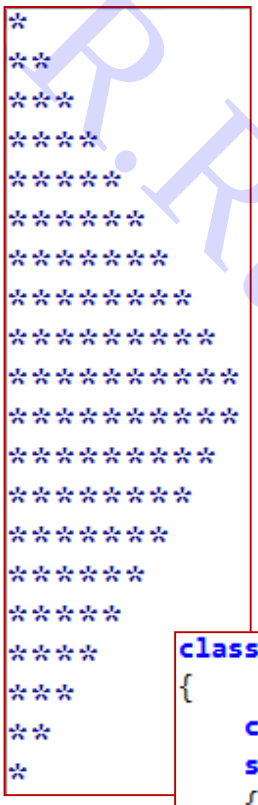
```
 1   2   3   4   5   6   7   8   9  10
 2   4   6   8  10  12  14  16  18  20
 3   6   9  12  15  18  21  24  27  30
 4   8  12  16  20  24  28  32  36  40
 5  10  15  20  25  30  35  40  45  50
 6  12  18  24  30  36  42  48  54  60
 7  14  21  28  35  42  49  56  63  70
 8  16  24  32  40  48  56  64  72  80
 9  18  27  36  45  54  63  72  81  90
10  20  30  40  50  60  70  80  90 100
```

مثال: برنامه ای بنویسید که با استفاده از حلقه های تو در تو، مجموع اعداد طبیعی را طبق شکل روبرو محاسبه و نمایش دهد

```
1 = 1
1 + 2 = 3
1 + 2 + 3 = 6
1 + 2 + 3 + 4 = 10
1 + 2 + 3 + 4 + 5 = 15
1 + 2 + 3 + 4 + 5 + 6 = 21
1 + 2 + 3 + 4 + 5 + 6 + 7 = 28
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 = 36
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 45
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55
```

```
class Program
{
    const int SIZE = 10; // تعداد اعداد
    static void Main(string[] args)
    {
        int sum = 0;
        for (int i = 1; i <= SIZE; i++) // گردش به تعداد اعداد
        {
            sum = 1; // مجموع
            Console.Write(1); // چاپ اولین عدد
            for (int j = 2; j <= i; j++)// گردش تا عدد حلقه بیرونی
            {
                Console.Write(" + {0}",j); // نمایش سری
                sum = sum + j; // مجموع
            }
            Console.Write(" = {0}",sum); // نمایش مجموع سری
            Console.WriteLine();// بعد از هر سطر انتقال به خط بعد
        }
    }
}
```

مثال: برنامه ای طراحی کنید که خروجی زیر را تولید کند.

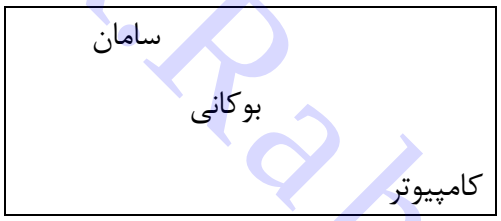


```
*
**
***
****
*****
******
*******
********
*********
**********
***********
***********
**********
*********
********
*******
******
****
***
**
*
```

```
class Program
{
    const int SIZE = 10; // تعداد
    static void Main(string[] args)
    {
        int i, j;
        for (i = 1; i <= SIZE; i++) // تعداد سطر
        {
            for (j = 1; j <= i; j++)// تعداد در هر سطر
            {
                Console.Write("*"); //
            }
            Console.WriteLine();// بعد از هر سطر انتقال به خط بعد

        }
        for (i = SIZE; i >= 1; i--) // // تعداد سطر بصورت نزولی
        {
            for (j = 1; j <= i; j++)// تعداد در هر سطر
            {
                Console.Write("*"); //
            }
            Console.WriteLine();// بعد از هر سطر انتقال به خط بعد
        }
    }
}
```

تمرینات و پروژه برنامه نویسی

۱) در زمینه سیاه و به رنگ زرد، نام و نام خانوادگی، رشته تحصیلی خود را با فاصله ۳ tab از هم بصورت قطری نمایش دهید.



۲) صداهایی با فرکانس ۵۰۰، ۱۰۰۰، ۱۵۰۰ و ۲۰۰۰، به ترتیب با فواصل ۱، ۲، ۳ و ۴ ثانیه پخش کنید.

۳) سری فیبوناتچی را تا ۵۰ عدد نمایش دهید

۴) نام تیم، تعداد بازی، تعداد برد، تعداد مساوی، تعداد باخت را خوانده و با محاسبه امتیاز، جدول زیر را تشکیل دهید(امتیاز = برد*۳ + مساوی *۱)

اگر جمع برد، مساوی و باخت با تعداد بازی برابر نباشد، خطا اعلام کند

نام تیم	بازی	برد	مساوی	باخت	امتیاز
=====	=====	=====	=====	=====	=====
بارسلونا	۳۵	۲۹	۲	۴	۸۹

۵) فرض کنید در کنکور شرکت کرده اید، تعداد کل سوالات، تعداد پاسخ درست، تعداد پاسخ غلط، تعداد بدون پاسخ را خوانده و با محاسبه درصد، جدول زیر را تشکیل دهید (درصد =100*[(تعداد کل*۳)/ (غلط*۱ - درست *۳)])

اگر جمع درست، غلط و سفید با تعداد سوالات برابر نباشد، خطا اعلام کند

تعداد سوالات	درست	غلط	سفید	درصد
=====	=====	=====	=====	=====
۱۰۰	۷۰	۲۰	۱۰	٪۶۳

۶) تاریخ را خوانده و محاسبه کنید تا آن تاریخ چند روز از سال گذشته و چند روز مانده است. (دقت کنید از ماه ۱ تا ۶ تعداد روز هرماه ۳۱ و برای بقیه ۳۰ روز لحاظ شود. اسفند هم ۲۹ روز است).

مثلاً اگر امروز ۲۵ مرداد باشد(۱۳۹۴/۵/۲۵)، چهار ماه کامل ۳۱ روزه تا آخر تیر گذشته و ۲۵ روز هم از ماه پنجم سپری شده است. پس:

تعداد روزهای گذشته = ۴*۳۱ +۲۵=۱۴۹

تعداد روزهای مانده سال = ۳۶۵ - ۱۴۹ = ۲۶۶

فصل ششم- تبدیل نوع داده

متغیر در برنامه برای ذخیره داده ها بکار می رود و هر متغیر دارای ظرفیت و گنجایش مشخص است یعنی مقدار از نوع خاص و تا محدوده معینی را می تواند نگه دارد. مثلاً متغیر از نوع byte قادر به ذخیره سازی اعداد صحیح ۰ تا ۲۵۵ می باشد.

مثال: مقدار سن خود را در متغیری ذخیره کرده و سپس نمایش دهید.

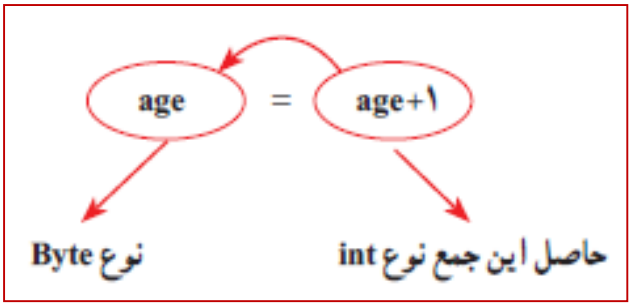
```
class Program
{
    static void Main(string[] args)
    {
        byte age = 16;
        Console.WriteLine("My age is " + age);
    }
}
```

حال اگر بخواهیم به متغیر خود یک واحد اضافه کنیم، برنامه بصورت زیر خواهد بود:

```
class Program
{
    static void Main(string[] args)
    {
        byte age = 16;
        Console.WriteLine("My age is " + age);
        age = age + 1;
        Console.WriteLine ("Now my age is " + age);
    }
}
```

با اجرای برنامه در خط چهارم خطا اعلام می شود! دلیل خطا چیست؟

جواب: در عبارت خط چهارم که به متغیر یک واحد اضافه می کند، عدد ۱ بصورت پیش فرض از نوع int در نظرگرفته می شود و حاصل جمع age+1 بصورت int تبدیل می گردد و در متغیر age که از نوع byte است، قابل ذخیره نیست.



راه حل رفع مشکل:

از مترجم برنامه بخواهیم که حاصل جمع را بصورت byte در نظر بگیرد. برای اینکار تبدیل نوع را بصورت زیر انجام خواهیم داد:

عبارت محاسباتی مبدأ (نوع داده مقصد) = متغیر از نوع مقصد

age = (byte) (age +1)

برنامه اصلاح شده بصورت زیر است:

```
class Program
{
    static void Main(string[] args)
    {
        byte age = 16;
        Console.WriteLine("My age is " + age);
        age = (byte)(age + 1);
        Console.WriteLine ("Now my age is " + age);
    }
}
```

مثال: برنامه ای بنویسید که به کمک تبدیل نوع، عدد و کارکتر را به هم تبدیل کند.

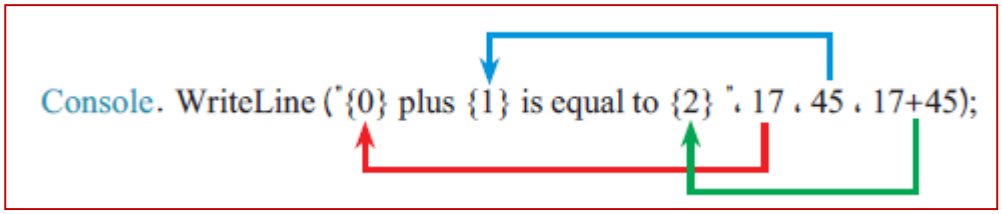
```
static void Main(string[] args)
{
    int aNumber = 67; // تعریف متغیر عددی
    Console. Write ( (char) aNumber); // تبدیل عدد به کارکتر و نمایش آن یعنی حرف C
    char ch = '#' ;// تعریف متغیر کارکتری
    Console. WriteLine (ch); // نمایش متغیر کارکتری
    Console. WriteLine ("Code of C letter : " + (int) 'C'); //تبدیل کارکتر 'C' به عدد
    Console. WriteLine ("Code of # : " + (int) ch); //تبدیل کارکتر ذخیره شده در ch به عدد
}
```

```
C:\Windows\system32\cmd.exe
C#
Code of C letter : 67
Code of # : 35
Press any key to continue . . .
```

خروجی نمایش کارکتری عدد ۶۷(معادل حرف c) و به دنبال آن کارکتر # و کد عددی حرف c و کارکتر : #

الگوی جایگذاری در رشته ها

به جای استفاده از علامت + در دستور write می توان به ازای هر رشته که در الحاق شرکت می کند، یک شماره با شروع از صفر در داخل { } در قالب زیر استفاده کنیم:



این شماره ها محل قرار گیری داده ها در رشته خروجی را تعیین می کنند.

خروجی مثال بالا بصورت زیر خواهد بود:

17 plus 45 is equal to 62

پس قالب کلی :

Console.write("{شماره}.... ", لیست متغیرها یا مقادیر);

علاوه بر شماره محل قرارگیری در داخل {} می توان سه مورد زیر را تعیین کرد:

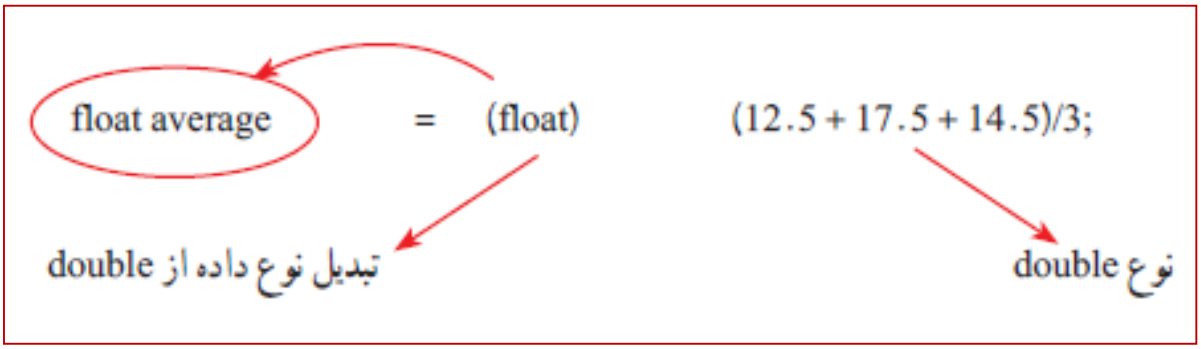
{الگوی نمایش : عدد تراز، شماره}

- **شماره**: شماره محل داده در رشته خروجی
- **عدد تراز**: تعداد فضای خالی برای نمایش داده، اگر عدد منفی باشد، فضای اختصاصی چپ چین می شود و اگر مثبت باشد، راست چین می گردد.
- **الگوی نمایش**: کارکتر نحوه نمایش داده مثلاً حرف F برای نمایش اعداد اعشاری با نقطه ثابت بکار می رود.

مثال : برنامه محاسبه میانگین سه نمره و نمایش آن:

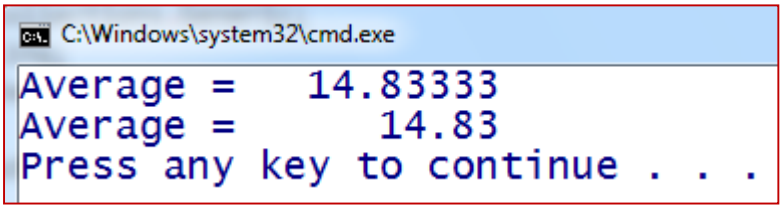
```
class Program
{
    static void Main(string[] args)
    {
        float average = (float)(12.5 + 17.5 + 14.5) / 3; // میانگین سه نمره
        Console.WriteLine("Average = {0,10}", average);// نمایش میانگین با ۱۰ فضا
        Console.WriteLine("Average = {0,10:F}", average);// نمایش میانگین با ۱۰ فضا و بصورت اعشار
    }
}
```

دقت کنید چون حاصل جمع نمرات بصورت double فرض می شود، نتیجه توسط عبارت (float) به عدد اعشار معمولی تبدیل شده است.



همچنین در الگوی خروجی از حرف F برای تبدیل عدد نتیجه به اعشار معمولی استفاده شده است.

خروجی برنامه با ۱۰ فضا برای نمایش عدد و بصورت راست چین:



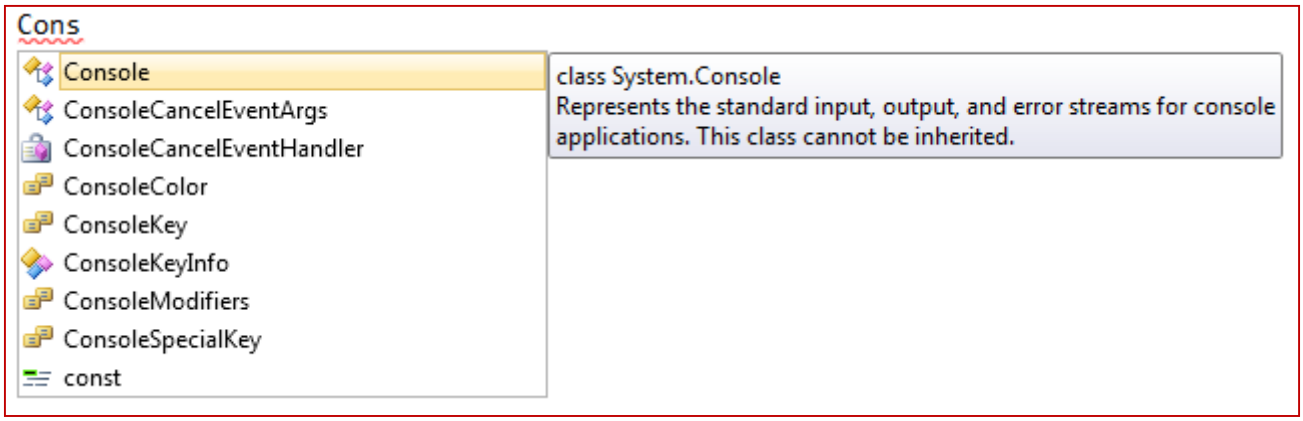
مثال: به کمک تبدیل نوع و الگوی نمایش رشته ها، کد کارکترهای اسکی را نمایش دهید.

```
code ==> Char   code ==> Char   code ==> Char   code ==> Char   code ==> Char   code ==>
=============   =============   =============   =============   =============   ========
1    ==> ☺      23   ==> ↨      46   ==> .      69   ==> E      92   ==> \      115  ==>
2    ==> ☻      24   ==> ↑      47   ==> /      70   ==> F      93   ==> ]      116  ==>
3    ==> ♥      25   ==> ↓      48   ==> 0      71   ==> G      94   ==> ^      117  ==>
4    ==> ♦      26   ==> →      49   ==> 1      72   ==> H      95   ==> _      118  ==>
5    ==> ♣      27   ==> ←      50   ==> 2      73   ==> I      96   ==> `      119  ==>
6    ==> ♠      28   ==> ∟      51   ==> 3      74   ==> J      97   ==> a      120  ==>
7    ==>        29   ==> ↔      52   ==> 4      75   ==> K      98   ==> b      121  ==>
8    ==>        30   ==> ▲      53   ==> 5      76   ==> L      99   ==> c      122  ==>
9    ==>        31   ==> ▼      54   ==> 6      77   ==> M      100  ==> d      123  ==>
10   ==>        32   ==>        55   ==> 7      78   ==> N      101  ==> e      124  ==>
                33   ==> !      56   ==> 8      79   ==> O      102  ==> f      125  ==>
11   ==> ♂      34   ==> "      57   ==> 9      80   ==> P      103  ==> g      126  ==>
12   ==> ♀      35   ==> #      58   ==> :      81   ==> Q      104  ==> h      127  ==>
13   ==>        36   ==> $      59   ==> ;      82   ==> R      105  ==> i
14   ==> ♫      37   ==> %      60   ==> <      83   ==> S      106  ==> j
15   ==> ☼      38   ==> &      61   ==> =      84   ==> T      107  ==> k
16   ==> ►      39   ==> '      62   ==> >      85   ==> U      108  ==> l
17   ==> ◄      40   ==> (      63   ==> ?      86   ==> V      109  ==> m
18   ==> ↕      41   ==> )      64   ==> @      87   ==> W      110  ==> n
19   ==> ‼      42   ==> *      65   ==> A      88   ==> X      111  ==> o
20   ==> ¶      43   ==> +      66   ==> B      89   ==> Y      112  ==> p
21   ==> §      44   ==> ,      67   ==> C      90   ==> Z      113  ==> q
22   ==> ▬      45   ==> -      68   ==> D      91   ==> [      114  ==> r
```

```
static void Main(string[] args)
{
    int i;
    Console.WriteLine(" code ==> Char");//عنوان خط اول
    Console.WriteLine(" =============");// عنوان خط دوم
    for (i = 1; i < 128; i++) // حلقه
        Console.WriteLine(" {0,-4} ==> {1}", i, (char)i);// چاپ کد و کارکتر
}
```

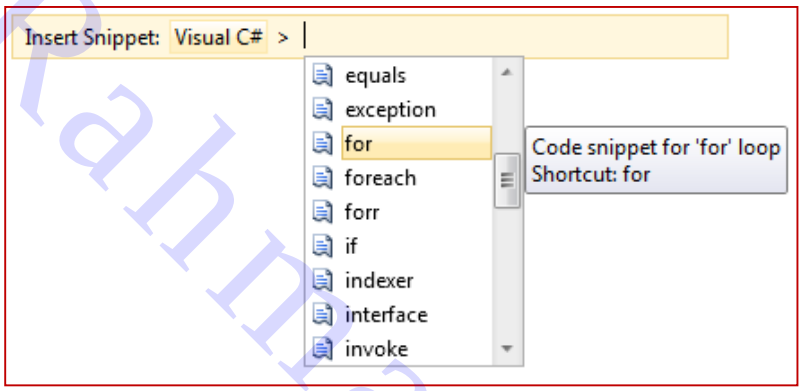
نکات سرعت بخشیدن به تایپ برنامه

(۱) در محیط برنامه نویسی VS، در هنگام نوشتن برنامه، یکی از امکانات ارزشمند آن IntelliSense به کمک می آید. این امکان همراه با تایپ بخشی از دستور، لیستی را به نمایش در می آورد که کمک می کند با زدن دکمه Tab یا Space دستور را کامل کنیم.مثلاً برای تایپ Console با نوشتن قسمتی از آن لیستی مثل زیر باز می شود و کلمات



پیشنهادی تکمیلی را نمایش می دهد. در سمت راست توضیحی مختصر از هر پیشنهاد را نیز ارائه می نماید.برای ظاهر کردن امکان IntelliSense در هنگام تایپ برنامه کلید ترکیبی CTRL+SPACE را فشار دهید.

(۲) در محیط برنامه نویسی کلیک راست نمایید و گزینه insert Snipet را انتخاب کنید یا کلید ترکیبی CTRL + K, X را فشار دهید. سپس گزینه Visual C# و در نهایت دستور مورد نظر مثلاً for را کلیک کنید



بصورت خودکار قالب دستور انتخاب شده(مثلاً for یا if) تایپ می گردد:

```
for (int i = 0; i < length; i++)
{
    |
}
```

```
if (true)
{

}
```

استفاده از مقدار ثابت در برنامه

برای تعریف شناسه یا نام ثابت از کلمه کلیدی const استفاده می شود. معمولاً نامی که برای اعداد ثابت تعریف می شود با حروف بزرگ نوشته می شود تا در برنامه مشخص باشد که این شناسه،یک مقدار ثابت است. نحوهٔ تعریف ثابت ها، مانند تعریف متغیرها می باشد با این تفاوت که در ابتدای تعریف آنها کلمه const قرار دارد.

```
const نوع داده نام ثابت = مقدار;
const int SIZE = 15;
```

☜ نکته: مقدار ثابت در طول برنامه از نقطه تعریف به بعد قابل استفاده بوده ولی مقدار آن دیگر قابل تغییر نیست.

مثال: برنامه ای بنویسید که نمره تعدادی دانش آموز را خوانده و مجموع و میانگین آنها را نمایش دهد. تعداد دانش آموزان توسط ثابت تعریف شود.

نمونه خروجی برنامه:

```
Enter a mark:15
Enter a mark:16
Enter a mark:17
Enter a mark:14.5
Enter a mark:18.5
Total :81
Average :16.2
```

```
class Program
{
    const int SIZE = 5; // تعریف ثابت
    static void Main ()
    {
        float number, total =0, average;
        string input;
        for (int i = 0; i < SIZE; i++) // گردش حقله به تعداد ثابت داده شده
        {
            Console. Write (" Enter a mark:" ) ;//پیام
            input = Console. ReadLine ( ); // خواندن نمره
            number = float. Parse (input); //float تبدیل به
            total = total + number; // جمع نمرات
        }
        average = total / SIZE ; // میانگین به کمک تعداد ثابت
        Console. WriteLine ("Total :" + total); // نتایج
        Console. WriteLine ("Average :" + average);
    }
}
```

عملکرد کارکتر \ (back slash):

این کارکتر اگر به همراه علائم جدول زیر در رشته خروجی در دستور Console.Write() استفاده گردد، کاربرد و معنی خاصی دارد.

عملکرد	دنباله
ایجاد یک بوق هشدار[۲]	\a
حذف یک کاراکتر(Backspace)	\b
ایجاد یک خط خالی(New Line)	\n
ایجاد یک فاصله افقی زیاد tab	\t
ایجاد یک تک کوتیشن (')	\'
ایجاد یک دابل کوتیشن (")	\"
ایجاد یک Back slash (\)	\\

مثلاً برای نوشتن عبارت روبرو، باید دستور نمایش خروجی به بدین شکل باشد:

I like programing with 'C#'

```
Console.Write("I like \t programing with \' C# \' ");
```

فصل هفتم – آرایه ها

متغیر: مکانی نامدار از حافظه برای ذخیره موقت اطلاعات

آرایه: مجموعه ای از متغیرهای همنام و هم نوع که بصورت متوالی در حافظه ذخیره می شوند.

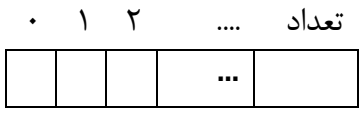
نحوه تعریف آرایه: در دو مرحله انجام می شود

- معرفی آرایه

```
نام متغیر آرایه  [ ] نوع داده;
```

- تعریف اندازه آرایه

```
نام متغیر آرایه= new نوع داده[اندازه آرایه];
```



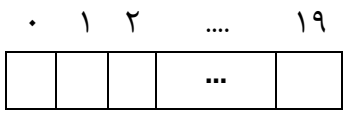
مثال: ذخیره نمرات(نوع داده اعشاری) درس برنامه سازی همه دانش آموزان(۲۰ نفر)

```
float[] mark;        ← مرحله اول
new float [20]       ← مرحله دوم
```

به جای دو دستور بالا می توانیم دستور زیر را جایگزین نماییم:

```
float [ ] mark = new float [20] ;
```

حال ۲۰ متغیر داریم که از شماره صفر تا ۱۹ پشت سرهم قرار گرفته اند:



پس شکل کلی تعریف آرایه بصورت زیر است:

```
نام آرایه [ ] نوع داده = new نوع داده[اندازه آرایه]
```

دسترسی به اعضای آرایه

به هریک از مکانهای آرایه عنصر آرایه می گویند. برای تفکیک و دسترسی به این مکانها، از یک عدد صحیح به نام اندیس استفاده می شود. اولین عنصر آرایه با اندیس صفر مشخص می شود و عنصر بعد از آن شماره یک و به ترتیب جلو می رود تا آخرین عنصر که یک واحد کمتر از تعداد عناصر آرایه است.

برای دسترسی به عناصر از نام آرایه و شماره محل آن استفاده می کنیم:

```
نام متغیر آرایه [شماره محل یا اندیس]
```

mark [15] → *نمره دانش آموز شماره ۱۵*

می توان تک تک آنها را با دستور انتساب مقدار داد (تغییر داد) یا از ورودی خواند یا چاپ کرد:

```
mark [15]=17.5 f ;              → تغییر نمره دانش آموز شماره ۱۵ به ۱۷/۵
mark [10]= float. Parse (input); → خواندن نمره دانش آموز شماره ۱۰
Console.Write ( mark [6] );      → چاپ نمره دانش آموز شماره ۶
```

نکته ۱: برای پردازش همه عناصر آرایه معمولاً از حلقه for استفاده می شود.

مثال ۱ : دستوراتی بنویسید که ۱۰ نمره را خوانده، در آرایه ذخیره نموده و سپس آنها را همراه با مجموع و میانگین چاپ کند.

```
const int SIZE = 10; // تعداد نمرات
static void Main(string[] args)
{
    string input; // ورودی
    float sum = 0, average; // مجموع و میانگین
    float[] list = new float[SIZE];// تعریف آرایه به اندازه معین
    for (int i = 0; i < SIZE; i++) // حلقه تکرار به اندازه داده شده
    {
        Console.Write("list[{0}]:",i+1);// نمایش پیام به همراه شماره اندیس
        input = Console.ReadLine(); // ورودی
        list[i] = float.Parse(input); // ذخیره ورودی در آرایه
        sum = sum + list[i]; // جمع عناصر آرایه
    }
    average = sum / SIZE; // محاسبه میانگین
    Console.WriteLine("Mark List :");
    for (int i = 0; i < SIZE; i++)
    {
        Console.Write("{0,6:f}",list[i]); // نمایش نمرات با فاصله ۶ و اعشاری
    }
    Console.WriteLine("\n\n sum = {0} , average = {1}",sum,average);// مجموع و میانگین
}
```

نمونه خروجی را در زیر مشاهده می کنید:

```
list[1]:18.5
list[2]:17
list[3]:14
list[4]:11
list[5]:12
list[6]:15
list[7]:16
list[8]:20
list[9]:17
list[10]:11
Mark List :
 18.50 17.00 14.00 11.00 12.00 15.00 16.00 20.00 17.00 11.00

 sum = 151.5 , average = 15.15
```

مثال: برنامه مثال نمرات را طوری تغییر دهید که درصد تعداد دانش آموزانی که نمره بیشتر از میانگین کسب کرده اند نشان دهد.

100(تعداد کل دانش آموزان/تعداد افرادی که نمره بالای میانگین دارند)=درصد بالاتر از میانگین*

پس باید نخست تعداد دانش آموزانی که نمره بالای میانگین دارند بشماریم:

```
int n = 0; // شمارش
float percent;// درصد بالای میانگین
for (int i = 0; i < SIZE; i++)
{
    if (list[i] >= average) n++;  // شمارش تعداد دانش آموزان با نمره بیش از میانگین
    Console.Write("{0,6:f}", list[i]); // نمایش نمرات با فاصله ۶ و اعشاری
}
percent =((float) n / SIZE)*100;// محاسبه درصد بصورت اعشاری
```

نمونه خروجی :

```
Mark List :
18.50 17.00 14.00 11.00 12.00 15.00 16.00 20.00 17.00 11.00
sum = 151.5 , average = 15.15
n =5 , percent = 50 %
```

```
const int SIZE = 10; // تعداد نمرات
static void Main(string[] args)
{
    string input; // ورودی
    float sum = 0, average; // مجموع و میانگین
    float[] list = new float[SIZE];// تعریف آرایه به اندازه معین
    for (int i = 0; i < SIZE; i++) // حلقه تکرار به اندازه داده شده
    {
        Console.Write("list[{0}]:", i + 1);// نمایش پیام به همراه شماره اندیس
        input = Console.ReadLine(); // ورودی
        list[i] = float.Parse(input); // ذخیره ورودی در آرایه
        sum = sum + list[i]; // جمع عناصر آرایه
    }
    average = sum / SIZE; // محاسبه میانگین
    Console.WriteLine("\n Mark List :");
    int n = 0; // شمارش
    float percent;// درصد بالای میانگین
    for (int i = 0; i < SIZE; i++)
    {
        if (list[i] >= average) n++;  // شمارش تعداد دانش آموزان با نمره بیش از میانگین
        Console.Write("{0,6:f}", list[i]); // نمایش نمرات با فاصله ۶ و اعشاری
    }
    percent =((float) n / SIZE)*100;// محاسبه درصد بصورت اعشاری
    Console.WriteLine("\n sum = {0} , average = {1}", sum, average);// مجموع و میانگین
    Console.WriteLine("n ={0} , percent = {1} % ", n,percent);// مجموع و میانگین
}
```

روش دیگر مقدار دهی عناصر آرایه در هنگام تعریف و ایجاد آرایه است که در انتهای تعریف آرایه در بین علامت های { } مقدار هر عنصر از آرایه را به ترتیب معین می کنیم:

نام آرایه [] نوع داده = new نوع داده [] {مقدار، مقدار، ...، مقدار} ;

همان طور که مشاهده می کنید عددی بین علامت های [ ] وجود ندارد، بنابراین اندازه آرایه را ذکر نمی کنیم بلکه اندازه چنین آرایه هایی با تعداد مقادیر نوشته شده بین علامت های { } تعیین می شود.
مثلاً برای ایجاد آرایه ای برای سکه های رایج بر حسب ریال، به صورت زیر تعریف می کنیم.

```
int [] coin = new int [] {500, 1000,  2000,  5000 };
```

دستور بالا آرایه ای به نام coin شامل چهار عنصر ایجاد می کند که هر عنصر آن مقدار یک سکه را مشخص می کند که از سکه 500 ریالی شروع و به سکه 5000 ریالی ختم می شود.

۰	۱	۲	۳
۵۰۰	۱۰۰۰	۲۰۰۰	۵۰۰۰

آرایه coin ←

همچنین با توجه به اینکه آرایه coin دارای مقادیر اولیه مشخص است می توان بدون استفاده از عملگر new آن را ایجاد کرد. بنابراین دستورات بالا را می توان با دستور زیر جایگزین نمود:

```
int [] coin = {500, 1000, 2000, 5000};
```

پس شکل کلی تعریف آرایه با مقادیر مشخص بصورت زیر است:

نام آرایه [] نوع داده = { مقدار، ...، مقدار، مقدار };

- بعد از ایجاد آرایه، نمی توانید اندازه آن را تغییر دهید یعنی نمی توانید عنصری به آن اضافه و یا کم کنید.
- در زبان C# محدوده اندیس آرایه کنترل می شود و نباید از عدد صفر کمتر و همچنین از اندازه آرایه بیشتر یا مساوی باشد.

int A [ ] = new int [10]; ← محدوده اندیس قابل قبول : از ۰ تا ۹

۰	۱	۲	....	۹
			...	

A[9] ← درست A[0] ← درست A[10] ← خط A[-2] ← خطا

مثال: برنامه ای به کمک آرایه ها طراحی کنید که با دریافت تاریخ روز(ماه و روز)، تعداد روزهای سپری شده و مانده سال را محاسبه و نمایش دهد.
حل: آرایه ای ۱۲ عنصری تعریف می کنیم و در آن تعداد روزهای هر ماه سال را به ترتیب ذخیره می کنیم(۶ ماه اول ۳۱، ۵ ماه بعدی ۳۰ و ماه آخر ۲۹ روز دارد).

```
// آرایه ذخیره روزهای هر ماه
int[] daysInMonth = { 31, 31, 31, 31, 31, 31, 30, 30, 30, 30, 30, 29 };
```

البته چون اندیس آرایه از صفر شروع می گردد، شماره ماه یک واحد از اندیس آرایه بیشتر است. سپس با حلقه for از شماره صفر تا شماره متناسب با ماه تاریخ ورودی، عناصر آرایه را جمع کرده و حاصل را با تعداد روز تاریخ جمع نموده تا تعداد روزهای سپری شده بدست آید. تعداد روزهای مانده هم برابر عدد ۳۶۵ منهای تعداد روزهای گذشته سال است.
مثلاً اگر امروز ۹۴/۷/۲۷ باشد، خروجی زیر را داریم:

```
Enter month: 7
Enter day: 27
 days passed = 213
 days reamined = 152
```

۶ ماه و ۲۷ روز از سال گذشته

```
int[] daysInMonth = { 31, 31, 31, 31, 31, 31, 30, 30, 30, 30, 30, 29 };
```

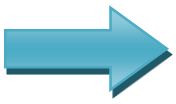
تاریخ: ۹۴/۷/۲۷ 6*31+27=213

```
static void Main(string[] args)
{
    const int DAYS_IN_YEAR = 365;    // تعداد روزهای سال
    int month, day;                  // تاریخ: ماه و روز
    int daysPassed=0, daysRemained=0;// تعداد روزهای سپری شده و مانده سال
    string input;                    // متغیر دریافت ورودی
    // آرایه ذخیره روزهای هر ماه
    int[] daysInMonth = { 31, 31, 31, 31, 31, 31, 30, 30, 30, 30, 30, 29 };
    Console.Write("Enter month: ");
    input = Console.ReadLine();
    month = int.Parse(input);          // خواندن ماه
    Console.Write("Enter day: ");
    input = Console.ReadLine();
    day = int.Parse(input);            // خواندن روز
    for (int i = 0; i < month - 1; i++) // پیمایش به تعداد ماه
    {
        daysPassed = daysPassed + daysInMonth[i]; //جمع روزهای ماههای سپری شده سال
    }
    daysPassed = daysPassed + day; //  جمع روز تاریخ با حاصلجمع
    daysRemained = DAYS_IN_YEAR - daysPassed;  // محاسبه تعداد روزهای مانده
    Console.WriteLine(" days passed = "+ daysPassed);
    Console.WriteLine(" days reamined = "+daysRemained);
}
```

حلقه foreach

در پیمایش آرایه ها و بررسی تمام عناصر آن مثل نمایش آرایه، به جای حلقه تکرار for از ساختار foreach می توان استفاده نمود:

```
float[] list = new float[10];
for (int i = 0; i < 10; i++)
{
  Console.Write(list[i]);
}
```



```
foreach (float m in list)
{
    Console.Write(m);
}
```

شکل کلی دستور foreach بصورت زیر است:

```
foreach (نام آرایه  in  نام متغیر  نوع داده)
        دستور ؛
```

در دستور مثال جدول فوق به جای متغیر حلقه i و پیمایش تک تک عناصر list[i]، متغیر m جایگزین عناصر شده و همه عناصر را به ترتیب انتخاب کرده و در خروجی نمایش می دهد.

مثال: نمرات را خوانده و تعداد مردودی را با دستور foreach بشمارید.

```
const int SIZE = 10;//تعداد
static void Main(string[] args)
{
    float[] list = new float[SIZE];
    string input;
    int fail = 0; // تعداد مردودی
    for (int i = 0; i < SIZE; i++) // خواندن نمرات
    {
        Console.Write("mark[{0}]:",i+1);
        input = Console.ReadLine();
        list[i] = float.Parse(input);
    }
    foreach (float m in list) // شمارش تعداد مردودی
    {
        if (m < 12) fail ++; // نمره کمتر از ۱۲
    }
    Console.WriteLine("fail = " + fail);

}
```

```
mark[1]:15
mark[2]:17
mark[3]:15.5
mark[4]:14
mark[5]:11
mark[6]:16
mark[7]:11.5
mark[8]:18
mark[9]:10
mark[10]:19
fail = 3
```

ویژگی Length: توسط این ویژگی می توان اندازه آرایه(تعداد خانه ها) را بدست آورد.

Length نام آرایه

float [ ] mark = new float [20] ; → مثال

mark.Length → اندازه آرایه که عدد ۲۰ است

مثال: فرض کنید اندازه آرایه mark را نمی دانیم. چگونه همه محتویات آنرا نمایش دهیم؟ جواب: به کمک ویژگی Length:

```
for ( int  i =0 ; i < mark.Length; i++)
        Console.Write(mark[i]);
```

فرض کنید می خواهیم نمرات دانش آموزان را بخوانیم و میانگین را حساب نماییم. همچنین می خواهیم برای هر کلاس با هر تعداد دانش آموز اینکار را انجام دهیم.

ولی تعداد آنها را از اول نمی دانیم که آرایه ای به اندازه مورد نظر تعریف کنیم. راه چاره چیست؟

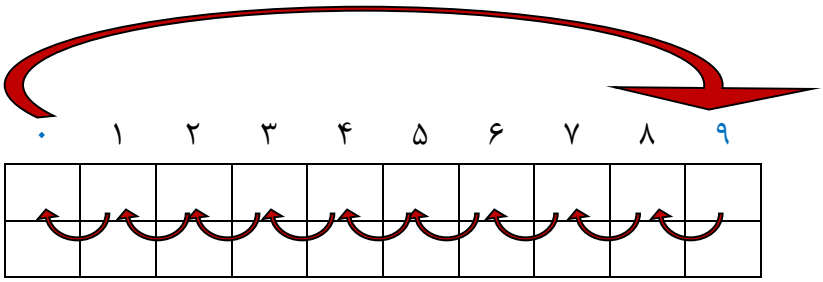
یک روش این است که آرایه ای با تعداد زیاد تعریف کنیم که به ازای هر تعداد کار کند. اما این روش مناسب نیست چون بخش زیادی از آرایه بلا استفاده می ماند و ویژگی Length هم بکار نمی آید زیرا این ویژگی اندازه کل را نشان می دهد.

روش مناسب این است که نخست تعداد دانش آموزان را خوانده و در متغیری ذخیره کنیم و سپس آرایه ای به اندازه آن تعریف کنیم.

```
class Program
{
    static void Main(string[] args)
    {
        string input; // ورودی
        float sum = 0, average; // مجموع و میانگین
        int arraySize;  // تعداد دانش آموزان
        Console.Write("enter Array size: ");
        input = Console.ReadLine();
        arraySize = int.Parse(input); // تعداد دانش آموزان
        float[] list = new float[arraySize];// تعریف آرایه به اندازه تعداد دانش آموزان
        for (int i = 0; i < arraySize; i++) // حلقه تکرار به اندازه داده شده
        {
            Console.Write("list[{0}]:", i + 1);// نمایش پیام به همراه شماره اندیس
            input = Console.ReadLine(); // ورودی
            list[i] = float.Parse(input); // ذخیره ورودی در آرایه
            sum = sum + list[i]; // جمع عناصر آرایه
        }
        average = sum / arraySize; // محاسبه میانگین
        Console.WriteLine("Mark List :");
        for (int i = 0; i < arraySize; i++)
           Console.Write("{0,6:f}", list[i]); // نمایش نمرات با فاصله ۶ و اعشاری

        Console.WriteLine("\n\n sum = {0} , average = {1}", sum, average);// مجموع و میانگین
    }
}
```

```
enter Array size: 5
list[1]:14
list[2]:15
list[3]:16
list[4]:17.5
list[5]:18
Mark List :
 14.00 15.00 16.00 17.50 18.00

 sum = 80.5 , average = 16.1
```

مثال: دستوراتی بنویسید که اعداد ۱ تا ۱۰ را به ترتیب در آرایه ذخیره کرده و سپس محتویات آنرا یک خانه به سمت چپ بصورت چرخشی جابجا نماید.



حل: نخست مقدار خانه اول را در متغیری کمکی مثل t ذخیره کرده و سپس با حلقه for همه عناصر را به چپ جابجا می کنیم. یعنی هر خانه با خانه بعدی جایگزین می شود:

```
A[i] = A[i + 1]; // مقدار هر خانه آرایه با مقدار بعدی جایگزین شود
```

بعد از جابجایی همه عناصر، آخرین خانه با خانه اول که در متغیر کمکی t قرار داده شده بود، مقداردهی می شود.

```
class Program
{
    static void Main(string[] args)
    {
        int[] A = {1,2,3,4,5,6,7,8,9,10 };// آرایه با مقادیر اولیه
        int t;
        Console.WriteLine("before shift"); // پیام قبل از جابجایی
        foreach (int m in A)
        {
            Console.Write("{0,3}", m); // نمایش آرایه اولیه
        }
        t =A[0]; // ذخیره مقدار خانه اول در متغیر کمکی
        for (int i = 0; i < 9; i++) // جابجایی عناصر
        {
            A[i] = A[i + 1]; // مقدار هر خانه آرایه با مقدار بعدی جایگزین شود
        }
        A[9] = t; // ذخیره مقدار خانه اول در آخر
        Console.WriteLine("\nafter shift");// پیام بعد از جابجایی
        foreach (int m in A)
        {
            Console.Write("{0,3}",m); // نمایش آرایه نتیجه
        }
        Console.WriteLine();

    }
}
```

```
before shift
  1  2  3  4  5  6  7  8  9 10
after shift
  2  3  4  5  6  7  8  9 10  1
```

فصل هشتم – داده شمارشی، کلاس و متد

نوع داده شمارشی(Enumerated Type):

نوع داده شمارشی مجموعه ای از چند نام دلخواه می باشد که حالت ها و مقادیر مختلف یک موضوع را نشان می دهد . مثلاً برای روزهای هفته به جای اعداد ۱تا ۷ (یا ۰ تا ۶) از نام ها و کلمات معنی دار استفاده می کنیم.

برای تعریف داده شمارشی از کلمه کلیدی enum طبق قالب زیر استفاده می گردد:

```
نام دلخواه enum نوع دسترسی
{
    لیستی از نام‌ها و کلمات
}
```

در لیست نام ها و کلمات در نوع داده شمارشی، هر نام با علامت کاما از نام دیگر جدا می شود .نقطه ویرگول در این تعریف استفاده نمی شود.

مثلاً برای تعریف ایام هفته، بصورت زیر عمل می کنیم:

```
public enum DayOfWeek
{
    Sunday,
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday
}
```

هر یک از اعضای نوع داده شمارشی معادل با یک عدد ثابت است، این اعداد به طور پیش فرض از عدد صفر شروع می شوند و به ترتیب، یک واحد اضافه می شوند.

مثلاً در تعریف فوق به نام Sunday مقدار صفر و به نام Friday مقدار ۵ اختصاص می یابد.

نکته: اگر بخواهید به هر نام مقدار دلخواهی اختصاص دهید توسط علامت مساوی = می توان این کار را انجام داد.

مثال: تعریف ماههای سال میلادی که به اولین ماه عدد ۱ اختصاص داده شده است و به ترتیب به بقیه عددی بعدی ۲ و ۳و ... داده می شود.

```
public enum MonthOfYear
{
    January = 1,
    February,
    March,
    April,
    May,
    June,
    July,
    August,
    September,
    October,
    November,
    December
}
```

استفاده از داده های شمارشی: داده های شمارشی مانند داده های دیگر در برنامه قابل استفاده اند اما باید طبق قالب زیر آنها را بکار برد:

با نوشتن نام نوع داده شمارشی و سپس نام عضو که این دو با علامت نقطه از هم جدا می شوند،می توانیم به اعضا یا مقادیر یک نوع داده شمارشی دسترسی پیدا کنیم.

```
نام عضو . نوع داده شمارشی
        ↓       ↓
DayOfWeek . Saturday
```

تعریف یک متغیر از نوع داده شمارشی:

مانند متغیرهای ساده، متغیر نوع شمارشی قادر است فقط یکی از مقادیر نوع شمارشی را درخود جای دهد

```
نوع داده شمارشی  نام متغیر ;
DayOfWeek         holiday;
```

مقداردهی متغیرهای شمارشی:

مقداردهی متغیرهای نوع شمارشی معمولاً از طریق دستور انتساب انجام می شود:

توجه داشته باشید مقداری که در متغیر نوع شمارشی قرار می گیرد باید با نوع آن مطابقت داشته باشد. مثلاً دخیره عدد ۳ به جای Sunday در متغیر holiday مقدور نیست!

```
نام متغیر = مقدار ;
holiday = DayOfWeek. Sunday;
```

نمایش مقدار یک عضو یا محتوای یک متغیر نوع شمارشی:

همانند متغیرهای دیگر می توان از متد Write برای نمایش مقادیر آنها استفاده نمود.

مثال: شماره ماه را دریافت کرده و نام آنرا به کمک داه شمارشی نمایش دهید.

```
class Program
{
    public enum Months // داده شمارشی نام ماه های سال
    {
        Farwardin,Ordibehest,Khordad,Tir,Mordad,Shahrivar,Mehr,Aban,Azar,Dey,Bahman,Esfand
    }
    static void Main(string[] args)
    {
        Console.Write("shomare mah (1-12):"); // دریافت شماره ماه
        int m = int.Parse(Console.ReadLine());
        Console.Write("Name mah = ");
        switch (m) { // نمایش نام ماه بر اساس شماره آن و داده شمارشی
            case 1: Console.WriteLine(Months.Farwardin); break;
            case 2: Console.WriteLine(Months.Ordibehest); break;
            case 3: Console.WriteLine(Months.Khordad); break;
            case 4: Console.WriteLine(Months.Tir); break;
            case 5: Console.WriteLine(Months.Mordad); break;
            case 6: Console.WriteLine(Months.Shahrivar); break;
            case 7: Console.WriteLine(Months.Mehr); break;
            case 8: Console.WriteLine(Months.Aban); break;
            case 9: Console.WriteLine(Months.Azar); break;
            case 10: Console.WriteLine(Months.Dey); break;
            case 11: Console.WriteLine(Months.Bahman); break;
            case 12: Console.WriteLine(Months.Esfand); break;
        }
    }
}
```

```
shomare mah (1-12):8
Name mah = Aban
```

برنامه نویسی شیئ گرا

شیئ(Object):

- *هر شئ دارای ویژگی هایی است که آن را از اشیای دیگر متمایز می سازد.* مثلاً یک توپ فوتبال دارای ویژگی هایی مانند رنگ، طرح و اندازه است که آن را از توپ فوتبال دیگر متمایز می سازد و یا در وضعیت ساکن و یا در حال حرکت است.
- *بر روی هر شئ نیز عملیاتی می توان انجام داد.* مثلاً در مورد توپ فوتبال می توان عمل شوت زدن، پرت کردن و یا گرفتن توپ را انجام داد.
- اشیاء و موجودات مختلف با یکدیگر در ارتباط و تعامل هستند. توپ فوتبال توسط دانش آموزی شوت می شود، توپ به حرکت درآمده و سرعت می گیرد و وضعیت آن تغییر می کند.

تعریف زبان شیئ گرا: زبان های برنامه نویسی که در آنها امکان تعریف ویژگی ها و رفتارهای اشیاء و موجودات فراهم شده است، همچنین اجازه می دهند اشیایی برمبنای ویژگی های تعریف شده، ایجاد شوند و با یکدیگر ارتباط و نسبت به هم واکنش داشته باشند، زبان های شئ گرا نامیده می شود. C++، C# و java نمونه هایی از این زبانها هستند.

مراحل برنامه نویسی شیئ گرا

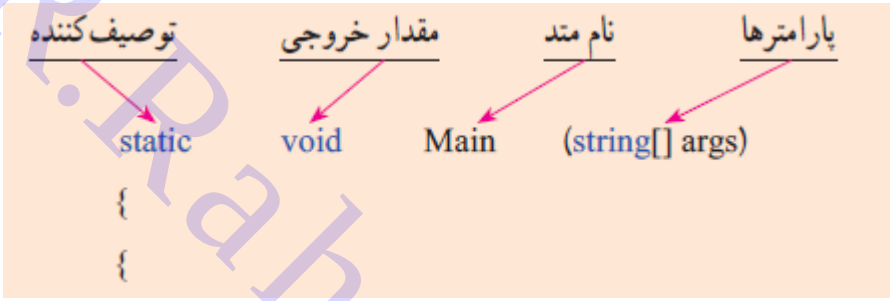
۱) ویژگی ها و وضعیت یک شئ به وسیله تعدادی متغیر که فیلد نامیده می شوند، مشخص می شود
۲) رفتارهای اشیاء در قالب متدها تعریف می گردند
۳) محل و مکان تعریف فیلدها و متدهای یک شئ، در داخل یک کلاس است.

```
class MyClass   // تعریف کلاس
{
    // تعریف فیلدها
    static void Main(string[] args) // عنوان متد
    {
        // بدنه متد
    }
}
```

کلاس(class): تعریف مشخصات، وضعیت و رفتارهای یک شئ را در بردارد و نوع شئ را مشخص می کند

متد(method): مجموعه ای از دستورات است که برای انجام یک عمل خاص و حل یک مسئله کوچک به کار می رود. به طورکلی متد را می توان مانند یک دستگاه در نظر گرفت، که از یک طرف داده هایی وارد آن می شود و از طرف دیگر نتایج پردازش، از آن خارج می شود.

تعریف متد از دو قسمت تشکیل می گردد:



- عنوان متد: مشخصاتی مانند نام متد، نوع داده خروجی، لیست داده های ورودی(پارامترها) و همچنین روش دسترسی به متد و نحوه استفاده از آن معین و تعریف می شود
- بدنه متد: همان دستوراتی است که در داخل متد نوشته می شود

توصیف کننده: روش ایجاد و نحوه دسترسی را مشخص می نماید:

- static: به محض اجرای برنامه قابل استفاده می باشد. متد Main() باید از این نوع باشد. متد ReadLine() نیز از نوع static است که در کلاسی به نام Console تعریف شده است.
- private, protected,public: توصیف کننده های دیگری هستند که نحوه دسترسی را مشخص می کنند. کاربرد آنها بعداً مشخص می گردد.

مقدار خروجی(نوع برگشتی): نوع مقداری را مشخص می کند که متد بعد از پردازش باید برگرداند. مثلاً نوع برگشتی متد ReadLine() از نوع string است. یعنی مقدار حاصل از آن باید در متغیر رشته ای قرار گیرد:

```
String myName = Console.ReadLine();
```

☜ممکن است متدی نوع برگشتی نداشته باشد. در این صورت نوع برگشتی آن void یا همان بدون برگشتی است. مثل متد WriteLine() که فقط چیزی نمایش می دهد و نتیجه ای به ما نمی دهد.

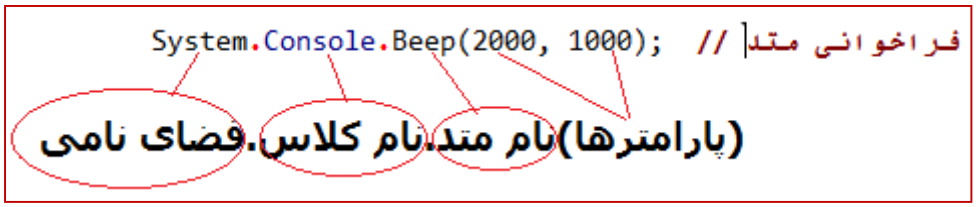
```
Console.WriteLine(myName);
```

☜ممکن است متدی مقدار ورودی نداشته باشد؛ مثل ReadLine() که داخل پرانتز خالی است.

نام متد: نامی دلخواه با رعایت قوانین نامگذاری که بهتر است بیان کننده کاری باشد که انجام می دهد.

لیست پارامتر ارسالی: تعداد و نوع داده های ورودی به متد را مشخص می کند. مثلاً پارامتر ورودی متد WriteLine() رشته ای است که قرار است نمایش دهد.

فراخوانی متد: منظور از فراخوانی، استفاده از متد در برنامه است که با تعیین و ارائه پارامترها به آن کاری انجام می دهد.



کار با متدها و کلاس های آماده

کلاس Math: برای انجام اعمال ریاضی، توابعی در این کلاس تعریف شده اند که بصورت static می باشند و مستقیماً قابل استفاده هستند. این کلاس، در فضای نامی System تعریف شده است.

برخی از خصوصیات و متدهای کلاس Math به شرح زیر است:

- Pow(x,y) : پارامتر x را به توان y می رساند. یعنی محاسبه x^y که ورودی آن (x,y) از نوع اعشاری و خروجی نیز اعشاری با دقت بالاست.

```
double tavan = Math.Pow(5, 2);
Console.WriteLine(tavan); // --> ۲۵ نتیجه نمایش نتیجه ۵ به توان ۲ یعنی
```

مثال: همه توانهای عدد ۲ را بنویسید($2^0,2^1,...,2^{16}$).

```
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
2^6 = 64
2^7 = 128
2^8 = 256
2^9 = 512
2^10 = 1024
2^11 = 2048
2^12 = 4096
2^13 = 8192
2^14 = 16384
2^15 = 32768
2^16 = 65536
```

```
class Program
{
    static void Main(string[] args)
    {
        for (int i = 0; i <= 16; i++)
        {
            double number = Math. Pow (2, i);
            Console. WriteLine ("{0}^{1} = {2}",2,i,number);

        }
    }
}
```

- عدد π: این عدد در ریاضی برابر نسبت محیط دایره به قطر آن است و برابر عدد ثابت ۳,۱۴ می باشد. برای استفاده از این عدد با دقت بالا از ثابت Math.PI استفاده کنید. مثال: مساحت دایره:

```
// مساحت دایره = πr2
double r = 5; // شعاع دایره
double mosahat = Math.PI * Math.Pow(r, 2); //مساحت دایره
```

برخی از متدهای دیگر در جدول زیر آمده اند.

جدول ۱ــ۴ــ برخی متدهای ریاضی از کلاس Math

نام متد	کاربرد	شکل کلی	مثال
Cos	کسینوس یک زاویه برحسب رادیان را حساب می کند.	Math.Cos (عدد)	Math.Cos(10)
Log	لگاریتم عدد را در مبنای تعیین شده حساب می کند.	Math. Log (مبنا، عدد)	Math. Log(16,2)
Max	عدد بزرگ‌تر از بین دو عدد را پیدا می کند.	Math.Max (عدد، عدد)	Math.Max(10,20)
Min	عدد کوچک‌تر از بین دو عدد پیدا می کند.	Math.Min (عدد،عدد)	Math.Min(10,20)
Pow	حاصل عدد به توان تعیین شده را حساب می‌کند.	Math.Pow (توان، عدد)	Math.Pow(10,3)
Round	عدد را گرد می کند.	Math.Round (عدد)	Math.Round(10.7)
Sin	سینوس یک زاویه برحسب رادیان را حساب می کند.	Math.Sin (عدد)	Math.Sin(20)
Sqrt	جذر عدد را حساب می‌کند. همان رادیکال خودمان.	Math.Sqrt (عدد)	Math.Sqrt(100)
Tan	تانژانت یک زاویه برحسب رادیان را حساب می کند.	Math.Tan (عدد)	Math.Tan(20)
Truncate	بخش اعشار عدد را حذف می کند.	Math.Truncate (عدد)	Math.Truncate (10.5)

مثال ۱: کاربرد توابع Math

```
static void Main(string[] args)
{
    Console.WriteLine("Max(100, 200) = " + Math.Max(100, 200)); // بیشترین
    Console.WriteLine("Min(100, 200) = " + Math.Min(100, 200)); //کمترین
    Console.WriteLine("Pow(2, 3) = " + Math.Pow(2, 3));         //توان
    Console.WriteLine("Round(18.2) = " + Math.Round(18.2));     // گرد به پایین
    Console.WriteLine("Round(18.5) = " + Math.Round(18.5));     //گرد وسط
    Console.WriteLine("Round(18.6) = " + Math.Round(18.6));     // گرد به بالا
    Console.WriteLine("Sqrt(225) = " + Math.Sqrt(225));         // ریشه یا جذر
    Console.WriteLine("Truncate(1.73) = " + Math.Truncate(1.73)); // حذف اعشار
    Console.WriteLine("Log(1024,2) = " + Math.Log(1024, 2)); // لگاریتم مبنای ۲
    Console.WriteLine("Log(1024,10) = " + Math.Log(1024, 10)); // لگاریتم مبنای ۱۰
}
```

خروجی:

```
Max(100, 200) = 200
Min(100, 200) = 100
Pow(2, 3) = 8
Round(18.2) = 18
Round(18.5) = 18
Round(18.6) = 19
Sqrt(225) = 15
Truncate(1.73) = 1
Log(1024,2) = 10
Log(1024,10) = 3.01029995663981
```

مثال: سینوس و کسینوس و تانژانت همه زاویه های تا ۹۰ درجه را با فاصله ۱۵ درجه نمایش دهید.

حل: تابع سینوس و کسینوس پارامتر زاویه را برحسب رادیان می پذیرند. پس باید درجه به رادیان تبدیل شود:

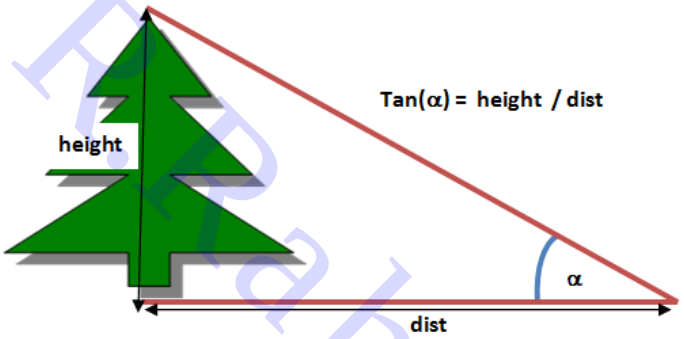
زاویه بر حسب رادیان =زاویه بر حسب درجه)/۱۸۰(عدد پی *

```
static void Main(string[] args)
{
    for (int i = 0; i <= 90; i+=15)  // زاویه بر حسب درجه
    {
        double rad = (i * Math.PI) / 180;  // زاویه برحسب رادیان
        double sinus = Math.Sin(rad);
        double cosinus = Math.Cos(rad);
        double tangant=0;
        if (i != 90)   // تانژانت ۹۰ درجه تعریف نشده است
            tangant = Math.Tan(rad);
        // نمایش اعداد نتیجه با ۵ فاصله و بصورت اعشار با دقت کم
        Console.WriteLine("sin({0}) = {1,5:F} , cos({2}) = {3,5:F},tan({4}) = {5,5:F} "
                          ,i, sinus, i, cosinus,i,tangant); // ادامه دستور در خط دوم
    }
}
```

```
sin(0) =  0.00 , cos(0) =  1.00 , tan(0) =  0.00
sin(15) =  0.26 , cos(15) =  0.97 , tan(15) =  0.27
sin(30) =  0.50 , cos(30) =  0.87 , tan(30) =  0.58
sin(45) =  0.71 , cos(45) =  0.71 , tan(45) =  1.00
sin(60) =  0.87 , cos(60) =  0.50 , tan(60) =  1.73
sin(75) =  0.97 , cos(75) =  0.26 , tan(75) =  3.73
sin(90) =  1.00 , cos(90) =  0.00 , tan(90) =  0.00
```

تمرین ۱: با در دست داشتن طول سایه(dist) یک درخت و زاویه دید نسبت به نوک درخت(α)، ارتفاع آنرا (height)حساب کنید.

راهنمایی: تانژانت زاویه برابر است با ارتفاع تقسیم بر سایه



توجه کنید که زاویه باید بر حسب رادیان تبدیل شود:

$Rad = (\alpha * \pi)/180$

پس ارتفاع برابر است با:

$Height = dist * tan(rad)$

تمرین ۲: سه عدد ضرایب معادله درجه ۲ را بخوانید و ریشه های آنرا نمایش دهید ($ax^2+bx+c=0$): راهنمایی: از روش دلتا و تابع Math.Sqrt() برای محاسبه جذر استفاده نمایید:

$$\Delta = b^2 - 4ac \quad , \quad x1 = \frac{-b+\sqrt{\Delta}}{2a} \quad , x2 = \frac{-b-\sqrt{\Delta}}{2a}$$

کلاس String: برای ذخیره رشته کارکترها از این کلاس استفاده می کنیم. کارکترها به ترتیب مانند آرایه ا زمحل صفر تا آخر ذخیره می شوند. پس برای دسترسی به یک کارکتر خاص از رشته مانند آرایه می توان اندیس کارکتر مورد نظر را مشخص نمود.

|شماره کاراکتر| نام رشته متنی

```
string test = "sample";
Console.WriteLine(test[0]); // نمایش حرف اول رشته  --> s
Console.WriteLine(test[5]); // نمایش حرف آخر رشته  --> e
```

نکته ۱ : اندیس رشته مانند آرایه از صفر شروع می شود.

نکته ۲: مقدار رشته قابل تغییر نیست و با هر بار مقداردهی دوباره از اول ساخته می شود. یعنی نمی توان مثلاً یک کارکتر از آن را عوض کرد.

کلاس string دارای متدهای جدول روبرو است. این متد ها روی خود رشته تأثیر نمی گذارند بلکه یک رشته دیگر بر می گردانند . بنابراین محتوای متغیر رشته ها را تغییر نمی دهند

جدول ۲ــ۴ــ برخی متدهای رشته‌ای از کلاس String[1]

متد	شرح کار متد	نوع برگشتی
CompareTo()	مقایسه دو رشته	bool
IndexOf()	موقعیت وجود یک کاراکتر در رشته را برمی گرداند.	int
ToLower ()	تبدیل تمام کاراکترهای یک رشته به حروف کوچک	string
ToUpper ()	تبدیل تمام کاراکترهای یک رشته به حروف بزرگ	string
Insert()	درج یک کاراکتر یا یک رشته در درون یک رشته دیگر	string
Length	تعداد کاراکترهای یک رشته را بر می گرداند.	int
Replace()	یک کاراکتر در یک رشته را با کاراکتر یا یک رشته دیگری عوض می کند.	string

مثال: مقدار یک رشته را بصورت حروف بزرگ و کوچک نمایش دهید:

```
string test = "sAmPlE";
Console.WriteLine(test.ToLower()); // نمایش حروف کوچک  --> sample
Console.WriteLine(test.ToUpper()); // نمایش حروف بزرگ  --> SAMPLE
```

مثال: عملکرد متدهای کلاس String

```
static void Main(string[] args)
{
    string s = " This is a String! ";
    Console.WriteLine("the main string : \"{0}\"", s);  // رشته اصلی
    Console.WriteLine(" ToLower: \"{0}\"", s.ToLower()); // حروف کوچک
    Console.WriteLine(" ToUpper: \"{0}\"", s.ToUpper());// حروف بزرگ
    Console.WriteLine(" Insert: \"{0}\"", s.Insert(0, "Note,")); //درج کلمه جدید در محل صفر
    Console.WriteLine(" Length: {0}", s.Length); // طول رشته = تعداد کارکتر
    Console.WriteLine(" index of 'i': {0}", s.IndexOf('i'));  // محل اندیس در رشته
    Console.WriteLine(" Replace:\"{0}\"", s.Replace("This is", "we have")); // جایگزینی
    Console.ReadKey();
}
```

تمرین ۳: یک جمله متن از ورودی دریافت کرده و تمام حروف a آنرا به حرف e تغییر دهید.

کلاس Array

این کلاس شامل متدهای مختلف استاتیک، برای عملیات بر روی آرایه ها است که در جدول زیر آورده شده اند:

متدها	توضیحات
Clear( )	با توجه به نوع عناصر آرایه مقدار صفر یا false و یا null را به آنها اختصاص می دهد.
Copy(Array, Array, num)	تعدادی از عناصر آرایه اول از ابتدای آرایه را در آرایه دوم کپی می نماید. این تعداد توسط num مشخص می شود.
CopyTo(Array, num)	تمام عناصر آرایه یک بعدی فعلی را در آرایه یک بعدی دیگر از یک عنصر خاص کپی می نماید. اندیس آن عنصر توسط num مشخص می شود.
GetLength	یک عدد 32-bit که نشان دهنده تعداد عناصر یک بعد خاص از آرایه می باشد را بر می گرداند.
GetLongLength	یک عدد 64-bit که نشان دهنده تعداد عناصر یک بعد خاص از آرایه می باشد را بر می گرداند.
GetLowerBound	حد پایین یک بعد خاص از آرایه را بر می گرداند.
GetType	نوع آرایه را مشخص می کند.
GetUpperBound	حد بالای یک بعد خاص از آرایه را بر می گرداند.
GetValue(Int32)	مقدار یک عنصر در یک اندیس خاص از یک آرایه تک بعدی را بر می گرداند. این اندیس با یک عدد 32-bit مشخص می شود.

IndexOf(Array, Object)	اندیس اولین رخداد object را در آرایه بر می گرداند.
Reverse(Array)	ترتیب عناصر آرایه یک بعدی را معکوس می نماید.
SetValue(Object, Int32)	یک مقدار را به یک مکان خاص از آرایه اختصاص می دهد. اندیس این مکان با یک عدد 32-bit مشخص می شود.
Sort(Array)	عناصر یک ارایه را مرتب می نماید.
ToString	یک رشته را بر می گرداند که نشان دهنده شیئ فعلی می باشد.

مثال: توابع آرایه را در مثال زیر ببینید

```
static void Main(string[] args)
{
    float [] myArray = {15,17,13,16,14.5f,20,18.5f,14,12,10 };// آرایه
    float[] toCopy = new float[10]; // آرایه خالی
    Console.WriteLine(" Initial Array: ");
    for (int i = 0; i < myArray.Length; i++) { // نمایش آرایه اصلی
        Console.Write(" {0}\t ", myArray[i]);
    }
    Console.WriteLine(" Sorted Array: ");
    Array.Sort(myArray);   // مرتب سازی
    for (int i = 0; i < myArray.Length; i++)  // نمایش آرایه مرتب شده
    {
        Console.Write(" {0}\t ", myArray[i]);
    }
    Console.WriteLine(" Copied Array: ");
    Array.Copy(myArray, toCopy, myArray.Length);  // کپی آرایه در دیگری
    for (int i = 0; i < toCopy.Length; i++)  // نمایش آرایه کپی شده
    {
        Console.Write(" {0}\t ", toCopy[i]);
    }
    Console.WriteLine(" Reverse Array: ");
    Array.Reverse(toCopy);                  //معکوس کردن آرایه
    for (int i = 0; i < toCopy.Length; i++)// نمایش آرایه معکوس شده
    {
        Console.Write(" {0}\t ", toCopy[i]);
    }
}
```

```
Initial Array:
15      17      13      16      14.5    20      18.5    14      12      10
 Sorted Array:
10      12      13      14      14.5    15      16      17      18.5    20
 Copied Array:
10      12      13      14      14.5    15      16      17      18.5    20
 Reverse Array:
20      18.5    17      16      15      14.5    14      13      12      10
```

فصل نهم – ایجاد برنامه های ویندوزی

برنامه هایی که تا کنون نوشتیم کنسولی بودند و برای برنامه های کاربردی مناسب نیستند. حالا که با اصول برنامه نویسی آشنا شده ایم، نیاز است که برنامه های ویندوزی و مبتنی بر فرم، منو و ... را طراحی کنیم.

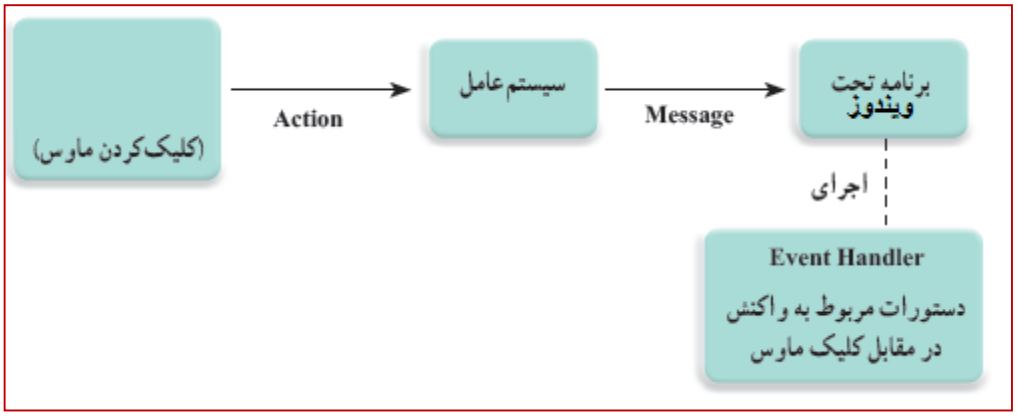
در برنامه های کنسولی، اجرای برنامه از متدی به نام Main() شروع می شود و دستورات داخل آن، به ترتیب و خط به خط اجرا می شوند .در چنین برنامه هایی با استفاده از متدهای موجود مانند ReadLine() برای دریافت یک رشته، درخواست هایی به سیستم عامل داده می شود که عملیاتی را برای ما انجام دهد.

در برنامه های ویندوزی نیز اجرای برنامه با متدی به نام Main() شروع می شود اما برخلاف برنامه های کنسول، برنامه

در حالت انتظار قرار می گیرد تا یک اتفاق یا رویداد رخ دهد(مانند کلیک روی یک دکمه) که در این صورت نسبت به آن واکنش نشان دهد

رویداد(Event) چیست؟ رویداد یک اطلاع (Notification) است که از طرف سیستم عامل به برنامه داده می شود تا نشان دهد که یک اتفاق رخ داده است.

مدیریت رویداد(Event Handler یا EH): برنامه نویس باید پیش بینی کند اگر کاربر عملی را انجام دهد، برنامه چگونه نسبت به آن واکنش نشان دهد و برای این منظور باید متدهایی را بنویسد که در مواجهه با یک رخداد یا رویداد مانند کلیک ماوس در یک محل، به کامپیوتر اعلام کند که چه کاری باید انجام شود.



شکل کلی متد مدیریت رویداد(EH) بصورت زیر است:

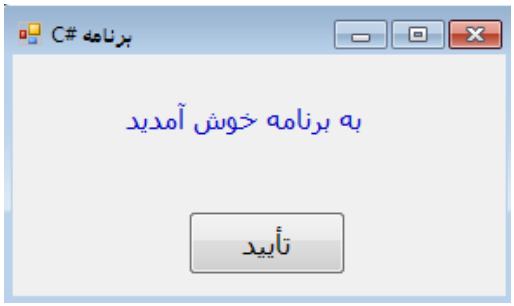
```
void نام متد (فرستنده پیام، جزئیات رویداد)
{
دستورات واکنش به رویداد
}
```

هر متد EH دارای دو پارامتر ورودی است که اولی نوع شئ(Object) فرستنده پیام را مشخص می کند و پارامتر دوم مربوط به جزئیات رویداد است مثلاً در مورد کلیک ماوس شامل اطلاعاتی در مورد موقعیت مکانی ماوس در لحظه کلیک و کلیدی(کلید چپ، راست یا وسط)از ماوس است که فشار داده شده است.

تفاوت دیگر بین برنامه ویندوزی و کنسولی این است که، برنامه های ویندوزی پس از واکنش به رویدادها و انجام عملیات مربوطه، مجدداً در حالت انتظار برای رویداد بعدی به سر می برند تا در نهایت کاربر از برنامه خارج گردد.

```
private void btnOk_Click(object sender, EventArgs e)
{
   /*
    محل نوشتن دستورات واکنش به رویداد
    کلیک روی دکمه
    */
}
```

واسط گرافیکی کاربری



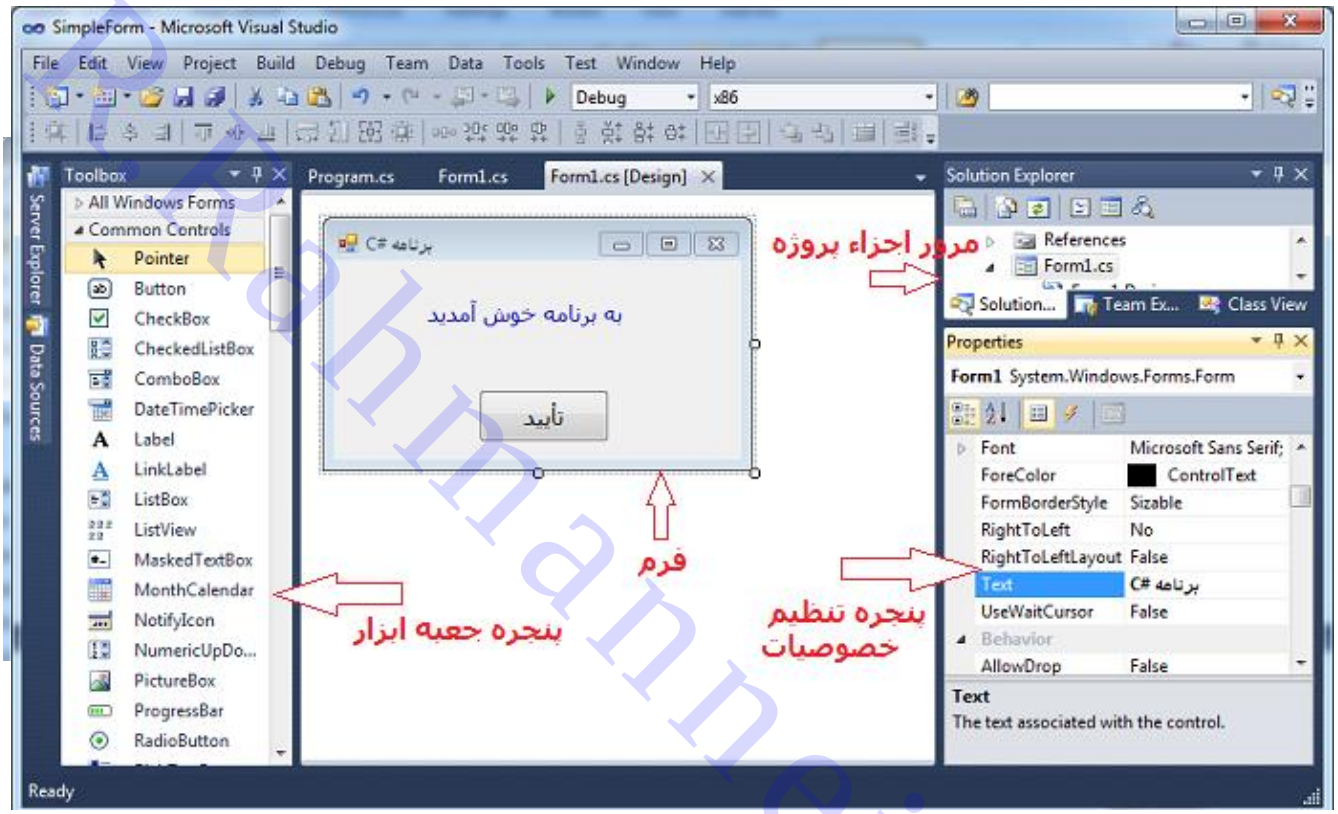
در ساخت یک برنامه ویندوزی باید صفحه یا فرمی در اختیار داشته باشید تا انواع دکمه ها، منوها، تصاویر و نوشته ها را روی آن قرار دهید . این فرم، واسط گرافیکی کاربر یا GUI نامیده می شود.

فرم، صفحه ای است که اجزای گرافیکی گوناگونی بر روی آن قرار می گیرد .اندازه یک فرم متناسب با تعداد و اندازه اشیای گرافیکی است که قرار است روی آن جای گیرند.

ایجاد پروژه ویندوزی در VS

در محیط برنامه نویسی VS مسیر زیر را دنبال کنید:

File → new →project →Visual C# → Windows Forms Application → انتخاب نام و مسیر ذخیره پروژه



آشنایی با پنجره ویژگی ها و خواص اشیاء (properties)

در این پنجره ویژگی ها و خواص شئ و همچنین واکنش به رویدادهای هر شئ را می توانیم مقداردهی و تنظیم کنیم.

برخی مشخصه های مهم اشیاء عبارتند از:

- Text: تنظیم عنوان
- Font: تنظیم نوع، اندازه فونت نوشته های روی کنترلها
- ForeColor: رنگ نوشته های روی کنترلها
- BackColor: رنگ پس زمینه
- Name: نام کنترل
- Size: اندازه پهنا و ارتفاع کنترل بر حسب پیکسل
- Location: موقعیت کنترل روی فرم نسبت به گوشه چپ بالا
- Visible: اگر true باشد، نمایش داده می شود و اگر false باشد، کنترل نامرئی خواهد بود
- ...

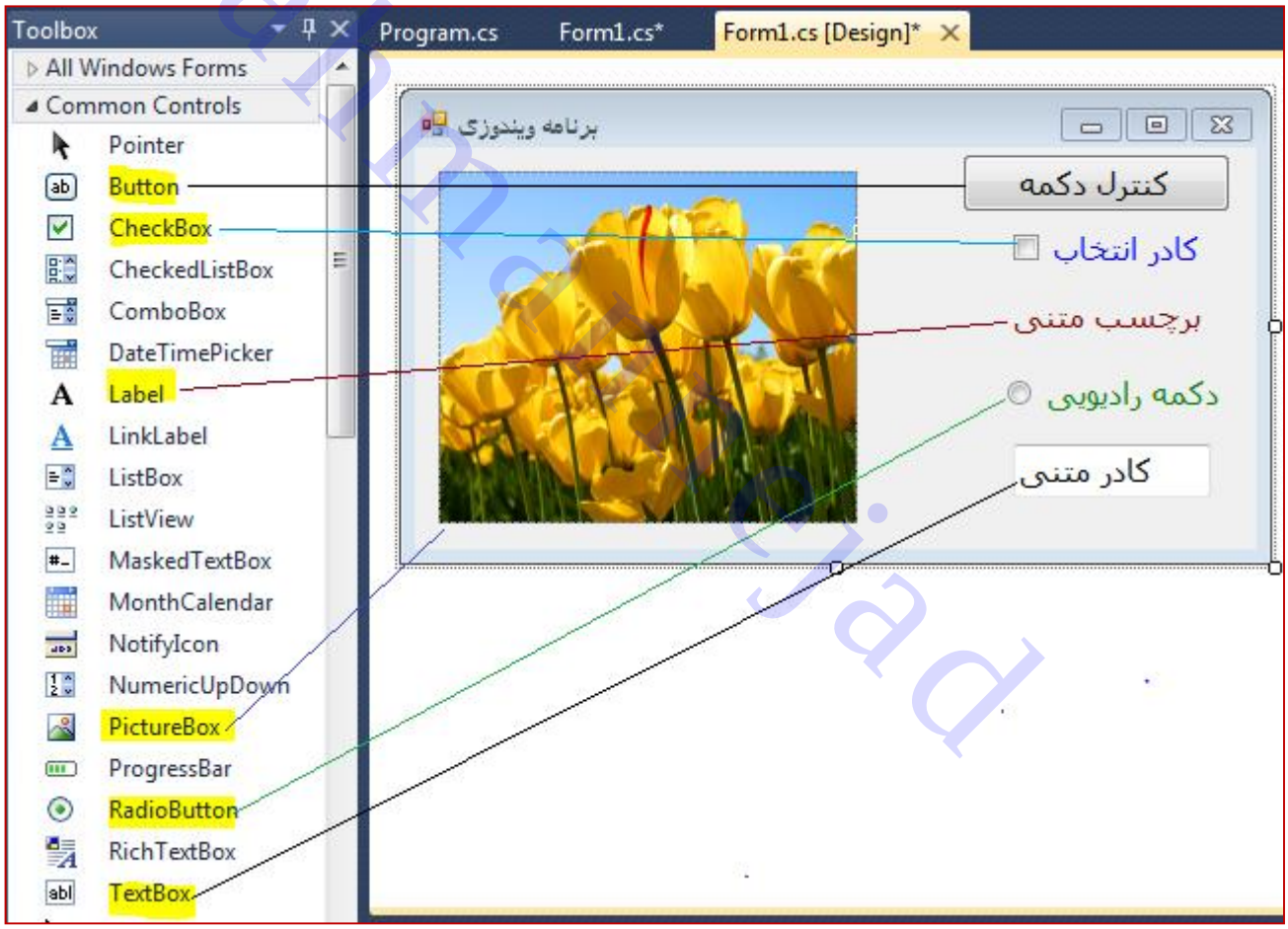
استفاده از جعبه ابزار و قراردادن کنترل ها در روی فرم

برای قرار دادن یک کنترل بر روی فرم، می توانید روی کنترل مورد نظر دوبار کلیک کنید و یا با درگ کردن کنترل و رها کردن آن روی فرم، کنترل مورد نظر را روی فرم بچسبانید.

برخی کنترلهای پرکاربرد عبارتند از:

- Button: دکمه قابل کلیک برای اجرای برنامه ها
- Label: برچسب متنی برای نمایش یک پیام یا متن

- TextBox: کادر متنی برای ورود اطلاعات
- RadioButton: دکمه رادیویی برای انتخاب یک حالت از بین چند حالت
- CheckBox: کادر انتخاب برای انتخاب حالت با تیک زدن
- PictureBox: کادر تصویر برای نمایش تصویرها
- ...



تمرین: فرم زیر را طراحی کنید

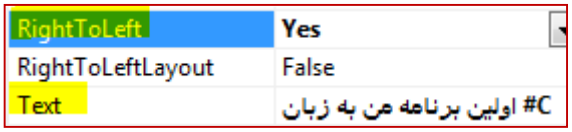
حل: مراحل زیر را دنبال کنید

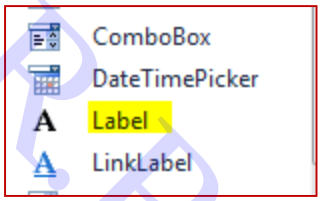
۱) ایجاد پروژه جدید

File→ new→project →Visual C# →
Windows Forms Application→ انتخاب نام و مسیر

۲) *روی فرم کلیک کرده و خصوصیات زیر را در پنجره Properties تنظیم کنید*

RightToLeft →yes
Text → C# اولین برنامه من به زبان

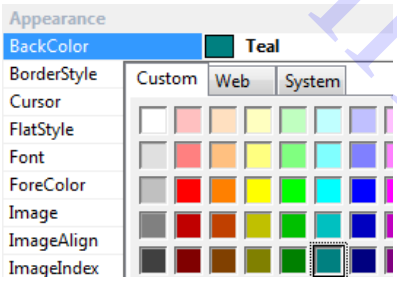




۳) در پنجره Toolbox کنترل Label دوبار کلیک کنید یا با ماوس روی فرم بکشید

۴) آنرا با ماوس روی فرم جابجا کنید تا در محل مناسب قرار گیرد یا از منوی Format گزینه Center in Form |Horizontally را انتخاب نمایید.

۵) روی آن کلیک کرده تا انتخاب شود. *خصوصیات زیر را در پنجره Properties تنظیم کنید*



RightToLeft →yes

Text → به برنامه من خوش آمدید

BackColor →Custom رنگ سبز تیره در زبانه

ForeColor → رنگ زرد

Font →Tahoma size:14

۶) قراردادن کنترل PictureBox روی فرم و تنظیم اندازه آن با ماوس و سپس تنظیم خصوصیات زیر:

sizeMode →StretchImage تا تصویر به اندازه کادر آن درآید

Image → انتخاب تصویر دلخواه با زدن دکمه ...

در مرحله انتخاب تصویر با پنجره زیر روبرو می شوید. دکمه Import را زده، تصویر را انتخاب نموده و سپس Ok را بزنید.

چگونه از تغییر اندازه فرم جلوگیری کنیم؟

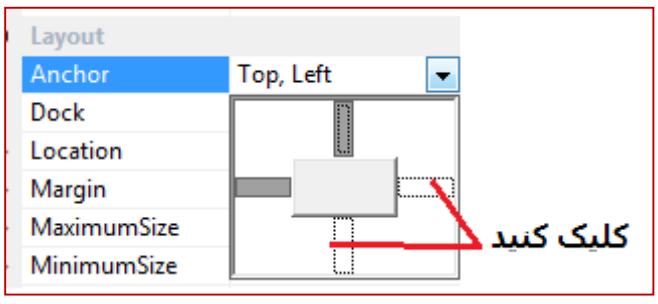
ویژگی FormBorderStyle مربوط به فرم در پنجره Proprties نحوه کنترل تغییر اندازه را با تنظیم مقادیر زیر مشخص می کند:

- Sizeable: حالت عادی و امکان تغییر اندازه
- FixedSingle : تغییر اندازه ممکن نیست اما می توان آنرا maximize نمود(کل صفحه)
- FixedDialog: تغییر اندازه ممکن نیست اما می توان آنرا maximize نمود(کل صفحه)
- None : بدون کنترلهای × ☐ ⊡ (بستن، بیشینه و کمینه کردن فرم) و نیز نوار عنوان فرم

چگونه با تغییر اندازه فرم، اندازه تصویر داخل فرم نیز تغییر کند؟

روی کادر تصویر کلیک کنید تا انتخاب شود. سپس در پنجره Proprties ویژگی Anchor (یعنی لنگرگاه) آنرا تنظیم کنید. این ویژگی فاصله ثابت از چهار طرف را تنظیم می کند که بصورت پیش فرض Top, Left تنظیم شده است. با کلیک روی فاصله از راست(right) و فاصله از پایین(Bottom) نوار مربوطه به رنگ خاکستری در می آید. با این کار فاصله تصویر از هر چهار طرف ثابت می ماند و با بزرگ کردن فرم، تصویر نیز متناسباً بزرگ می شود.

برای اینکه برچسب(Label) نیز در وسط باقی بماند، خصوصیت Anchor آنرا فقط به Top تنظیم کنید.

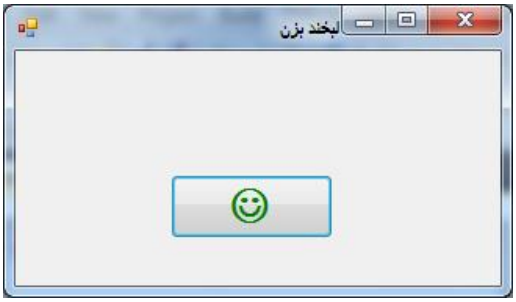


نکته ۱: برای تنظیم جهت راست به چپ برای متون فارسی در فرم ها، هردو خصوصیت RightToLeft و RightToLeftLayout مربوط به فرم را به True تنظیم نمایید.

نکته ۲: اگر ویژگی ControlBox مربوط به فرم را به False تغییر دهید، کنترلهای × ☐ ⊡ (بستن، بیشینه و کمینه کردن فرم)در گوشه سمت راست بالا ناپدید می شوند و برای بستن برنامه باید Alt+F4 را بزنید.

تمرین: فرم زیر را بسازید

ایجاد برنامه های ویندوزی با واکنش نسبت به رویدادها



مثال ۱: فرمی به شکل زیر طراحی کنید.

در پروژه ای جدید، دکمه ای روی فرم قرار دهید و Font آنرا به Windings و اندازه ۲۴ و رنگ زمینه سبز تغییر دهید و در عنوان Text حرف J را وارد کنید.

اگر عنوان Text را به حرف K یا L تغییر دهید، علامتهای دیگری بدست می آورید.

نکته : میتوانید تصویری را به عنوان پس زمینه دکمه قرار دهید :

انتخاب تصویر پس زمینه → BackgroundImage

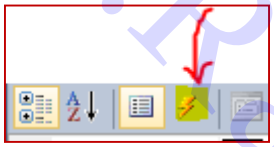
Stretch → BackgroundImageLayout

تا کل تصویر روی دکمه جا داده شود

مثال: می خواهیم با زدن دکمه لبخند، پیامی روی فرم ظاهر شود.

حل: یک کنترل Label روی فرم قرار دهید و در خصوصیت Text آن متن دلخواهی بنویسید. خصوصیت Visible آنرا با false تنظیم کنید که فعلاً مخفی باشد.

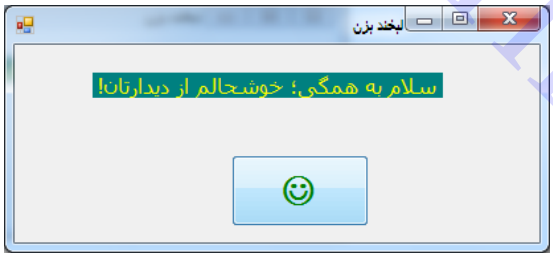
حال روی دکمه لبخند کلیک کنید تا انتخاب شود و سپس در قسمت properties روی آیکن زیر کلیک کنید تا لیست رویداد ها نمایش داده شود، روی کادر مقابل رویداد Click دوبار کلیک کنید.



قسمت کد نویسی فعال می شود دستور زیر را داخل رویداد بنویسید تا برچسب حاوی پیام از حالت مخفی درآید و ظاهر شود:

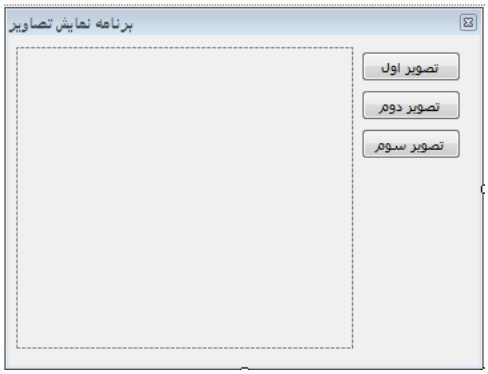
```
private void button1_Click(object sender, EventArgs e)
{
    // محل نوشتن رویداد
    label1.Visible = true; // برچسب ظاهر شود
}
```

حال برنامه را اجرا کنید و سپس روی دکمه کلیک کنید تا دستور مربوط به ظاهرسازی برچسب اجرا گردد:



مثال:

۱) در یک پروژه جدید، سه دکمه و یک کادر تصویر به شکل زیر اضافه کنید

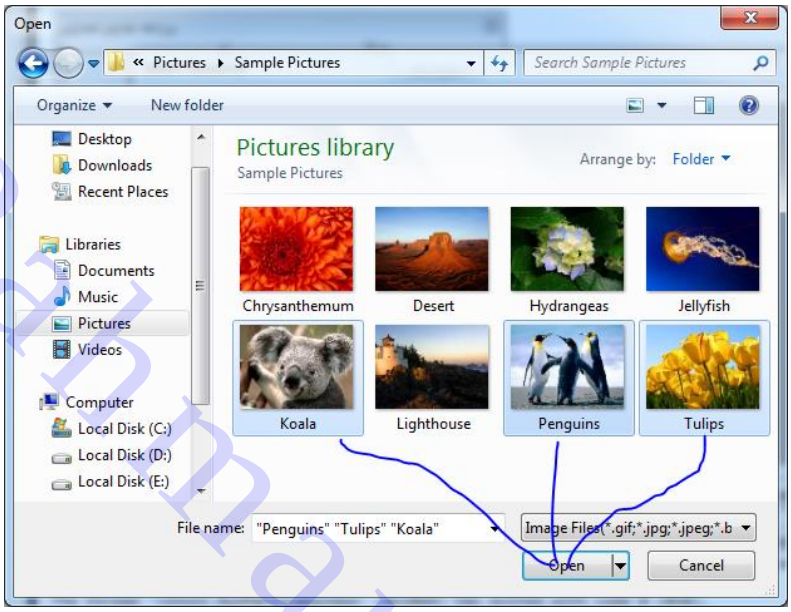


۲) خصوصیت FormBorderStyle مربوط به فرم را به مقدار FixedToolWindow تغییر دهید تا فقط کنترل علامت بستن × روی عنوان فرم باشد.

۳) خصوصیات زیر را برای کنترل کادر تصویر تنظیم کنید:

Name: pic
sizeMode: StretchImage

۴) در قسمت Image و با زدن دکمه ... سه تصویر دلخواه را با زدن دکمه import اضافه کنید.



۵) تصاویر انتخابی در پوشه Resources از مسیر ذخیره پروژه قرار می گیرند و توسط دستور زیر می توان آنها را در برنامه به کادر تصویر مرتبط کرد.

Pic.Image = Properties.Resources.نام منبع یا نام فایل

۶) با کلیک روی هر دکمه می خواهیم تصویر انتخابی نمایش داده شود. برای اینکار در رویداد Click هر دکمه دستورات زیر را بنویسید:

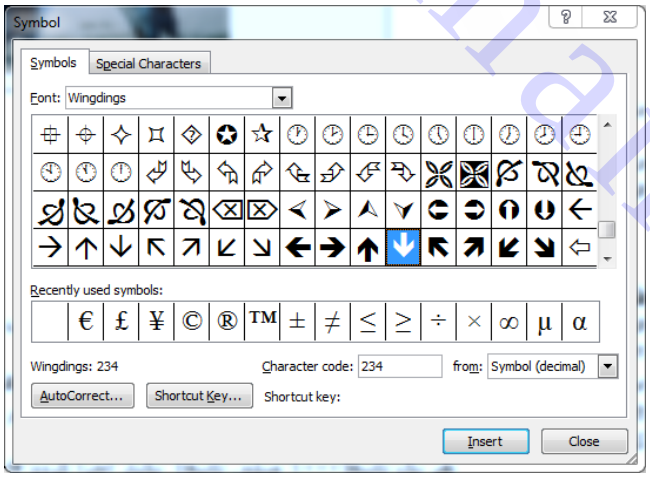
```
private void button1_Click(object sender, EventArgs e)// کلیک دکمه اول
{
    pic.Image = Properties.Resources.Koala; //Koala انتخاب فایل
}

private void button2_Click(object sender, EventArgs e)// کلیک دکمه دوم
{
    pic.Image = Properties.Resources.Penguins;//Penguins انتخاب فایل
}

private void button3_Click(object sender, EventArgs e)// کلیک دکمه سوم
{
    pic.Image = Properties.Resources.Tulips;//Tulips انتخاب فایل
}
```

خروجی بعد از کلیک روی دکمه دوم:

مثال: حرکت دادن یک برچسب به کمک دکمه های فرمان

۱) فرمی به شکل زیر طراحی کنید و نام Label را به LblMove تغییر دهید

۲) برای نوشتن علامت فلشها از مجموعه فونت Wingdings و کدهای اسکی ۲۳۱ تا ۲۳۴ با نگه داشتن Alt و تایپ عدد ها در قسمت ماشین حساب صفحه کلید، در مشخصه Text استفاده کنید. در برنامه Word می توانید آنها را در قسمت Insert Symbol مشاهده نمایید. مطابق شکل زیر:



۳) برای رویداد دکمه ها و جابجایی برچسب از خصوصیت Top(فاصله از بالای فرم) و Left (فاصله از چپ) استفاده می کنیم. مثلاً برای جابجایی به پایین باید خصوصیت Top زیاد شود و برای حرکت به چپ خصوصیت Left کاهش یابد.

```
private void button1_Click(object sender, EventArgs e)
{
    LblMove.Top -= 10; // حرکت به بالا
}

private void button2_Click(object sender, EventArgs e)
{
    LblMove.Top += 10; // حرکت به پایین
}

private void button3_Click(object sender, EventArgs e)
{
    LblMove.Left += 10;// حرکت به راست
}

private void button4_Click(object sender, EventArgs e)
{
    LblMove.Left -= 10;// حرکت به چپ
}
```

تمرین: حرکت در هر جهت را کنترل کنید که از حاشیه بیرون نرود. مثلاً در حرکت به راست از حاشیه سمت راست فراتر نرود. برای اینکار مشخصه Left برچسب نباید از پهنای فرم بیشتر شود. عدد ۵۰ در مثال زیر برای لحاظ کردن حاشیه های فرم است.

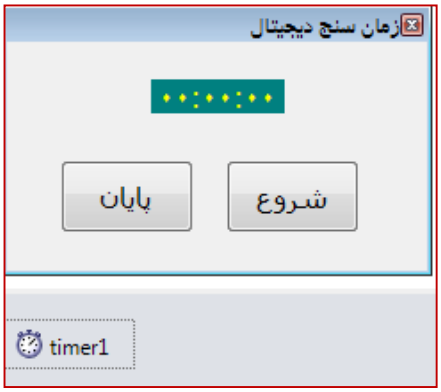
```
private void button3_Click(object sender, EventArgs e)
{
    int w = Form1.ActiveForm.Width; // بدست آوردن پهنای فرم
    if (LblMove.Left < w - 50) // موقعیت از چپ کمتر از پهنای فرم
        LblMove.Left += 10;// حرکت به راست
}
```

تمرین: یک ماشین حساب ساده به شکل زیر طراحی کنید:

راهنمایی: برای دکمه ها مانند جمع دستوراتی مشابه زیر بنویسید:

```
private void button1_Click(object sender, EventArgs e)
{
    int a, b, c;
    a = int.Parse(textBox1.Text);// دریافت عدد اول
    b = int.Parse(textBox2.Text);// دریافت عدد دوم
    c = a + b;//محاسبه
    textBox3.Text = string.Format("{0}",c); // نمایش نتیجه در کادر سوم
}
```

طراحی زمان سنج دیجیتال



۱) فرمی به شکل زیر طراحی کنید که دارای دو کنترل دکمه شروع و پایان(Button) به نامهای startBtn و stopBtn و یک برچسب(Label) و یک Timer یا زمان سنج است.

۲) برای تنظیم جهت راست به چپ برای متون فارسی در فرم ها، هردو خصوصیت RightToLeft و RightToLeftLayout مربوط به فرم را به True تنظیم نمایید.

۳) کنترل زمان سنج(Timer) را به شکل زیر تنظیم کنید:

Interval : 1000 → هر یک ثانیه(۱۰۰۰ میلی ثانیه) یکبار اجرا گردد

Enabled: false → در شروع کار غیرفعال شود

- کنترل زمان سنج برای اطلاع از سپری شدن فاصله های زمانی یکسان به کار می رود.

- فاصله زمانی توسط خصوصیت Interval تعیین می شود که بر حسب میلی ثانیه می باشد.
- رویداد Tick دارد که دستورات داخل این رویداد به اندازه فاصله زمانی تعیین شده در Interval اجرای مجدد خود را انجام می دهد.
- مثلاً اگر Interval = 500 باشد، هر نیم ثانیه یکبار دستورات داخل رویداد Tick اجرا خواهند شد.
- خصوصیت Enabled اگر true باشد، زمان سنج کار می کند و گرنه متوقف می گردد.

۴) برای ذخیره مقادیر ساعت، دقیقه و ثانیه در بخش کدنویسی داخل class و قبل از Formمتغیرهای زیر را تعریف نمایید.

```
public partial class Form1 : Form
{
    byte hour, second, minutes;  // ذخیره ساعت، دقیقه، ثانیه
    public Form1()
    {
        InitializeComponent();
    }
```

۵) برای دکمه «شروع» زمان سنج را فعال می کنیم و مقادیر شمارنده ها صفر می شوند، دقت کنید که همه متغیرها در یک دستور همزمان صفر شده اند:

```
private void startBtn_Click(object sender, EventArgs e) // دکمه شروع
{
    timer1.Enabled = true;   //فعال سازی زمان سنج
    hour = minute = second = 0; // مقدار گذاری شروع
}
```

۶) برای دکمه «پایان» زمان سنج را غیرفعال(متوقف) می کنیم:

```
private void stopBtn_Click(object sender, EventArgs e)// دکمه پایان
{
    timer1.Enabled = false;   //غیر فعال سازی زمان سنج
}
```

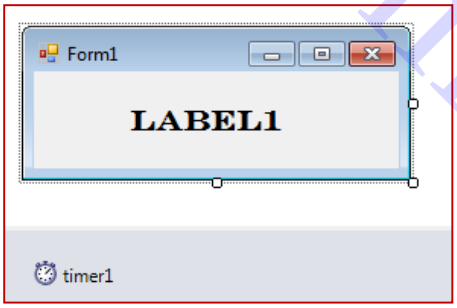
نوشتن رویداد Tick برای زمان سنج :

```
private void timer1_Tick(object sender, EventArgs e)//رویداد زمان سنج
{
    second ++;          // افزایش ثانیه
    if (second == 60) // اگر ثانیه به ۶۰ رسید
    {
        second = 0;
        minute ++;    //افزایش دقیقه
    }
    if (minute == 60) // اگر دقیقه به ۶۰ رسید
    {
        minute = 0;
        hour++;       //افزایش ساعت
    }
    //Format نمایش نتایج به صورت قالب دو رقمی به کمک مشخصه
    label1.Text = string.Format("{0:00}:{1:00}:{2:00}", hour, minute, second);
}
```

چون فاصله زمانی ۱۰۰۰ میلی ثانیه(۱ ثانیه) تنظیم شده است، پس هربار یک واحد به ثانیه اضافه می کنیم. اگر ثانیه به ۶۰ رسید، آنرا صفر کرده و یک واحد به دقیقه اضافه می گردد. اگر دقیقه هم به ۶۰ رسید، متغیر دقیقه را صفر کرده و ساعت یک واحد اضافه می گردد.

تمرین ۱: به برنامه فوق، دو دکمه «مکث» و «ادامه» اضافه کنید که به جای شروع مجدد، از مقدار قبلی ادامه دهد.

تمرین ۲: شمارش معکوس را پیاده سازی کنید. مثلاً از ۱:۲۰:۳۰ (یک ساعت و ۲۰ دقیقه و ۳۰ ثانیه مانده به شروع مسابقه ...)



مثال: به کمک Timer ساعت فعلی سیستم را نمایش دهید.

۱) در پروژه ای جدید، دو کنترل Label و Timer را روی فرم قرار دهید.

۲) برای کنترل Timer، مشخصه های زیر را تنظیم کنید:

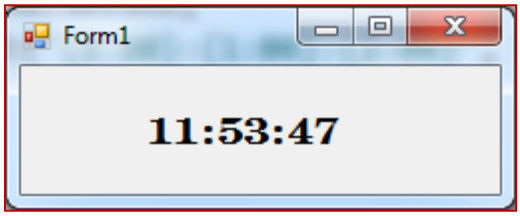
Enabled = true و Interval = 1000

۳) در رویداد Tick از کنترل Timer دستورات زیر را بنویسید

```
private void timer1_Tick(object sender, EventArgs e)
{
    int h = System.DateTime.Now.Hour;
    int m = System.DateTime.Now.Minute;
    int s = System.DateTime.Now.Second;
    string str = string.Format("{0:00}:{1:00}:{2:00}", h, m, s);
    label1.Text = str;
}
```

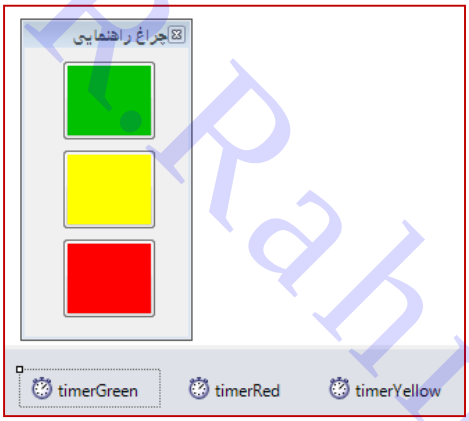
دستور System.DateTime.Now زمان فعلی سیستم را به ما می دهد و مشخصه های Hour,Minute, Second به ترتیب ساعت، دقیقه و ثانیه را د راختیار ما قرار می دهند.

مقادیر بدست آمده را توسط متد Format به شکل *ثانیه:دقیقه:ساعت* با دو رقم تبدیل می کنیم.



چراغ راهنمایی

چراغ راهنمایی با سه چراغ سبز، زرد و قرمز به ترتیب با فواصل زمانی ۱۵، ۳، ۱۰ ثانیه طراحی کنید.



۱) در پروژه ای جدید، سه کنترل button (به نامهای btnGreen، btnYellow و btnRed) و سه کنترل Timer (به نامهای timerGreen، timerYellow و timerRed) را روی فرم قرار دهید.

۲) برای کنترلهای Timer، مشخصه های زیر را تنظیم کنید:

Interval = 1000
Enabled = false

۳) دستورات زیر را در اول برنامه بنویسید:

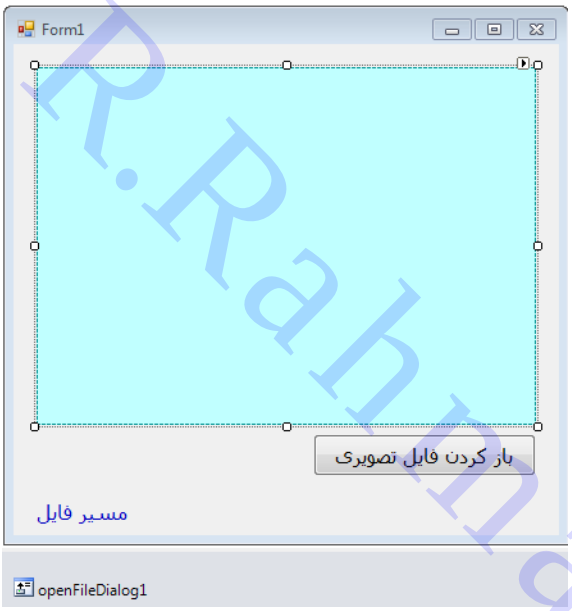
```
byte green, red, yellow;
public Form1()
{
    InitializeComponent();
    green = 15;
    red = 10;
    yellow = 3;
    timerGreen.Enabled = true;
    btnGreen.Visible = true;
}
```

۴) در رویداد Tick از هر کنترل Timer دستورات زیر را بنویسید

```
private void timerGreen_Tick(object sender, EventArgs e)  // چراغ سبز
{
    green--;  // شمارش معکوس
    if (green == 0) {
        timerGreen.Enabled = false; // سبز غیرفعال
        timerYellow.Enabled = true;  // زرد فعال
        btnGreen.Visible = false;  // سبز ناپدید
        btnYellow.Visible = true; // زرد ظاهر
        green = 15;  // مقدار دهی مجدد سبز
    }
    btnGreen.Text = string.Format("{0:00}",green);  // نمایش ثانیه شمار
}

private void timerYellow_Tick(object sender, EventArgs e) //چراغ زرد
{
    yellow--;
    if (yellow == 0)
    {
        timerYellow.Enabled = false;
        timerRed.Enabled = true;
        btnRed.Visible = true;
        btnYellow.Visible = false;
        yellow = 3;
    }
    btnYellow.Text = string.Format("{0:00}", yellow);
}

private void timerRed_Tick(object sender, EventArgs e)  // چراغ قرمز
{
    red--;
    if (red == 0)
    {
        timerRed.Enabled = false;
        timerGreen.Enabled = true;
        btnRed.Visible = false;
        btnGreen.Visible = true;
        red = 10;
    }
    btnRed.Text = string.Format("{0:00}", red);
}
```


طراحی برنامه نمایشگر تصویر با انتخاب تصویر در زمان اجرا

۱) در پروژه جدید، کنترلهای کادر تصویر(PictureBox) و کادر محاوره ای انتخاب فایل (OpenFileDialog)، یک دکمه (Button) و یک برچسب(Label) را روی فرم قرار دهید.



۲) ویژگی SizeMode مربوط به کادر تصویر PictureBox را به StretchImage تغییر دهید.

۳) به کمک کنترل کادر محاوره ای انتخاب فایل، می توان فایلها را از محیط ذخیره سازی مانند هارد انتخاب نمود. ویژگی های مهم آن عبارتند از:

- FileName: نام فایل انتخاب شده
- Filter: نوع فایل هایی را که کادر محاوره ای نشان می دهد، مشخص می نماید. هر بخش آن از دو قسمت شرح و نوع پسوند تشکیل شده که توسط علامت | از هم جدا می شوند.

شرح نوع ۱ | نوع ۱ | شرح نوع ۲ | نوع ۲ |.....

مثال : Jpg Files|*.jpg|jpeg Files|*.jpeg

با تنظیم مقدار مشخصه Filter بصورت زیر، فقط فایلهای تصویری از نوع jpg یا bmp قابل نمایش و انتخاب می باشند.

Filter → Jpg Files|.jpg|bmp Files|*.bmp*

برای استفاده از کادر محاوره ای انتخاب فایل متد ShowDialog آنرا در رویداد کلیک دکمه فراخوانی می کنیم.

```
openFileDialog1.ShowDialog(); // باز کردن کادر محاوره ای انتخاب فایل
```

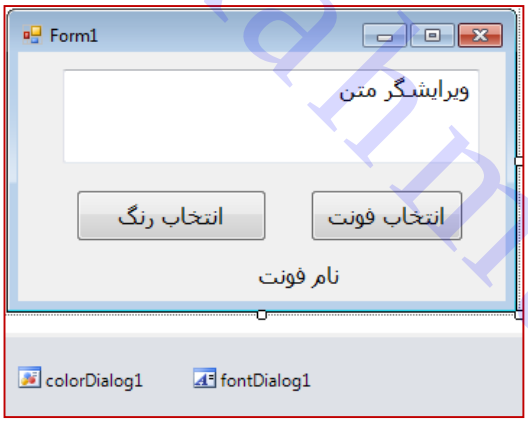
کادر محاوره ای انتخاب فایل دارای رویداد FileOK می باشد که پس از انتخاب فایل مورد نظر اجرا می گردد. پس در این رویداد تصویر انتخاب شده را به کادر تصویر مربوط می کنیم.

```
pictureBox1.ImageLocation = openFileDialog1.FileName; // مربوط کردن تصویر انتخاب شده به کادر تصویر
label1.Text = openFileDialog1.FileName;  // نمایش نام و مسیر فایل در برچسب
```

```
private void openFileDialog1_FileOk(object sender, CancelEventArgs e)
{
    // مربوط کردن تصویر انتخاب شده به کادر تصویر
    pictureBox1.ImageLocation = openFileDialog1.FileName;
    // نمایش نام و مسیر فایل انتخابی در برچسب
    label1.Text = openFileDialog1.FileName;
}

private void button1_Click(object sender, EventArgs e)
{
    // باز کردن کادر محاوره ای انتخاب فایل
    openFileDialog1.ShowDialog();
}
```

مثال: تنظیم فونت و رنگ کادر متن به کمک کادرهای محاوره ای

۱) در پروژه جدید، کنترلهای کادر متن(TextBox) به نام editor، کادر محاوره ای انتخاب فونت (FontDialog)، کادر محاوره ای انتخاب رنگ (ColorDialog) برای رنگ پس زمینه متن، دو دکمه (Button)، یک برچسب(Label) را روی فرم قرار دهید.



۲) خصوصیات Textbox را بصورت زیر تنظیم کنید:

RightToLeft →*Yes* *جهت نوشتن راست به چپ*

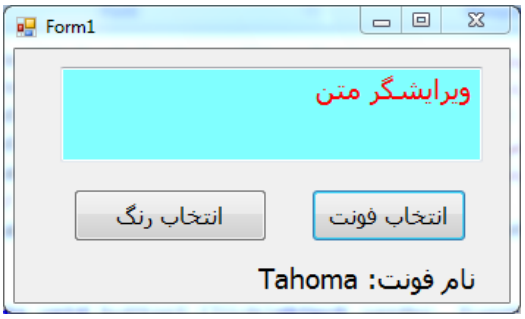
Multiline →*True* *خصوصیت چند خطی*

Name → *editor* *نام کنترل*

۳) خصوصیت ShowColor مربوط به کادر محاوره ای انتخاب فونت را به True مقداردهی کنید تا بتوان رنگ فونت را نیز تغییر داد.

۴) دستورات مربوط به دکمه ها را بصورت زیر بنویسید:

```
private void button1_Click(object sender, EventArgs e) // تنظیم فونت
{
    fontDialog1.Font = editor.Font; // نمایش فونت فعلی
    fontDialog1.ShowDialog();  // بازکردن کادر فونت
    editor.Font = fontDialog1.Font; // تنظیم فونت انتخابی
    editor.ForeColor = fontDialog1.Color; // تنظیم رنگ فونت انتخابی
    label1.Font = fontDialog1.Font;// تنظیم فونت برچسب
    // نمایش نام فونت انتخابی
    label1.Text = "نام فونت: "+fontDialog1.Font.Name;
}

private void button2_Click(object sender, EventArgs e) // تنظیم رنگ
{
    colorDialog1.Color = editor.BackColor; // نمایش رنگ فعلی
    colorDialog1.ShowDialog(); // بازکردن کادر رنگ
    editor.BackColor = colorDialog1.Color;// تنظیم رنگ انتخابی
}
```



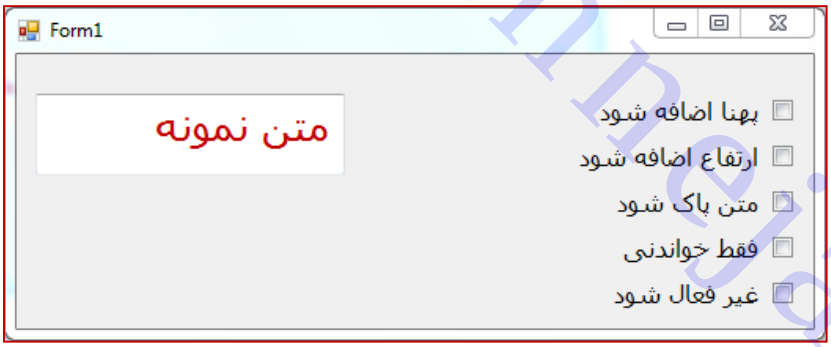
استفاده از کنترل CheckBox

هرگاه لازم باشد دو حالت انتخاب (☑)و عدم انتخاب(☐) را بررسی کنیم از کنترل CheckBox استفاده می شود.

رویداد این کنترل بصورت زیر است: مشخصه Checked اگر درست(true) باشد، انتخاب شده است (☑)

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    if (checkBox1.Checked)
        // اجرای دستورات در صورت انتخاب شدن
    else
        // اجرای دستورات در صورتیکه انتخاب نشود
}
```

مثال: برنامه ای بصورت زیر طراحی کنید که با تیک زدن هر مورد عمل مربوطه انجام گردد.



حل:

۱) ۵ عدد کنترل کادر انتخاب Checkbox و یک کنترل کادر متن TextBox به فرم اضافه کنید.

۲) فونت و رنگ TextBox را به دلخواه انتخاب و سپس مشخصه های زیر را تنظیم نمایید

RightToLeft→ yes
MultiLine→ true

۳) خصوصیت Text هریک از Checkbox ها را مطابق شکل تنظیم کنید.

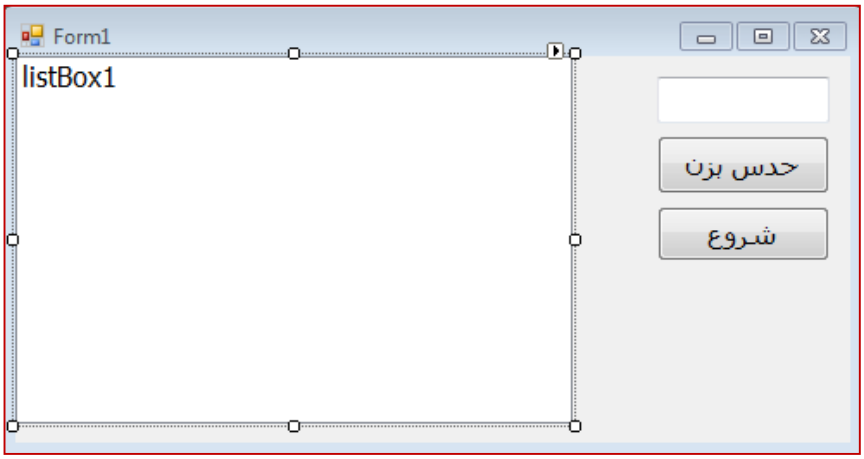
۴) در رویداد CheckedChanged هریک از Checkbox ها دستورات متناظر را بنویسید

```
private void checkBox1_CheckedChanged(object sender, EventArgs e) // پهنای کادر متن
{
    if (checkBox1.Checked) // اگر انتخاب شده است
        textBox1.Width +=  50; // پهنا اضافه شود
    else                       //و گرنه
        textBox1.Width -= 50; // پهنا کم شود
}

private void checkBox2_CheckedChanged(object sender, EventArgs e)// ارتفاع کادر متن
{
    if (checkBox2.Checked)// اگر انتخاب شده است
        textBox1.Height += 50;// ارتفاع اضافه شود
    else
        textBox1.Height -= 50;// ارتفاع کم شود
}

private void checkBox3_CheckedChanged(object sender, EventArgs e)// پاک کردن متن
{
    if (checkBox3.Checked) // اگر انتخاب شده است
        textBox1.Text = ""; // متن پاک شود
    else
        textBox1.Text= "متن نمونه";  // با متن پیش فرض تنظیم شود
}

private void checkBox4_CheckedChanged(object sender, EventArgs e) // فقط خواندنی
{
    if (checkBox4.Checked) // اگر انتخاب شده است
        textBox1.ReadOnly = true; // فقط خواندنی فعال (غیر قابل ویرایش) شود
    else
        textBox1.ReadOnly =false; // از حالت فقط خواندنی خارج شود
}

private void checkBox5_CheckedChanged(object sender, EventArgs e)// غیر فعال کردن
{
    if (checkBox5.Checked)// اگر انتخاب شده است
        textBox1.Enabled = false;// غیر فعال (غیر قابل کلیک و ویرایش) شود
    else
        textBox1.Enabled = true; // فعال شود
}
```

مثال: طراحی بازی حدس عدد. می خواهیم کامپیوتر عددی بصورت تصادفی(بین ۱ تا ۱۰۰) درنظر بگیرد و کاربر عدد را حدس بزند. کامپیوتر فقط به سه صورت جواب می دهد(بزرگتر، کوچکتر یا مساوی). تعداد حدس مجاز ۷ می باشد.

حل:

۱) یک کنترل TextBox، ListBox و دو دکمه (Button) روی فرم قرار دهید.



۲) متغیرهای لازم را در قسمت عمومی تعریف کنید

```
int secretNumber, guesses; // عدد سری کامپیوتر و تعداد حدسها
string msg;  // پیامها
bool found; // آیا عدد پیدا شد
public Form1()
{
    InitializeComponent();
}
```

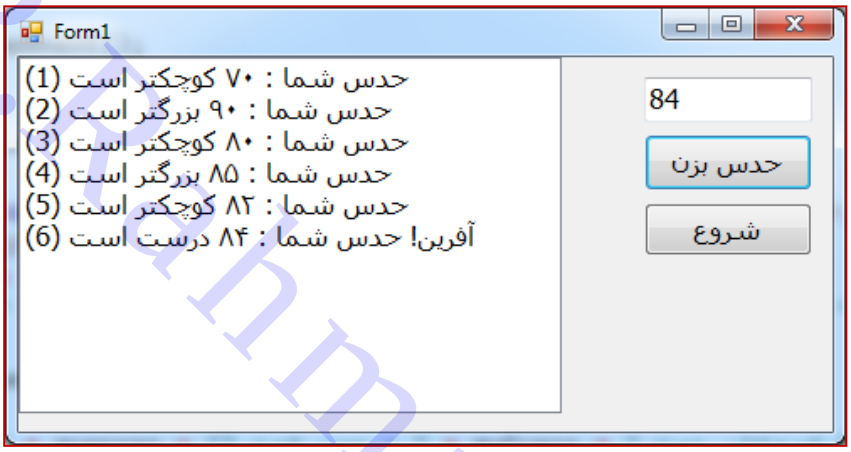
۳) برای تولید عدد تصادفی دستورات زیر را در دکمه شروع بنویسید

```
private void button2_Click(object sender, EventArgs e)  // دکمه شروع
{
    Random r = new Random();  // ایجاد شیئ تولید عدد تصادفی
    secretNumber = r.Next(100) + 1; // تولید عدد تصادفی بین ۱ تا ۱۰۰
    guesses = 0;    // تعداد حدسها
    found = false;
    button1.Enabled= true;  //  فعال کردن دکمه حدس زدن
    listBox1.Items.Clear();  //ListBox پاک کردن محتویات
}
```

۴) در دکمه حدس بزن، دستورات زیر را بنویسید.

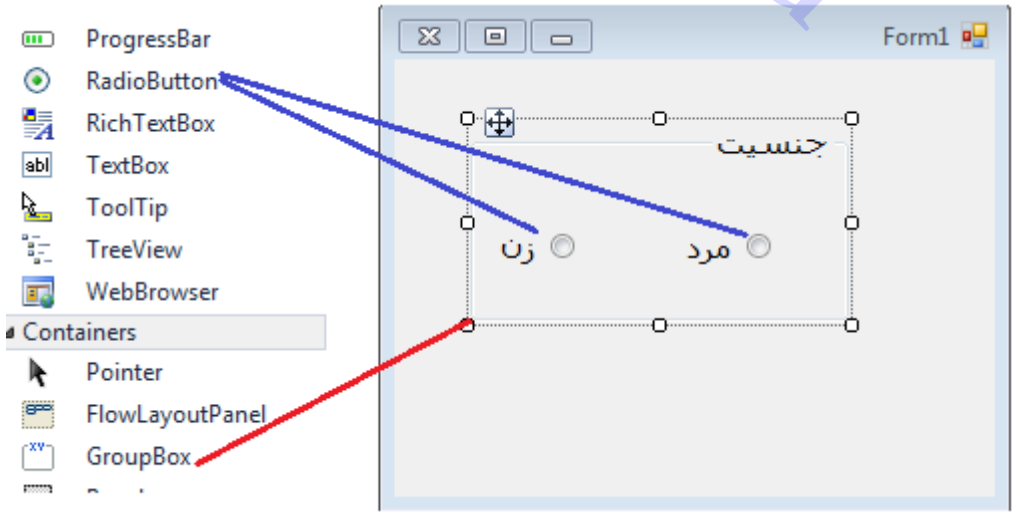
```
private void button1_Click(object sender, EventArgs e)// دکمه حدس بزن
{
    int myGuess =int.Parse(textBox1.Text); // خواندن حدس خود از کادر متن
    guesses++;  // شمارش حدسها
    if (myGuess > secretNumber) { // اگر حدس ما بزرگتر بود
        msg = "(" + guesses + ") حدس شما : " + myGuess + " بزرگتر است ";
    }
    else if (myGuess < secretNumber)
    {
        msg = "(" + guesses + ") حدس شما : " + myGuess + " کوچکتر است ";
    }
    else
    {
        msg = "(" + guesses + ") حدس شما آفرین! : " + myGuess + " درست است ";
        found = true; // عدد پیدا شد
    }
    if ((guesses > 6)&& (!found)) { // اگر تعداد مجاز حدس بیشتر از ۶ باشد
        // و عدد پیدا نشده است
        msg = "(" + guesses + ") شما باختید. عدد مورد نظر  =  " + secretNumber;
        button1.Enabled= false;
    }
    listBox1.Items.Add(msg); // اضافه کردن پیام به کادر لیست متنی
}
```

خروجی بصورت زیر خواهد بود:



دکمه انتخاب رادیویی Radio Buuton

- دکمه انتخاب رادیویی ⊙: هرگاه لازم باشد از بین چند گزینه، فقط یکی را انتخاب کنیم این کنترل بکار می رود.
- در صورتی که بخواهید دو دسته دکمه رادیویی مستقل از یکدیگر، روی فرم داشته باشید، باید هر دسته از دکمه ها را در داخل یک قاب مانند(GroupBox) قرار دهید.
- اگر بخواهید یک دکمه رادیویی بصورت پیش فرض در حالت انتخاب شده باشد، مشخصه Checked آنرا به True تغییر دهید.



داده شمارشی DialogResult

از نوع داده شمارشی DialogResult برای بررسی مقدار برگشتی دکمه فشرده شده در کادر محاوره ای استفاده می شود. مقادیر برگشتی آن طبق جدول زیر است:

جدول ۴ــ۱ــ مقادیر DialogResult

مقدار برگشتی	شرح
None	از کادر محاوره‌ای چیزی برنگشته است.
OK	دکمه ok فشرده شده است.
Cancel	دکمه Cancel فشرده شده است.
Abort	دکمه Abort فشرده شده است.
Retry	دکمه Retry فشرده شده است.
Ignore	دکمه Ignore فشرده شده است.
Yes	دکمه Yes فشرده شده است.
No	دکمه No فشرده شده است.

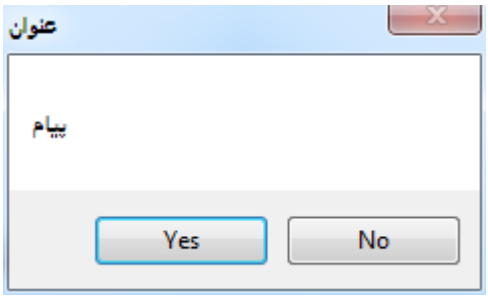
نمایش کادر محاوره ای MessageBox

برای نمایش کادر محاوره ای جهت تأیید یا لغو و موارد مشابه، از MessageBox به صورت زیر استفاده می گردد:

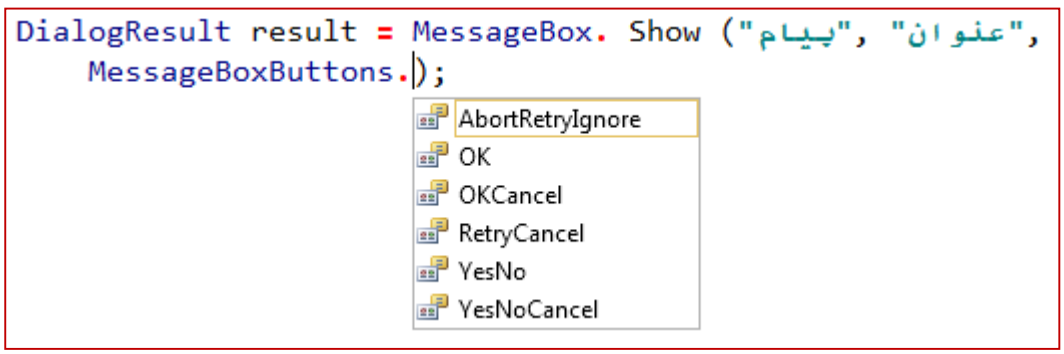
```
DialogResult  result = MessageBox. Show ("پیام" , "عنوان" , دکمه ها);
```

مثال: اگر دستور زیر را مثلاً در رویداد یک دکمه بنویسیم، خروجی حاصل را مشاهده می کنید

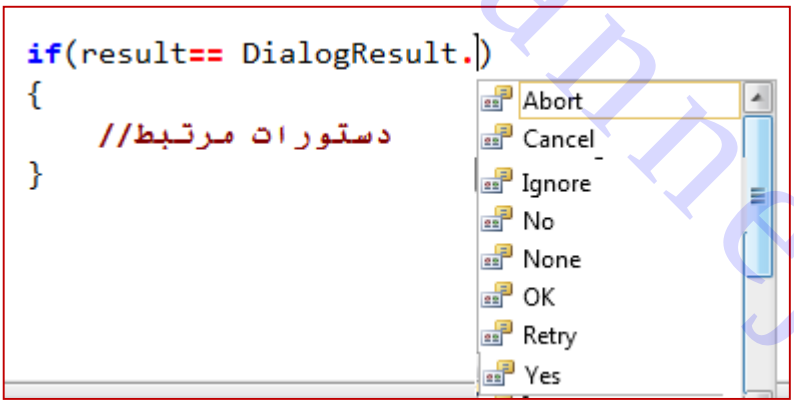
```
DialogResult  result = MessageBox. Show ("پیام", "عنوان",MessageBoxButtons.YesNo);
```



نکته ۱: انتخاب نوع دکمه ها توسط داده شمارشی MessageBoxButtons صورت گرفته و از بین موارد زیر است:



نام اعضا	شرح
AbortRetryIgnore	کادر پیام شامل دکمه‌های Abort, Retry و Ignore است.
OK	کادر پیام شامل دکمه OK است.
OKCancel	کادر پیام شامل دکمه‌های OK و Cancel است.
RetryCancel	کادر پیام شامل دکمه‌های Retry و Cancel است.
YesNo	کادر پیام شامل دکمه‌های Yes و No است.
YesNoCancel	کادر پیام شامل دکمه‌های Yes, No و Cancel است.



نکته ۲: نتیجه کلیک روی هر دکمه را نیز می توان بصورت زیر بررسی نمود و بر اساس هر دکمه کلیک شده، دستورات مرتبط را اجرا نمود.

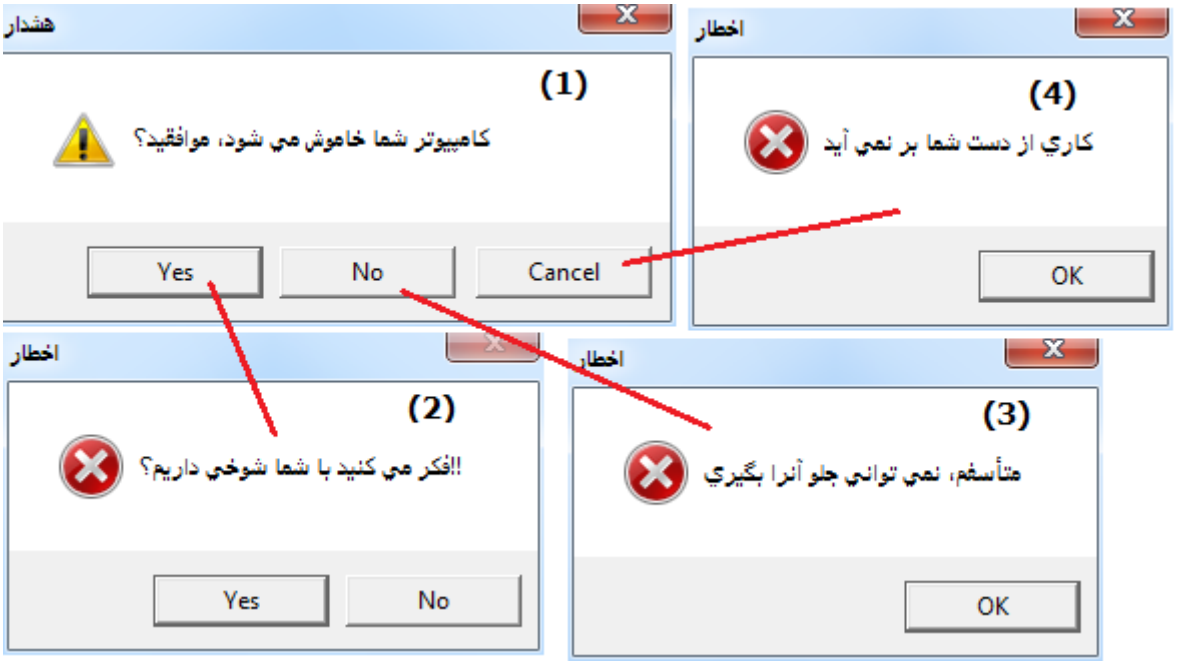
دستور بستن فرم

برای بستن فرم از متد ;Close() بصورت زیر استفاده می شود:

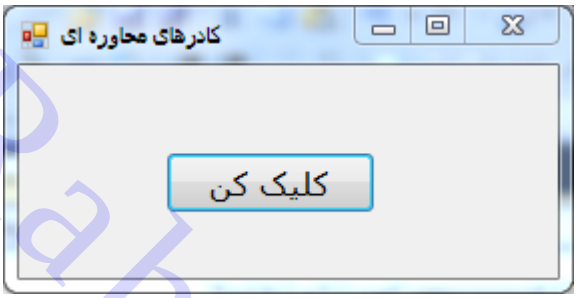
```
this.Close();
```

کلمه کلیدی this به نمونه ای از کلاس Form1 اشاره می کند و منظور کلاسی است که رویداد برای آن نوشته شده است.

تمرین: کادرهای محاوره ای را به ترتیب زیر نشان دهید.



حل: فرمی به همراه یک دکمه طراحی کنید و در رویداد دکمه دستورات زیر را بنویسید



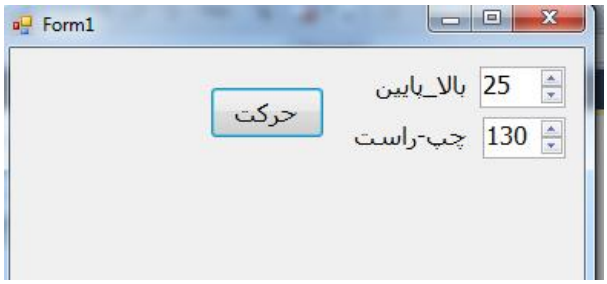

```
private void button1_Click(object sender, EventArgs e)
{
    DialogResult result = MessageBox.Show("کامپیوتر شما خاموش می شود، موافقید؟",
                          "هشدار", MessageBoxButtons.YesNoCancel);
    if (result == DialogResult.Yes) {
        result = MessageBox.Show("فکر می کنید با شما شوخی داریم؟",
                                 "اخطار", MessageBoxButtons.YesNo);
        this.Close();
    }
    else if (result == DialogResult.No)
    {
        result = MessageBox.Show("متأسفم نمی توانید جلو آنرا بگیرید!",
                                 "اخطار", MessageBoxButtons.OK);
        this.Close();
    }
    else if (result == DialogResult.Cancel)
    {
        result = MessageBox.Show("کاری از دست شما بر نمی آید!",
                                 "اخطار", MessageBoxButtons.OK);
        this.Close();
    }
}
```

کنترل عددی افزایشی-کاهشی(NumericUpDown)

برای دریافت داده های عددی در یک محدوده مشخص بکار می رود. مشخصه های مهم آن عبارتند از:

- Maximum: بیشترین مقدار ممکن؛ مقدار پیش فرض ۱۰۰ است
- Minimum: کمترین مقدار ممکن؛ مقدار پیش فرض ۰ می باشد.
- Value : مقدار فعلی
- Increment: میزان افزایش و کاهش در هر بار کلیک

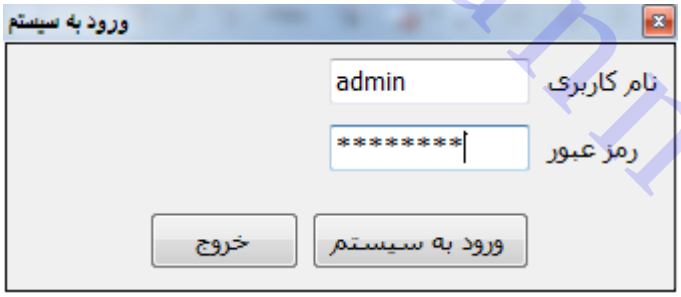
مثال: به کمک دو کنترل NumericUpDown، موقعیت یک دکمه را روی فرم تنظیم کنید



حل: مقدار مشخصه Increment مربوط به کنترلهای NumericUpDown، را به ۱۰ و Maximum را به ۵۰۰ تنظیم نمایید و سپس در رویداد ValueChanged آنها، دستورات زیر را بنویسید

```
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
{

    button1.Top = (int)numericUpDown1.Value;

}
private void numericUpDown2_ValueChanged(object sender, EventArgs e)
{
    button1.Left = (int)numericUpDown2.Value;
}
```

مثال: برنامه ای طراحی کنید که با دریافت نام کاربری و رمز عبور و مقایسه آنها با مقادیر مورد نظر صحت آنرا با کادرهای محاوره ای به کاربر اعلام کند.



حل: نخست در اول برنامه نام و رمز دلخواه خود را تنظیم می کنیم:

```
string uName = "admin";
string uPass = "admin123";
```

```
string uName = "admin";
string uPass = "admin123";
public Form1()
{
    InitializeComponent();
}
private void BtnLogin_Click(object sender, EventArgs e) // دکمه ورود به سیستم
{
    if ((textBox1.Text != uName) || (textBox2.Text != uPass))
    {
        MessageBox.Show("نام کاربری یا رمز عبور اشتباه است!",
            "ورود به سیستم", MessageBoxButtons.OK);
        textBox1.Focus();
    }
    else
    {
        MessageBox.Show("به برنامه خوش آمدید.",
            "خوش آمدید", MessageBoxButtons.OK);
    }
}
private void btnExit_Click(object sender, EventArgs e) // دکمه خروج
{
    this.Close();
}
```

نکته (۱): دستور Focus() مکان نمای ماوس را به روی یک کنترل منتقل می کند.

نکته (۲): برای کنترل تعداد حروف وارد شده مثلاً برای نام کاربری(بین ۵ تا ۳۰ مجاز باشد)، دستور شرطی زیر را می توان بکار برد(مشخصه Length از متن Text تعداد حروف کادر متنی را بر می گرداند):

```
if ((textBox1.Text.Length < 5) || (textBox1.Text.Length > 30))
{
    MessageBox.Show("طول نام کاربری، باید بین ۵ تا ۳۰ حرف باشد!",
                    "ورود به سیستم", MessageBoxButtons.OK);
    textBox1.Focus();
}
```

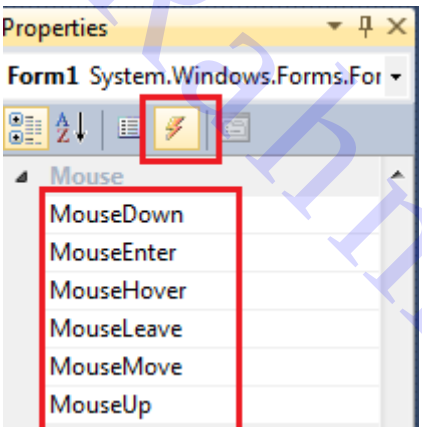
نکته (۳): اگر بخواهید مقدار وارد شده نسبت به حروف کوچک و بزرگ حساس ***نباشد***، می توانید توسط متد ToLower( )، متن وارد شده را به حروف کوچک تبدیل کنید و سپس با مقدار مورد نظر بر حسب حروف کوچک مقایسه نمایید

```
string uName = "admin";
string uPass = "admin123";
public Form1()
{
    InitializeComponent();
}
private void BtnLogin_Click(object sender, EventArgs e) // دکمه ورود به سیستم
{
    if ((textBox1.Text.Length < 5) || (textBox1.Text.Length > 30))
    {
        MessageBox.Show("طول نام کاربری، باید بین ۵ تا ۳۰ حرف باشد!",
                        "ورود به سیستم", MessageBoxButtons.OK);
        textBox1.Focus();
    }

    // بررسی درستی نام کاربری(حساس به متن نباشد) و رمز (حساس به متن) باشد
    else if ((textBox1.Text.ToLower() != uName.ToLower()) || (textBox2.Text != uPass))
    {
        MessageBox.Show("نام کاربری یا رمز عبور اشتباه است!",
            "ورود به سیستم", MessageBoxButtons.OK);
        textBox1.Focus();
    }
    else
    {
        MessageBox.Show("به برنامه خوش آمدید.",
            "خوش آمدید", MessageBoxButtons.OK);
    }

}
private void btnExit_Click(object sender, EventArgs e) // دکمه خروج
{
    this.Close();
}
```

در کد فوق، نام کاربری به حروف کوچک و بزرگ حساس **نیست** و هر جور تایپ شود، درست می باشد. یعنی مقدار admin با AdmiN یکی است. چون حالت تبدیل شده حروف کوچک هر دو در مقایسه شرکت می کنند. اما رمز عبور حساس است و باید عیناً رعایت گردد.

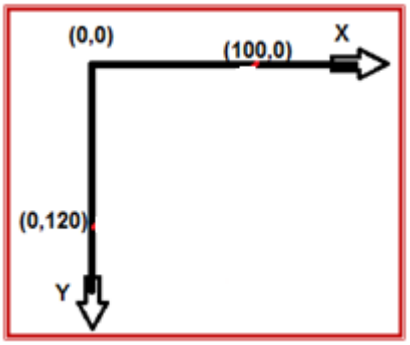
فصل دهم – رویدادهای ماوس و صفحه کلید

رویدادهای ماوس که در قسمت Event قابل مشاهده هستند، طبق جدول زیر می باشند:



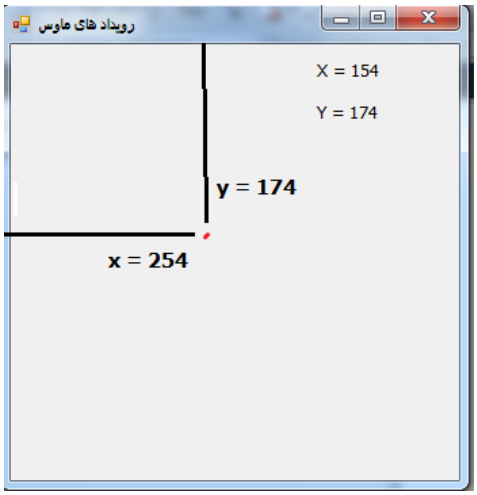
ردیف	نام رویداد	شرح رویداد
۱	MouseClick	کلیک ماوس روی فرم یا کنترل
۲	MouseDoubleClick	دابل کلیک ماوس روی فرم یا کنترل
۳	MouseDown	فشردن دکمه ماوس روی فرم یا کنترل
۴	MouseEnter	ورود از خارج فرم یا کنترل به روی آن
۵	MouseHover	چند لحظه نگه داشتن ماوس روی فرم یا کنترل
۶	MouseLeave	خارج شدن ماوس از روی فرم یا کنترل
۷	MouseMove	حرکت یا تغییر مکان ماوس روی فرم یا کنترل
۸	MouseUp	رها کردن دکمه فشرده شده ماوس

هنگامی که به وسیله ی ماوس بر روی فرم کلیک می کنید رویداد MouseClick و در صورتی که ماوس را حرکت دهید، رویداد MouseMove روی می دهد.



مثال: می خواهیم برنامه ای بنویسیم که با حرکت ماوس بر روی فرم، موقعیت ماوس با دو عدد طول و عرض نشان داده شود.

موقعیت ماوس در روی فرم با دو عدد سنجیده می شود. عدد اول که با حرف X نشان می دهیم، طول است و عبارت است از فاصله ماوس تا سمت چپ فرم وعدد دوم، فاصله ماوس تا بالای فرم را نشان می دهد که به آن عرض نقطه گفته می شود و با حرف Y نشان می دهیم. هنگام حرکت ماوس توسط رویداد MouseMove می توان موقعیت ماوس را در هر لحظه بدست آورده و در کنترل برچسب(Label) نشان دهیم:



روی فرم کلیک کرده و رویداد MouseMove را از لیست رویدادها انتخاب نموده و دستورات زیر را در آن بنویسید:

```
private void Form1_MouseMove(object sender, MouseEventArgs e)
{
    label1.Text = "X = " + e.X;
    label2.Text = "Y = " + e.Y;
    this.Cursor = Cursors.Cross;
}
```

توسط دستور زیر می توان نوع اشاره گر ماوس را به حالت شکل (+)تغییر داد:

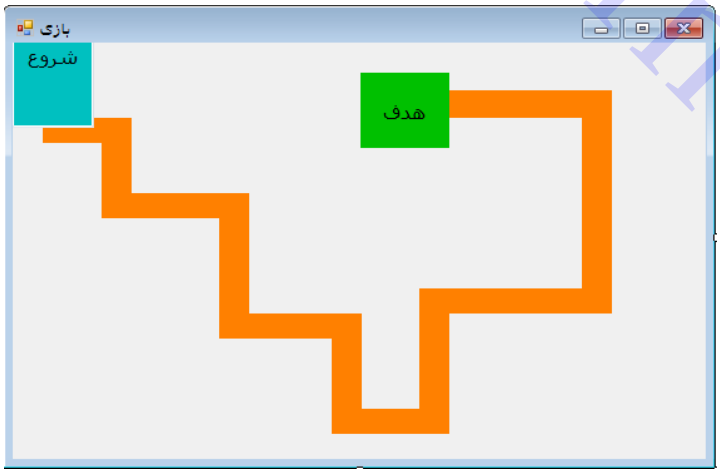
```
this.Cursor = Cursors.Cross;
```

نکته: کنترلها دارای ویژگی Cursor هستند و می توان نوع علامت اشاره گر ماوس را برای هر یک به دلخواه تنظیم نمود.مثلاً در رویداد Form_Load (که به محض بارگذاری فرم روی می دهد)می توان نوع اشاره ماوس روی برجسبها را به حالت عادی و پیش فرض برگرداند.

```
private void Form1_Load(object sender, EventArgs e)
{
    label1.Cursor = Cursors.Default;
    label2.Cursor = Cursors.Default;
}
```

بازی(حرکت روی مسیر) Maze

فرمی به شکل زیر طراحی کنید:



مراحل کار:

۱) یک Button به نام btnStart روی گوشه چپ و بالای فرم قرار دهید.

۲) یک برچسب روی فرم قرار دهید و مشخصات آنرا به صورت زیر تنظیم نمایید.

- رنگ زمینه : نارنجی
- Autosize = False

۳) از برچسب مرحله قبل Copy بگیرید و به تعداد لازم Paste کنید. سپس اندازه موقعیت آنها را تنظیم کنید تا مسیر ساخته شود.

۴) یک برچسب دیگر بسازید و نام آنرا به LblGoal ، رنگ زمینه سبز و عنوان «هدف» را برای آن تنظیم نمایید.

۵) در رویداد Form_Load دستور زیر را بنویسید تا مکان نمای ماوس روی دکمه شروع قرار گیرد:

```
btnStart.Focus();
```

۶) در قسمت شروع برنامه متغیر زیر را تعریف کنید. توسط این متغیر فعال بودن بازی را کنترل می کنیم تا پیام اشتباه بعد از برنده یا بازنده شدن نمایش داده نشود:

```
bool play = false;
```

۷) در رویداد MouseEnter برای کنترل Form دستورات زیر را بنویسید تا اگر از کنترل برچسب مسیر خارج و به فرم وارد شدیم، باخت اعلام شود:

```
if (play)
{
    MessageBox.Show("شما باختید!");
    play = false;
}
```

۸) در رویداد MouseEnter برای کنترل LblGoal دستورات زیر را بنویسید تا برنده شدن با رسیدن به برچسب «هدف» مشخص گردد:

```
if (play)
{
    MessageBox.Show("تبریک، شما برنده شدید.!");
    play = false;
}
```

۹) در رویداد دکمه «شروع» نیز دستور زیر را بنویسید:

```
play = true;
```

۱۰)کد نهایی بصورت زیر است:

```
public partial class Form1 : Form
{
    bool play = false; // متغیر کنترل شروع بازی
    public Form1()
    {
        InitializeComponent();
    }
    private void Form1_MouseEnter(object sender, EventArgs e)
    {
        if (play) // آیا حالت شروع بازی است
        {
            MessageBox.Show("شما باختید!");
            play = false;
        }
    }
    private void Form1_Load(object sender, EventArgs e)
    {
       btnStart.Focus(); // مکان نما روی کنترل دکمه قرار گیرد
    }

    private void lblGoal_MouseEnter(object sender, EventArgs e)
    {
        if (play)// آیا حالت شروع بازی است
        {
            MessageBox.Show("تبریک، شما برنده شدید.!");
            play = false;
        }
    }
    private void button1_Click(object sender, EventArgs e)
    {
        play = true; // حالت شروع بازی
    }
}
```

رویداد های صفحه کلید

هنگامی که کلیدی از صفحه کلید زده می شود، مانند وقتی که ماوس کلیک می شود، رویدادهای مختلفی رخ می دهد. مثلاً در هنگام فشردن و رها کردن کلیدی از صفحه کلید، سه رویداد مختلف KeyPress, KeyUp , KeyDown به ترتیب رخ می دهد که اطلاع می دهد چه کلیدی به وسیله کاربر زده شده است.

در روی صفحه کلید، کلیدهایی وجود دارند که با فشردن آنها علامتی یا کاراکتری روی مانیتور ظاهر می شود، به این کلیدها یا کاراکترها، کاراکترهای چاپ شدنی می گویند. کلید های چاپ شدنی توسط رویداد KeyPress و وِیژگی KeyChar قابل شناسایی هستند. حروف زبانهای غیر انگلیسی هم توسط این رویداد قابل شناسایی می باشند.

نام رویداد	شرح رویداد
KeyPress	فشردن یک کلید چاپ شدنی
KeyDown	فشردن یا پایین رفتن کلید
KeyUp	رها کردن کلید فشرده شده

بقیه کلیدها مانند Ctrl، ALT، Shift، Home، End و نیز F1 تا F12 توسط رویدادهای KeyUp , KeyDown قابل تشخیص هستند.

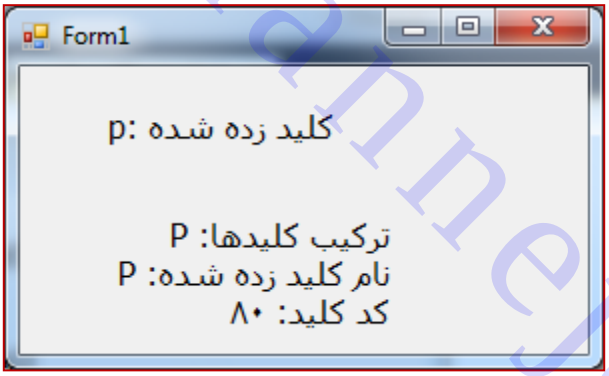
رویدادهای صفحه کلید دارای پارامتر e از نوع KeyPressEventArgs هستند و این پارامتر مشخصه های زیر را در اختیار ما می گذارد:

جدول ۱۲ــ۲ــ شرح ویژگی‌های پارامتر e

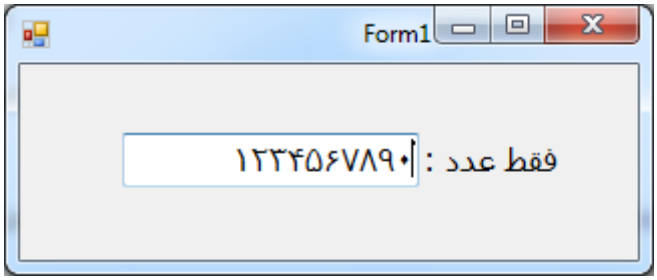
نام ویژگی	شرح ویژگی
Alt	اگر true باشد یعنی کلید Alt زده شده است.
Control	اگر true باشد یعنی کلید Ctrl زده شده است.
Shift	اگر true باشد یعنی کلید Shift زده شده است.
KeyCode	فقط نام کلید زده شده را اطلاع می‌دهد. اما قادر نیست ترکیب کلیدی زده شده را اطلاع دهد.
KeyData	نام کلید یا ترکیب کلیدی زده شده را اطلاع می‌دهد (مانند Ctrl + P ــ اگر بخواهیم بدانیم کلید همراه کدام کلید کنترلی بوده keydata مناسب است).
KeyValue	عدد صحیحی به عنوان کد کلید زده شده، تولید می کند.
SuppressKeyPress	یک متغیر Boolean است. دادن مقدار true به این ویژگی مانع از اجرا و تأثیر کلید روی کنترل می‌شود.

مثال: در یک پروژه جدید روی فرم دو برچسب(Label) قرار دهید و سپس دستورات زیر را در رویداد KeyPress, KeyDown مربوط به Form دستورات زیر را بنویسید:

```
private void Form1_KeyPress(object sender, KeyPressEventArgs e)
{
    label1.Text = " کلید زده شده :" + e.KeyChar;
}

private void Form1_KeyDown(object sender, KeyEventArgs e)
{
    label2.Text = "ترکیب کلیدها: " + e.KeyData+ " \n" +
                  "نام کلید زده شده: " + e.KeyCode + " \n" +
                  "کد کلید: " + e.KeyValue+ " \n" ;
}
```



نکته: دقت کنید که علامت "\n" در ترکیب رشته ها باعث می گردد که مقدار رشته ای بعد از آن در خط جدید قرار گیرد.



مثال: می خواهیم یک کادر متنی طراحی کنیم که فقط بتوان در آن عدد وارد کرد. از طرفی دکمه Delete , Backspace برای پاک کردن کار کند.

حل: یک TextBox روی فرم قرار دهید و نام آنرا به input تغییر دهید. حال در رویداد KeyPress مربوط به کادر متن input دستورات زیر را بنویسید:

```
private void input_KeyDown(object sender, KeyEventArgs e)
{
    // اگر کلید زده شده، رقم، پاک کردن، حذف نباشد
    //از کلید زده شده صرف نظر کن
    if (!(char.IsDigit((Char)e.KeyCode) || // بررسی ورود رقم
        e.KeyCode == Keys.Back ||   //BackSpace بررسی کلید پاک کردن
        e.KeyCode == Keys.Delete))  //Delete بررسی کلید حذف
        e.SuppressKeyPress = true;  // صرف نظر از کلید زده شده
}
```

شرح دستورات:

- شرط زیر بررسی می کند که آیا کلید زده شده رقم است:

(**char**.IsDigit((Char)e.KeyCode)

روال char.IsDigit( ) بررسی رقم بودن کارکتر مورد نظر را انجام می دهد. به نمونه زیر دقت کنید:

char.IsDigit('1') → true یعنی ورودی رقم است

char.IsDigit('a') → false یعنی ورودی رقم نیست

عبارت (Char)e.KeyCode،کلید زده شده را به کارکتر تبدیل می کند.

☜ روال char.IsLetter( ) بررسی حرف بودن کارکتر مورد نظر را انجام می دهد.

- شرط زیر بررسی می کند که آیا کلید زده شده BackSpace (کلید پاک کردن ←) است ؟

e.KeyCode == Keys.Back

- شرط زیر بررسی می کند که آیا کلید زده شده Delete (کلید حذف کردن) است ؟

e.KeyCode == Keys.Delete

- دستور زیر باعث می گردد که از کلید زده شده صرف نظر کند؛ یعنی آنرا نادیده بگیرد.

e.SuppressKeyPress = **true**;

پس دستور شرطی if بررسی می کند که اگر کلید زده شده، رقم، کلید پاک کردن، کلید حذف کردن نباشد(علامت **!** در شرط دستور if)، آن کلید زده شده را نادیده بگیرد.

نکته: اگر بخواهیم کلیدهای جابجایی به راست و چپ (→ ←) روی کادر متن کار کنند و رویداد KeyDown از آن صرف نظر نکند، با عملگر منطقی || شرط زیر را به داخل شرط if اضافه کنید:

تمرین(۱): برنامه ای طراحی کنید که کد ملی و سال تولد را دریافت و بر اساس شرایط زیر درستی آنها را

```
private void input_KeyDown(object sender, KeyEventArgs e)
{
    // اگر کلید زده شده، رقم، پاک کردن، حذف نباشد
    //از کلید زده شده صرف نظر کن
    if (!(char.IsDigit((Char)e.KeyCode) || // بررسی ورود رقم
        e.KeyCode == Keys.Back   ||   //BackSpace بررسی کلید پاک کردن
        e.KeyCode == Keys.Delete ||  //Delete بررسی کلید حذف
        e.KeyCode == Keys.Right  ||  //کلید جابجایی با راست
        e.KeyCode == Keys.Left))   // کلید جابجایی با چپ
        e.SuppressKeyPress = true;  // صرف نظر از کلید زده شده
}
```

بسنجد(راهنمایی: به کمک مثال فوق فقط عدد در دو کادر متن دریافت کنید و سپس به کمک مشخصه Length طول رشته عددی را کنترل نمایید):

- کد ملی باید دقیقاً ۱۰ رقمی باشد
- سال تولد عددی صحیح ۴ رقمی باشد.

تمرین (۲): یک بازی طراحی کنید که سه شیئ از سمت راست فرم وارد شوند و بصورت خودکار به پایین و چپ حرکت کنند. اگر همه آنها از چپ خارج شوند برنده بازی هستید و اگر شیئی به کف فرم برخورد کند، بازنده هستید. با کلیک روی هر شیئ می توانید موقعیت آنرا بالاتر ببرید تا به پایین فرم برخورد نکند.

حل:

۱) سه کنترل Button روی فرم قرار دهید و برای پس زمینه آنها تصویر کوچکی قرار دهید.

۲) سه کنترل Timer با فواصل زمانی(Interval) ۵۰۰،۷۵۰ و ۱۰۰۰ روی فرم قرار دهید. و آنها را فعال نمایید(Enabled=true)

۳) متغیری در اول برنامه برای شمارش تعداد اشیاء رد شده تعریف کنید.

```
public partial class Form1 : Form
{
    int passed = 0; // شمارش تعداد شیئ رد شده
    public Form1()
    {
        InitializeComponent();
    }
```

۴) موقعیت اولیه سه Button را در رویداد Form_Load تنظیم نمایید.

```
private void Form1_Load(object sender, EventArgs e)
{
    button1.Left = Width; //  محل قرارگیری اولیه: فاصله از چپ
    button1.Top = 10;     // محل قرارگیری اولیه: فاصله از بالا
    button2.Left = Width;
    button2.Top = 20;
    button3.Left = Width;
    button3.Top = 20;
}
```

☜مشخصه Width پهنای فرم را مشخص می نماید. پس در اول کار کنترل دکمه به اندازه پهنای فرم از چپ فاصله دارد(در سمت راست مخفی است و بعد شروع به ظاهر شدن می کند)

۵) با هر بار کلیک شیئ دکمه، باید به بالا جابجا شود. پس رویداد آن بصورت زیر است:

```
private void button1_Click(object sender, EventArgs e)
{
    button1.Top -= 20;  // با هر بار کلیک، به بالا حرکت کند
}
```

این رویداد را برای بقیه دکمه ها هم بنویسید و توجه داشته باشید که نام آنها را تنظیم کنید. (مثلاً button1 را به button2 تغییر دهید)

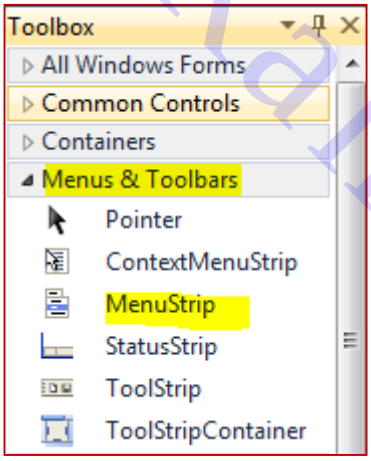
۶) در رویداد Timer_Tick برای همه زمان سنجها(Timer) دستورات مربوطه را بنویسید و متناسب با هر شیئ(دکمه)، نام آنها را تغییر دهید. کد مربوط به زمان سنج Timer1 بصورت زیر است:

```
private void timer1_Tick(object sender, EventArgs e)// زمان سنج شیئ اول
{
    button1.Left -= 10; // حرکت به چپ
    button1.Top += 10;  // حرکت به پایین
    if ((button1.Left > 0) && // اگر از سمت چپ فرم خارج نشده
       (button1.Top >= Height - button1.Height)) // و به لبه پایین فرم رسید
    {
        timer1.Enabled = false;   // توقف
        MessageBox.Show("Game Over!!"); // بازنده بازی
        this.Close();  // بستن برنامه
    }
    if (button1.Left < -button1.Width) // اگر کل شیئ از سمت چپ فرم خارج شد
    {
        passed++;   // شمارش شیئ خارج شده از چپ
        timer1.Enabled = false;  // توقف
    }
    if (passed == 3) // اگر تعداد اشیاء خارج شده به ۳ رسید
    {
        MessageBox.Show("You win!!");  // برنده بازی
        this.Close();  // بستن برنامه
    }
}
```

تمرین (۳): به کمک کلیدهای جهت نما(←↑↓→) و رویداد KeyDown یک شیئ دکمه را در محدوده فضای فرم جابجا کنید.

فصل یازدهم – منو(Menu)

منوها راهی آسان برای دسترسی و اجرای قسمتهای مختلف برنامه هستند و ابزاری قوی برای طراحی یک برنامه با واسط کاربری راحت می باشند.



دسترسی به گزینه های منو به دو صورت زیر است:

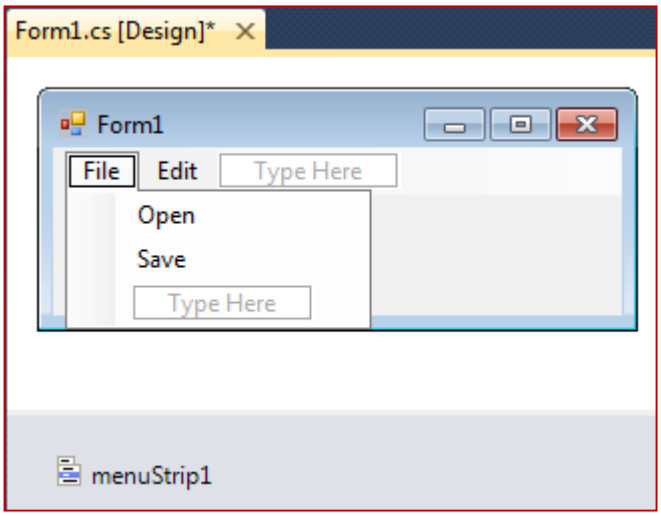
- استفاده از کلیدهای دسترسی سریع(Hot Key) یا میانبر(Shortcut Key)
- انتقال مکان نما به گزینه مورد نظر و کلیک ماوس یا زدن Enter

مراحل افزودن منو به برنامه

۱) یک پروژه جدید باز کنید

۲) در حالی که پنجره فرم نمایان است، در قسمت Menus&Toolbars گزینه MenuStrip را انتخاب کنید و به فرم اضافه نمایید.

۳) با این کار شیئ منو به پایین فرم و کنترل منو به زیر عنوان فرم اضافه می گردد.



۴) با تایپ عنوان منوها در قسمت Type Here زیر عنوان فرم می توانید منوهای خود را بسازید

۵) با کلیک روی هر عنوان در قسمت Properties می توانید تنظیمات هر مورد از منوها را انجام دهید. در جدول زیر خلاصه این ویژگیها را مشاهده می کنید.

۶) برای درج خط جداکننده بین منوها می توانید هنگام طراحی با کلیک روی عنوان منو یک خط تیره (-) در قسمت عنوان قرار دهید.

۷) برای نسبت دادن یک آیکن به منو ، تصویر آیکن مورد نظر را در ویژگی Image تنظیم کنید. آیکن های نمونه را می توانید با انجام جستجوی (*.ico) در پوشه Windows پیدا کنید.

۸) کلید دسترسی(HotKey) را می توان با قرار دادن علامت & قبل از حرف مورد نظر در مشخصه عنوان آن(Text) تعریف کنید. در هنگام اجرای برنامه، با فشردن کلید Alt و آن حرف خاص، منو انتخاب می گردد.

جدول ۱ــ۳ــ ویژگی‌های متداول شیء منو

ویژگی	شرح
Cheked	تیک دار کردن گزینه منو
Enabled	فعال یا غیرفعال کردن گزینه منو
Image	قرار دادن تصویر برای گزینه منو
Name	نام مربوط به منو یا گزینه‌های منو در کد نویسی
RightToLeft	راست به چپ کردن گزینه‌های منو (مانند زبان فارسی)
ShortCutKeys	تعریف کلید میانبر
Text	گزینه‌های منو
Visible	نمایش گزینه‌های منو

مثلاً اگر در مشخصه Text منوی Edit قبل از حرف E علامت & را قرار دهیم (&Edit)، هنگام اجرا بصورت Edit در می آید.

۹) برای تعریف کلید میانبر، در قسمت ShortcutKeys ، ترکیب کلید مورد نظر(Alt, Ctrl,Shift به همراه یک حرف الفبا) را مشخص نمود.

۱۰) برای نوشتن دستورات متناظر هر منو مانند کنترلهای دیگر در قسمت Event ها رویداد Click آنرا تعریف کنید.

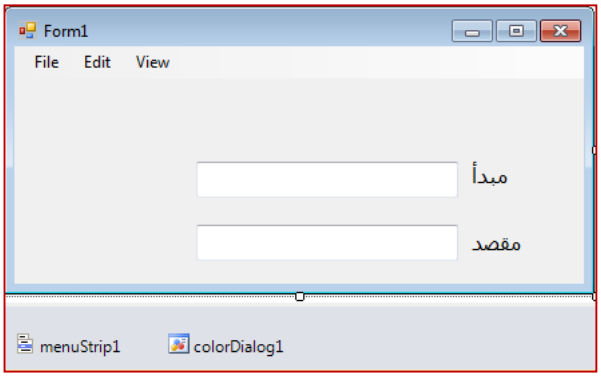
تمرین: منویی به شکل زیر ایجاد کنید. و در فرمان هر یک دستوراتی بنویسید که رنگ پس زمینه و پیش زمینه یک Label روی فرم تغییر کند.

	عنوان منو	کلید میانبر
Backcolor		
رنگ پس زمینه	Black	Ctrl+K
	White	Ctrl+W
	Yellow	Ctrl+Y
	----------	خط جداساز
	Gray	Ctrl+D
ForeColor		
رنگ پیش زمینه	Green	Ctrl+G
	Red	Ctrl+R
	Blue	Ctrl+B

مثال: در یک پروژه جدید، ساختار منوهای زیر را بسازید. کنترل ColorDialog و دو کادر متنی TextBox و دو برچسب اضافه کنید.

منوها:

- File→New , Exit
- Edit → Copy , Paste
- View → BackColor, ForeColor



۱) در قسمت سراسری متغیر زیر را تعریف کنید:

```
string str;
```

۲) کد مربوط به منوها را مطابق زیر بنویسید

```
private void newToolStripMenuItem_Click(object sender, EventArgs e) //new
{
    textBox1.Text = "";
    textBox2.Text = "";
}

private void exitToolStripMenuItem_Click(object sender, EventArgs e)//exit
{
    this.Close();
}

private void copyToolStripMenuItem_Click(object sender, EventArgs e)//copy
{
    if (textBox1.SelectionLength == 0) // بررسی اینکه متن انتخاب شده است یا نه
    {
        MessageBox.Show("متن مبدأ انتخاب نشده است!");
        str = "";
    }
    else
        str = textBox1.SelectedText;
}

private void pasteToolStripMenuItem_Click(object sender, EventArgs e)//paste
{
    textBox2.Text = str;
}
private void backColorToolStripMenuItem_Click(object sender, EventArgs e)//backcolor
{
    colorDialog1.ShowDialog();
    textBox1.BackColor = colorDialog1.Color;
    textBox2.BackColor = colorDialog1.Color;
}

private void foreColorToolStripMenuItem_Click(object sender, EventArgs e)//forecolor
{
    colorDialog1.ShowDialog();
    textBox1.ForeColor = colorDialog1.Color;
    textBox2.ForeColor = colorDialog1.Color;
}
```

نکته: مشخصه SelectionLength از کادر متن، تعداد کارکتر انتخاب شده را بر می گرداند.

منوی زمینه(Context Menu)

این منوها که با کلیک راست بر روی زمینه پدید می آیند، منوی زمینه یا میانبر نام دارند. این منوها در هر موقعیت مکانی از صفحه، ممکن است دارای گزینه های متفاوتی باشند. برای ایجاد آن از قسمت ابزارها گزینه زیر را انتخاب نمایید:

Menu & Tool Bars → ContextMenuStrip

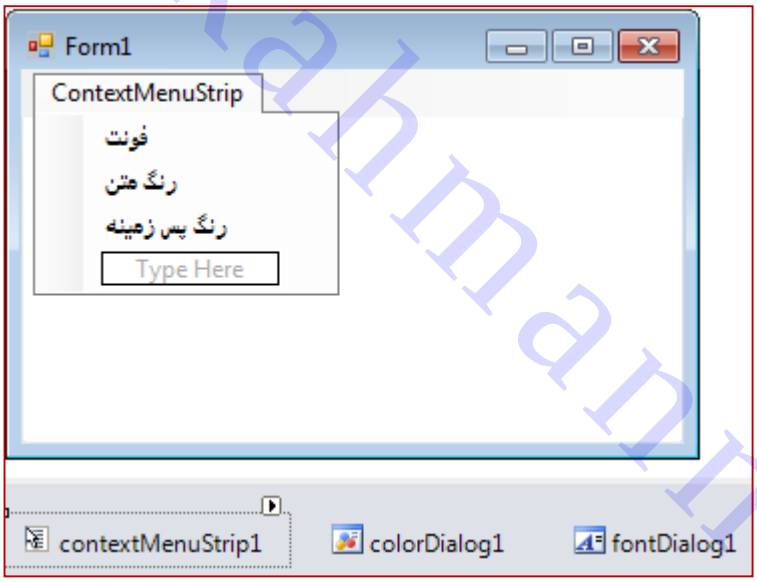
مثال: یک کادر متنی به فرم اضافه کنید و به کمک منوی زمینه ، فونت، رنگ زمینه و رنگ پس زمینه را تغییر دهید.

۱) یک کادر متنی به نام Editor به فرم اضافه کنید و خصوصیت MultiLine آنرا به True و حالت Dock آنرا به وسط چین(Fill) تنظیم نمایید تا کل فرم را بپوشاند.

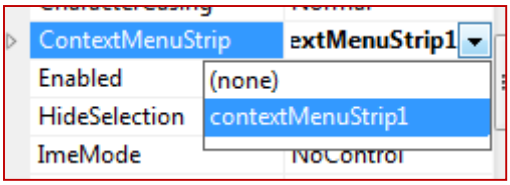
۲) از قسمت Dialog ابزارها، FontDialog, ColorDialog را اضافه نمایید.

۳) یک کنترل ContextMenuStrip اضافه نمایید. و گزینه های آنرا به صورت زیر تنظیم نمایید

- fontMenu با عنوان فونت
- colorMenu با عنوان رنگ متن
- backColorMenu با عنوان رنگ زمینه



۴) برای اینکه با کلیک راست روی کادر متنی editor منوی طراحی شده ما ظاهر شود، باید ویژگی ContextMenuStrip مربوط به کنترل editor را به نام منوی اصلی خود تنظیم کنیم.



۵) دستورات هر یک از منوها را به شکل زیر تنظیم نمایید.

```
private void fontMenu_Click(object sender, EventArgs e)//نوع فونت
{
    fontDialog1.ShowDialog();
    editor.Font = fontDialog1.Font;
}

private void backColorMenu_Click(object sender, EventArgs e)//رنگ پس زمینه
{
    colorDialog1.ShowDialog();
    editor.BackColor = colorDialog1.Color;
}

private void colorMenu_Click(object sender, EventArgs e)//رنگ فونت
{
    colorDialog1.ShowDialog();
    editor.ForeColor = colorDialog1.Color;
}
```

فصل دوازدهم – شیئ(Object) و کلاس(Class)

کلاسها، نقشه ایجاد و قالب اشیاء را مشخص می‌کنند(مثل نقشه یک ساختمان) و اشیاء نمونه های واقعی ساخته شده از کلاسها هستند(مثل ساختمان های ساخته شده طبق یک نقشه).

نکات مهم :

- هر شیئ نمونه ای از یک کلاس است. مثلاً شیئ button1 از روی کلاس Button ساخته می شود.
- هر شیئ دارای ویژگی هایی است که آنرا توصیف می کند. مثلاً شیئ button1 دارای ویژگی Text می باشد که متن عنوان آنرا تعیین می نماید.
- هر شیئ عملیات یا رفتاری را از خود بروز می دهد. این عملیات توسط متدها مشخص می شوند. مثلاً شیئ button1 دارای متد ResetText( ) است که متن عنوان آنرا پاک می کند.
- هرشیئ دارای یک هویت مستقل است و تنها چیزی که لازم داریم این است که بدانیم این شیء دارای چه ویژگی ها و متدهایی است وهرکدام چه کاربردی دارند.

اشیاء چگونه ساخته می شوند؟

شیء بر اساس کلاس ساخته می شود. در واقع شیء، نمونه ساخته شده بر اساس نقشه و مدلی است که کلاس مشخص می کند. ایجاد شیئ به معنای اختصاص حافظه به آن است. دستور ایجاد یک شیء جدید از کلاس در حالت کلی به صورت زیر است:

```
new نام کلاس ( )
```

مثلاً کلاس Form1 در برنامه که از روی کلاس Form تعریف شده است:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
```

و در فایل برنامه Program.cs ایجاد شیئ Form1 را می بینیم:

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form1());
}
```

همچنین در ایجاد یک شیئ دکمه داریم:

```
private System.Windows.Forms.Button button1;
```
تعیین نوع شیء button1

```
this.button1 = new System.Windows.Forms.Button();
```
ایجاد شیء button1

مثال: در یک پروژه فرم و یک کنترل Button قرار دهید و به قسمت کد برنامه در فایل Form1.cs رفته و در متد مقداردهی اولیه فرم InitializeComponent()، مقدار Text دکمه را تغییر دهید:

button1.Text = "دکمه" ;

در رویداد کلیک دکمه دستور زیر را بنویسید:

button1.ResetText ( );

```
public Form1()
{
    InitializeComponent();
    button1.Text = "دکمه";
}

private void button1_Click(object sender, EventArgs e)
{
    button1.ResetText();
}
```

متد ResetText ( ) مقدار داخل مشخصه Text را پاک می کند. ما از جزئیات دستورات داخل این متد و نحوه کار آن بی اطلاع هستیم. یعنی عملکرد داخلی آن از دید ما پنهان است و این یکی از ویژگیهای شیئ گرایی است.

شیئ گرایی

تعریف و به کارگیری متد

خصوصیات و وضعیت یک شیء به وسیله فیلدها و رفتارهای اشیاء در قالب متدها تعریف می گردند. بنابراین محل و مکان تعریف فیلدها و متدهای یک شیء در داخل یک کلاس است:

متد چیست؟

متد مجموعه ای از دستورات است که عمل خاصی را انجام می دهد. هر متد می تواند تعدادی ورودی داشته باشد و حداکثر یک مقدار برگشتی یاخروجی نیز داشته باشد. برای آنکه یک متد را ایجاد کنیم آن را داخل یک کلاس می نویسیم. نحوهٔ نوشتن یک متد به صورت زیر است:

- ورودی ها، پارامترهای متد را تعیین می کنند که بر اساس آنها عملیات خاصی انجام می شود. قالب معرفی ورودی ها در واقع مانند تعریف متغیرهاست که با کاما جدا می شوند:

```
توصیف‌کننده  نوع خروجی  نام متد (ورودی‌ها)
{
   دستورات
}
```

تعداد ورودی ها بستگی به شرایط دارد . گاهی ممکن است یک متد نیازی به ورودی هم نداشته باشد.

```
نوع ورودی ۱  نام ورودی ۱ , نوع ورودی ۲  نام ورودی ۲ , ....
```

- نوع خروجی: تعیین می کند که نوع برگشتی و نتیجه پردازش متد از چه نوع داده ای است.
- توصیف کننده: نحوه دسترسی را مشخص می کنند که بعداً آنها را توضیح خواهیم داد.

مثال: متدی تعریف کنید که نام و نام خانوادگی را دریافت کرده و با یک فاصله خالی آنها را کنار هم قرار دهد.

- کلمه private توصیف کننده است

```
private string GetFullName(string fname, string lname)
{
    return fname + "  " + lname;
}
```

- string اول نوع خروجی و برگشتی را مشخص می کند
- ورودی ها عبارتند از fname, lname که از نوع string می باشند.
- دستورات متد آنها را با فاصله کنار هم قرار می دهد و توسط دستور return برگشت می دهد.

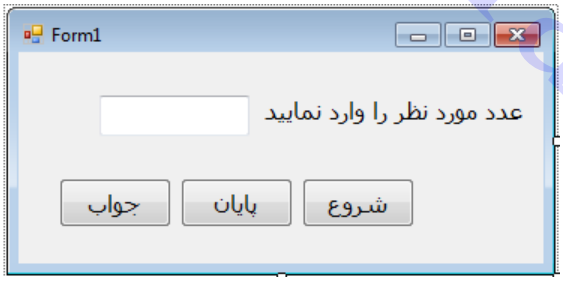
برای استفاده از یک متد باید آنرا فراخوانی نمود:

۱) نخست متغیری از نوع برگشتی متد تعریف کنید
۲) سپس نام متد به همراه مقادیر ورودی را بنویسید

```
string myname = GetFullName("Saman","Ahmadi");
```

با اجرای این دستور مقدار "Saman Ahmadi" در متغیر myname قرار می گیرد.

```
private string GetFullName(string fname, string lname)
{
    return fname + "   " + lname;
}

private void button1_Click(object sender, EventArgs e) // دکمه
{
    string myname = GetFullName("Saman","Ahmadi");
    MessageBox.Show(myname);
}
```

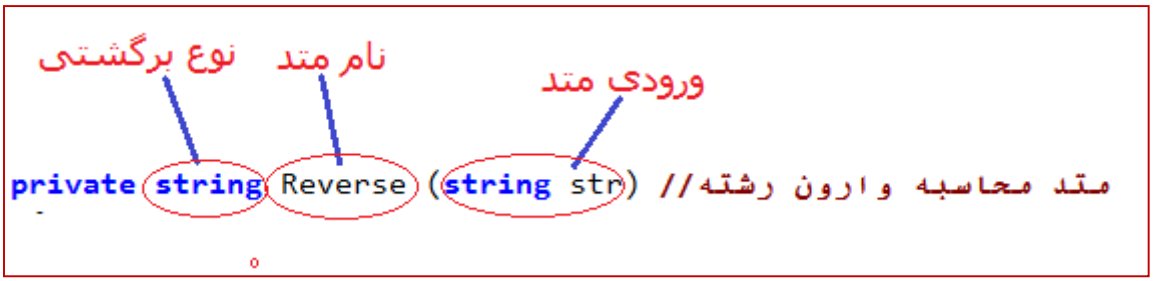


مثال: می خواهیم یک بازی دو نفره طراحی کنیم که نفر اول یک عدد چند رقمی وارد کند و دکمه شروع را بزند و بازیکن دوم اقدام به نوشتن همان رقم به صورت بر عکس نماید. با زدن دکمه پایان از طرف بازیکن دوم بازی به اتمام رسیده و نتیجه اعلام شود.

دقت کنید که نام دکمه ها را به ترتیب تنظیم کنید: شروع(start)، پایان(finish) ، جواب(answer) و در آغاز کار دکمه های answer , finish غیر فعال شوند(Enabled = false). نام textBox را به input تغییر دهید.

نخست باید یک متد طراحی کنیم که متن داخل کادر متنی را برعکس نماید. در داخل کلاس Form1، متد را تعریف می کنیم. برای بدست آوردن وارون رشته، در یک حلقه تکرار for ، حرف به حرف از آخر به اول رشته ورودی انتخاب کرده و به متغیر رشته نتیجه می چسبانیم. متد دارای یک ورودی است. (string str) و نوع برگشتی نیز string است.

پس کد تعریف متد به صورت زیر خواهد بود:



برای ذخیره متن اصلی قبل از پاک شدن، یک متغیر سراسری در شروع کلاس Form1 تعریف می کنیم:

```
public partial class Form1 : Form
{
    string goal; // رشته اصلی
    public Form1()
    {
        InitializeComponent();
    }
```

```
private string Reverse (string str) //متد محاسبه وارون رشته
{
    string result = ""; // نتیجه
     // حرف به حرف از آخر به اول انتخاب کرده
    for (int c = str.Length-1; c >= 0; c--)
    {
        result += str[c];// حرف را به رشته نتیجه می چسبانیم
    }
    return result; // برگشت نتیجه
}
```

رویداد کلیک دکمه ها و کنترل ورود عدد را بصورت زیر می نویسیم:

```
private void start_Click(object sender, EventArgs e)// شروع
{
    finish.Enabled = true;
    answer.Enabled = true;
    start.Enabled = false;
    goal = input.Text;
    input.Text = "";
}

private void answer_Click(object sender, EventArgs e) // جواب
{
    MessageBox.Show(" متن وارد شده برابر با " + goal +
        " و عکس آن برابر با " + Reverse(goal) + " می باشد");
}

private void finish_Click(object sender, EventArgs e)// پایان
{
    finish.Enabled = false;
    start.Enabled = true;
    if (Reverse(goal) == input.Text)
        MessageBox.Show("آفرین! جواب درست را وارد کردید");
    else
        MessageBox.Show("متأسفانه جواب شما درست نبود! ");
}
```

- شروع: متن اصلی را به متغیر goal انتقال داده و کادر متنی را پاک می کنیم. و دکمه پایان و جواب فعال می گردد.
- پایان: مقایسه مقدار وارد شده با وارون متن اصلی توسط متد ()Reverse که قبلاً تعریف کرده ایم. دکمه شروع فعال و پایان غیر فعال می گردد.
- رویداد KeyDown کادر متنی: باید فقط عدد، Backspace, delete را قبول نماید و از بقیه صرف نظر کند.
- جواب: متن اصلی و وارون آنرا در یک کادر پیام نمایش می دهیم.

```
private void input_KeyDown(object sender, KeyEventArgs e)
{
    // اگر کلید زده شده، رقم، پاک کردن، حذف نباشد،
    //از کلید زده شده صرف نظر کن
    if (!(char.IsNumber((Char)e.KeyCode) || // بررسی ورود رقم
        e.KeyCode == Keys.Back ||   //BackSpace بررسی کلید پاک کردن
        e.KeyCode == Keys.Delete))  //Delete بررسی کلید حذف
        e.SuppressKeyPress = true;  // صرف نظر از کلید زده شده
}
```

از مزایای متد آن است که ما یک بار آنرا تعریف می کنیم و بارها آنرا می توانیم استفاده کنیم.

استفاده از کلاس و شیئ(مثال ساعت)

می خواهیم ساعت را به عنوان شیئ در نظر گرفته و ویژگیها و رفتار آنرا مدل کنیم.

- ویژگیها: ساعت، دقیقه و ثانیه، رنگ، قیمت، مدل، نوع(دیجیتال یا عقربه ای) و ... که فقط مقدار ساعت، دقیقه، ثانیه را مدل می کنیم.
- رفتار(عملیات): نمایش مقدار ساعت، تنظیم ساعت و ...

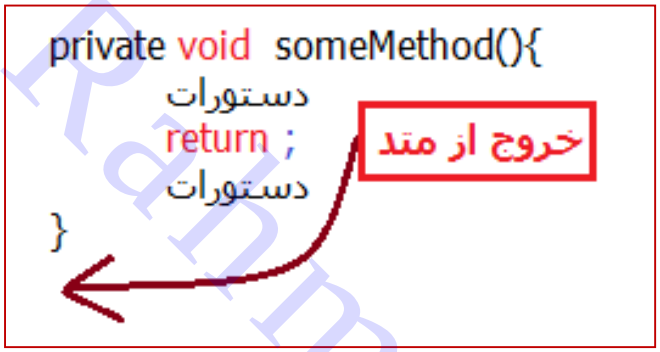
پیاده سازی:

باید یک کلاس تعریف کنیم که ساعت را مدل نماییم. تعریف کلاس معمولاً بعد از کلاس Form انجام می شود:

```
class Clock { // شروع تعریف کلاس
    byte hour, minute, second; // ها (فیلد) ویژگی ها

    // متد تنظیم ساعت با مقادیر داده شده
    public void SetClock(byte h, byte m, byte s)
    {
        hour = h;
        minute = m;
        second = s;
    }

    // متد نمایش ساعت در قالب ثانیه: قیقه: ساعت
    public void ShowClock()
    {
        MessageBox.Show(hour.ToString() + " : " + minute.ToString()+
         " : " + second.ToString());
    }
} // پایان تعریف کلاس
```

توضیح: نوع برگشتی void برای متد ShowClock( ) و SetClock( ) ، یعنی متد مقداری برگشت نمی دهد و دستور return لازم نیست.

نکته: اگر متدی برگشتی نداشته باشد می توان دستور return را بدون هیچ مقداری نوشت و تنها یک علامت ; در انتهای آن قرار داد که به منزله پایان کار متد و صرف نظر از دستورات بعد از آن می باشد.



نحوه استفاده از کلاس

برای استفاده از کلاس باید در دستورات برنامه،مراحل زیر را انجام داد:

- شیئی از نوع کلاس معرفی کنیم. این مرحله مانند تعریف متغیر است:

```
نام کلاس  نام شیئ;
Clock  myClock;
```

- به شیئ تعریف شده، حافظه اختصاص داد که توسط دستور new انجام می شود:

```
نام شیئ = new نام کلاس ( );
myClock = new Clock( );
```

می توان دو مرحله فوق را با هم ادغام نمود:

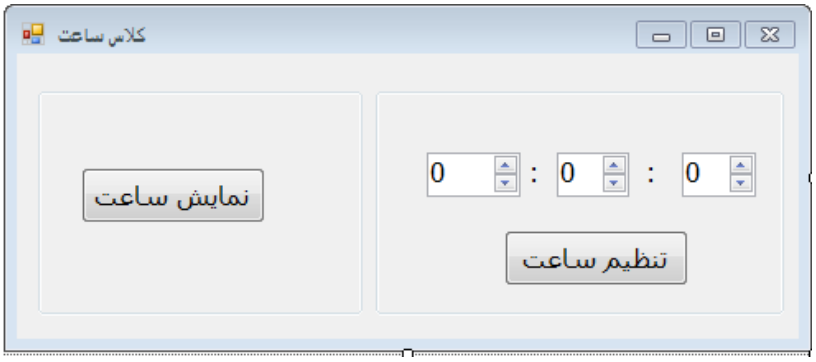
```
نام کلاس  نام شیئ = new نام کلاس ( );
Clock  myClock = new Clock( );
```

- دسترسی به ویژگی ها و متدهای شیئ: پس از ایجاد شیء می توانید به فیلدها و متدهای شیء دسترسی داشته باشید و آنها را فراخوانی کنید. برای دسترسی به اجزا و اعضای یک شیء، کافی است نام شیء و سپس نام عضو (فیلد یا نام متد) را به دنبال آن ذکر کنیم که با علامت نقطه از یکدیگر جدا می شوند.

```
نام شیئ•نام عضو
myClock•ShowClock( );
```

مثال عملی:

با فرض نوشتن کلاس Clock طبق الگوی فوق، فرمی به صورت زیر طراحی کنید:



- دو کادر گروه بندی (groupBox)
- دکمه تنظیم ساعت(با نام setClock)، دکمه نمایش ساعت(با نام showClock)

- کادر افزایشی-کاهشی ساعت(با نام hourBox و مقدار min:0, max:23)، دقیقه (با نام minuteBox و مقدار min:0, max:59) و ثانیه(با نام secondBox و مقدار min:0, max:59)
- دستور زیر در اول برنامه Form1 برای ایجاد شیئ ساعت:

```
public partial class Form1 : Form
{
    Clock myClock = new Clock(); //Clock ایجاد شیئ از کلاس ساعت
    public Form1()
    {
        InitializeComponent();
    }
```

- دستور دکمه تنظیم ساعت:

```
private void setClock_Click(object sender, EventArgs e) // تنظیم ساعت
{
    byte h = (byte)hourBox.Value; // مقدار ساعت
    byte m = (byte)minuteBox.Value; // مقدار دقیقه
    byte s = (byte)secondBox.Value; // مقدار ثانیه

    myClock.SetClock(h,m,s); // تنظیم ساعت با فراخوانی متد کلاس
}
```

دستورات دکمه نمایش ساعت:

```
        private void showClock_Click(object sender, EventArgs e)// نمایش ساعت
        {
            myClock.ShowClock();// نمایش ساعت با فراخوانی متد کلاس
        }
    }
```

100 %

Error List

1 Error | 0 Warnings | 0 Messages

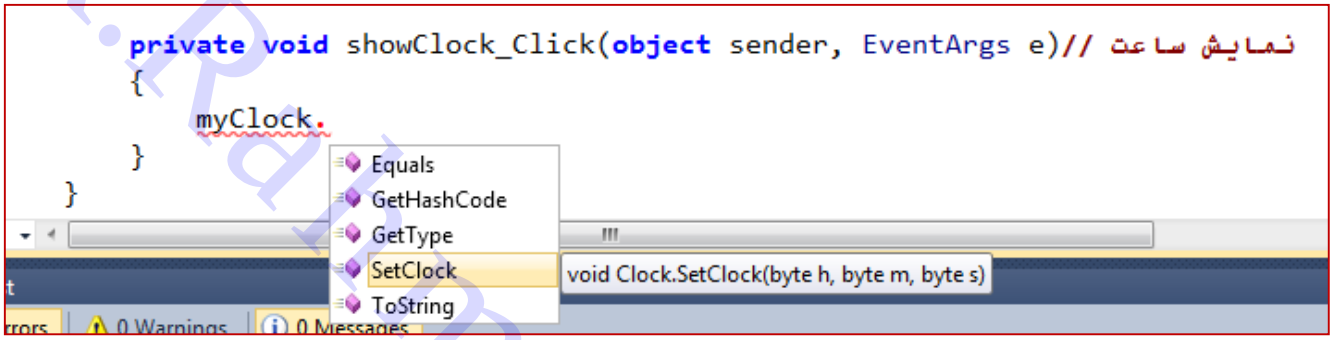
	Description	File	Li...	Column	Project
1	'ClockClass.Clock.ShowClock()' is inaccessible due to its protection level	Form1.cs	36	21	ClockClass

توصیف کننده ها

اگر در کلاس Clock توصیف کننده های متد ShowClock( ) و SetClock( ) را به جای public به private تغییر دهیم دیگر قابل دسترسی نخواهند بود:

پیغام 'ClockClass.Clock.ShowClock( ) is inaccessible due to its protection level' به این معناست که متد ShowClock( ) به دلیل سطوح حفاظتی، قابل دسترسی نیست!

اگر بعد از نوشتن شیئ myClock علامت نقطه بگذاریم، در لیست هوشمند پیشنهادی آن، نام متد ShowClock( ) دیده نمی شود:



شرح و کاربرد توصیف کننده ها طبق جدول زیر است:

توصیف کننده	توضیح	مثال
private	دسترسی از بیرون به متغیر یا متدی که با این توصیف‌کننده مشخص شده، وجود ندارد و در واقع فقط داخل کلاس (شیء) قابل استفاده است. اگر متد یا متغیری توصیف کننده نداشته باشد به‌صورت پیش فرض private خواهد بود.	مانند متغیرهای hour، minute و second که private هستند و می‌توان توسط دستورات داخلی کلاس Clock به آنها دسترسی داشت اما نمی‌توان از بیرون به این متغیرها دست پیدا کرد.
public	به یک متد یا متغیر از شیءای که در کلاس به‌صورت public تعریف شده است می‌توان از بیرون کلاس، دسترسی داشت.	مانند متدهای ShowClock و SetClock در کلاس Clock

مثال: پیاده سازی جنگ سفینه های فضایی(SpaceShip)

- ویژگیها: سرعت(speed)، سپردفاعی(shield) و اندازه سوخت(fuel) را در نظر می گیریم. که دارای مقدار بین ۰ تا ۱۰۰ برحسب درصد می باشند.
- رفتار(عملیات): نمایش وضعیت، تنظیم مقادیر ویژگیها و ...

برای مدل کردن رفتار سفینه در تنظیم مقادیر ویژگیها، سه روش داریم:

۱) استفاده از دستور انتساب و دسترسی به فیلدهای شیء
۲) استفاده از یک متد برای تغییر دادن مقدار فیلدها
۳) استفاده از ویژگی Property که روش معمول استاندارد C# می باشد.

الف) پیاده سازی کلاس SpaceShip

در یک پروژه جدید ویندوزی(Windows Form Application) که به اختصار WFA می خوانیم، بعد از علامت (}) پایان کلاس Form1 ، تعریف کلاس را انجام می دهیم:

```
public partial class Form1 : Form
{
    SpaceShip mySpaceShip;  // معرفی شئی از کلاس
    public Form1()
    {
        InitializeComponent();
        mySpaceShip = new SpaceShip(); // ایجاد شیئ
    }
}
// محل تعریف کلاس مورد نظر
class SpaceShip // تعریف کلاس سفینه
{
    public int fuel;   // مقدار سوخت
    public int shield; // مقدار سپر
    public int speed;  // مقدار سرعت
}
```

کلاس Form

کلاس سفینه

دقت کنید که کلاس خود را بعد از کلاس Form تعریف می کنیم و داخل کلاس Form شیئ از نوع کلاس می سازیم.

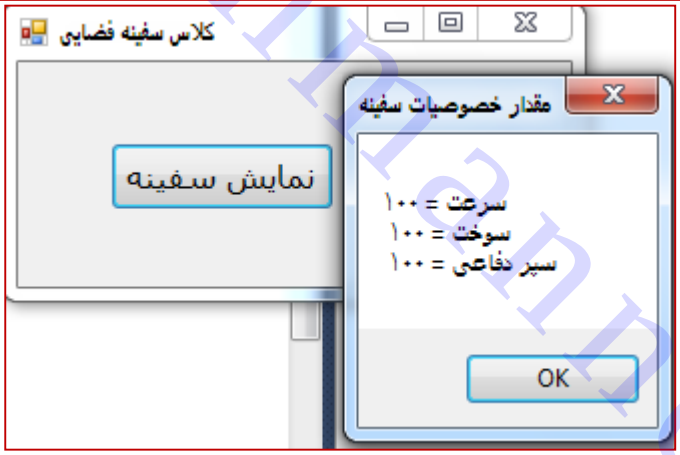
```
public partial class Form1 : Form
{
    SpaceShip mySpaceShip;  // معرفی شئی از کلاس
    public Form1()
    {
        InitializeComponent();
        mySpaceShip = new SpaceShip(); // ایجاد شیئ

        mySpaceShip.fuel = 100; // مقدار دهی اولیه با دسترسی به فیلدها
        mySpaceShip.shield = 100;
        mySpaceShip.speed = 100;

    }
}
// محل تعریف کلاس مورد نظر
class SpaceShip // تعریف کلاس سفینه
{
    public int fuel;   // مقدار سوخت
    public int shield; // مقدار سپر
    public int speed;  // مقدار سرعت
}
```

برای مقداردهی اولیه ویژگیهای کلاس، اگر از روش اول استفاده کنیم و مستقیماً به فیلدهای کلاس مقدار بدهیم کدها بصورت زیر خواهند بود که عدد ۱۰۰ به منزله ۱۰۰ درصد ویژگی مورد نظر است

حال می توان در یک رویداد دکمه مقادیر این ویژگیها را نمایش داد:

```
private void ShowBtn_Click(object sender, EventArgs e)
{
    string s = "سرعت = " + mySpaceShip.speed +
               "\n سوخت = " + mySpaceShip.fuel +
               "\n سپر دفاعی = " + mySpaceShip.shield;
    MessageBox.Show(s, "مقدار خصوصیات سفینه");
}
```

اجرای برنامه بصورت زیر است:




```
public partial class Form1 : Form
{
    SpaceShip mySpaceShip;  // معرفی شئی از کلاس
    public Form1()
    {
        InitializeComponent();
        mySpaceShip = new SpaceShip(); // ایجاد شیئ
        mySpaceShip.recharge(); // فراخوانی متد مقدار دهی
    }

    private void ShowBtn_Click(object sender, EventArgs e)
    {
        string s = "سرعت = " + mySpaceShip.speed +
                   "\n سوخت = " + mySpaceShip.fuel +
                   "\n سپر دفاعی = " + mySpaceShip.shield;
        MessageBox.Show(s, "مقدار خصوصیات سفینه");
    }
}
// محل تعریف کلاس مورد نظر
class SpaceShip // تعریف کلاس سفینه
{
    public int fuel;   // مقدار سوخت
    public int shield; // مقدار سپر
    public int speed;  // مقدار سرعت

    public void recharge() // متد مقدارهی
    {
        fuel = 100; // مقدار دهی به فیلدها به کمک متد
        shield = 100;
        speed = 100;
    }
}
```

اگر از روش دوم (استفاده از یک متد در کلاس سفینه) برای مقدار دهی استفاده کنیم، کد فوق را خواهیم داشت

اگر فیلدهای کلاس را بصورت private تعریف کنیم، در رویداد دکمه نمی توان به مقدار فیلدها از طریق شیئ دسترسی پیدا کرد. پس باید برای دسترسی به هر فیلد، متد واسطه طراحی کنیم و سپس آنها را فراخوانی کنیم. اگر بخواهیم مقدار هر فیلد را به دلخواه تغییر دهیم، باید متد دسترسی دیگری تعریف کنیم. پس برای سه فیلد کلاس، شش متد لازم است(سه متد برای دسترسی و تغییر و سه متد برای دسترسی خواندن انها). مثلاً برای دسترسی و تغییر فیلد سرعت(speed)، باید دو متد زیر را تعریف کنیم:

```
// محل تعریف کلاس مورد نظر
class SpaceShip // تعریف کلاس سفینه
{
    private int fuel;   // مقدار سوخت
    private int shield; // مقدار سپر
    private int speed;  // مقدار سرعت

    public void recharge() // متد مقدارهی
    {
        fuel = 100; // مقدار دهی به فیلدها به کمک متد
        shield = 100;
        speed = 100;
    }
    public int get_speed() // متد اطلاع از مقدار سرعت
    {
        return speed;
    }

    public void set_speed(int s) // متد تغییر مقدار سرعت
    {
         speed = s;
    }
}
```

حال برای تغییر مقدار سرعت باید دستور زیر را نوشت:

mySpaceShip.set_speed(70); → تنظیم مقدار سرعت به ۷۰

مثلاً اگر دکمه دیگری به فرم اضافه کنیم و رویداد آنرا بصورت زیر بنویسیم، مقدار سرعت را به ۷۰ درصد تغییر می دهد:

```
private void ChangeSpeed_Click(object sender, EventArgs e)// دکمه تغییر سرعت
{
    //mySpaceShip.speed = 70; //private :خطا! غیر قابل دسترسی مستقیم
    mySpaceShip.set_speed(70); // دسترسی از طریق متد
}
```

دقت نمایید که دسترسی مستقیم به فیلد speed از بیرون کلاس و از طریق شیئ mySpaceShip، امکان پذیر نیست؛ زیرا فیلد speed توسط توصیف کننده private به حالت خصوصی در آمده است.

برای اطلاع از مقدار سرعت باید دستور زیر را بکار برد:

mySpaceShip.get_speed();→اطلاع از مقدار سرعت

راه حل مناسبتر به جای نوشتن چندین متد برای دسترسی به فیلدها استفاده از Property است. در این روش متدی نوشته می شود که شبیه یک فیلد مورد استفاده برنامه نویس قرار می گیرد. در تعریف یک ویژگی، ترکیبی از روش تعریف فیلد و روش تعریف متد با هم استفاده می شود.

```
public  نوع داده فیلد  نام خصوصیت {
        get { return  نام فیلد ; }
        set {نام فیلد  = value ;}
}
```

```
public int Fuel {
    get { return fuel; }
    set { fuel = value; }
}
```

دقت کنید چون در زبان C# حروف کوچک و بزرگ متفاوت هستند، نام خصوصیت Fuel برای دسترسی به فیلد fuel استفاده شده است.

واژه های *public, get, set, value, return* کلمات کلیدی و رزرو شده هستند و برای تعریف property بکار می روند.

جدول ۹ــ۴ــ کلمات کلیدی مورد استفاده در تعریف ویژگی

کلمه کلیدی	شرح کار
get	بخشی از تعریف ویژگی را مشخص می‌کند که برای اطلاع از مقدار یک فیلد به‌کار می‌رود.
set	بخشی از تعریف ویژگی را مشخص می‌کند که برای انتساب یک مقدار به یک فیلد به‌کار می‌رود. در این بخش می‌توانیم با استفاده از دستور if مقداری که قرار است در فیلد قرار گیرد را کنترل و بررسی کنیم.
value	مقداری است که در برنامه معین شده و قرار است در فیلد قرار گیرد.
return	دقیقاً عملکردی مانند مقدار برگشتی متدها به برنامه اصلی را دارد.

هرگاه بخواهیم مقدار یک فیلد را تغییر دهیم یا آنرا بخوانیم، از property استفاده می کنیم. مثلاً اگر mySpaceShip شیئی از کلاس ما باشد، برای تغییر و خواندن مقدار فیلد fuel، دستوات زیر را بکار می بریم:

mySpaceShip.Fuel = 95; → *تغییر مقدار فیلد سوخت*

MessageBox.Show("سوخت = " + mySpaceShip.Fuel); → *خواندن مقدار فیلد سوخت*

برنامه کامل شده تا این مرحله بصورت زیر خواهد بود:

```
public partial class Form1 : Form
{
    SpaceShip mySpaceShip;  // معرفی شئی از کلاس
    public Form1()
    {
        InitializeComponent();
        mySpaceShip = new SpaceShip(); // ایجاد شیئ
        mySpaceShip.recharge(); // فراخوانی متد مقدار دهی
    }

    private void ShowBtn_Click(object sender, EventArgs e)
    {
        string s = "سرعت = " + mySpaceShip.Speed +
                   "\n سوخت = " + mySpaceShip.Fuel +
                   "\n سپر دفاعی = " + mySpaceShip.Shield;
        MessageBox.Show(s, "مقدار خصوصیات سفینه");
    }
}
class SpaceShip // تعریف کلاس سفینه
{
    private int fuel;   // مقدار سوخت
    private int shield; // مقدار سپر
    private int speed;  // مقدار سرعت

    public void recharge() // متد مقدارهی
    {
        fuel = 100; // مقدار دهی به فیلدها به کمک متد
        shield = 100;
        speed = 100;
    }
    public int Fuel // propery دسترسی به مقدار سوخت
    {
        get { return fuel; }
        set { fuel = value; }
    }
    public int Shield  // propery دسترسی به مقدار سپر حفاظتی
    {
        get { return shield; }
        set { shield = value; }
    }
    public int Speed // propery دسترسی به مقدار سرعت
    {
        get { return speed; }
        set { speed = value; }
    }
}
```

نکته: می توان در بخش تغییر مقدار property، قبل از ذخیره مقدار جدید، آنرا کنترل کرد که در محدوده مجاز باشد:

```
public int Shield  // propery دسترسی به مقدار سپر حفاظتی
{
    get { return shield; }
    set { if ((value>=0) && (value<=100))
            shield = value;
          else
            shield = 0;
    }
}
```

در این مثال، اگر مقدار وارد شده در محدوده مجاز ۰ تا ۱۰۰ باشد، ذخیره شده و گرنه صفر منظور خواهد شد.

mySpaceShip.Shield = 95; → *تغییر فیلد سپر به مقدار ۹۵*

mySpaceShip.Shield = 112; → *تغییر فیلد سپر به مقدار صفر(در محدوده مجاز ۰ تا ۱۰۰ نیست)*

پیاده سازی اعمال یک حساب بانکی به کمک مفاهیم شیئ گرایی

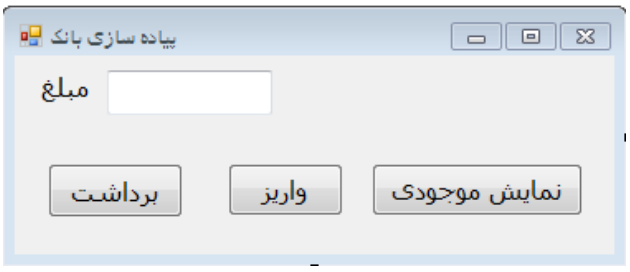
مثال : فرض کنید می خواهیم اعمال برداشت، واریز و مشاهده موجودی یک حساب بانکی را به کمک مفاهیم شیئ گرایی پیاده سازی کنیم:

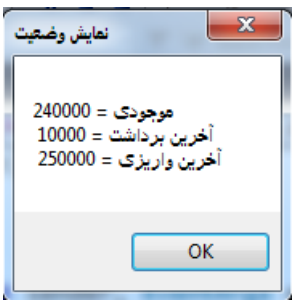
(۱) پیاده سازی کلاس بانک

- فیلدها: balance (موجودی)، withdraw (برداشت)، deposit(واریز)
- خاصیتهایی (Property) برای خواندن و ذخیره مقادیر فیلدها
  - Mowjudi : برای خواندن و تغییر موجودی
  - Bardasht: برای خواندن و تغییر فیلد برداشت و بررسی کافی بودن موجودی قبل از برداشت و به روزرسانی موجودی
  - Variz: برای خواندن و تغییر واریزی و به روزرسانی موجودی

```
class MyBank {
    private int balance ; // موجودی حساب
    private int withdraw ; // آخرین برداشت
    private int deposit;   // آخرین مقدار واریز

    public int Mowjudi {
        get { return balance; }
        set { balance = value; }
    } //Mowjudi
    public int Bardasht
    {
        get { return withdraw; }
        set {
            if (value <= balance)
            {
                withdraw = value; // ذخیره برداشت
                balance -= value; // به روزرسانی موجودی(کم می شود)ا
            }
            else
            {
                withdraw = 0;
                MessageBox.Show("موجودی کافی نیست","خطا در برداشت");
            }
        } // set
    }// Bardasht
    public int Variz
    {
        get { return deposit; }
        set {
            deposit = value; // ذخیره واریزی
            balance += value;// به روزرسانی موجودی(زیاد می شود)ا
        } //set
    }// Variz
} //MyBank آخر کلاس
```

(۲) پیاده سازی فرم برنامه





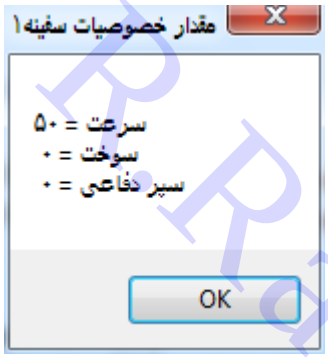
```
public partial class Form1 : Form
{   MyBank bank1; // معرفی شیئ بانک
    public Form1()
    {
        InitializeComponent();
        bank1 = new MyBank(); // ایجاد شیئ بانک
    }

    private void BardashtBtn_Click(object sender, EventArgs e)// برداشت
    {
        int money;
        money = int.Parse(moneyText.Text); // خواندن مقدار مبلغ
        bank1.Bardasht = money;  // برداشت
    }

    private void VarizBtn_Click(object sender, EventArgs e) // واریز
    {
        int money;
        money = int.Parse(moneyText.Text);
        bank1.Variz = money; // واریز
    }

    private void DisplayBtn_Click(object sender, EventArgs e) // نمایش
    {
        string str;
        str = "موجودی = " + bank1.Mowjudi +
              "\n آخرین برداشت = "+ bank1.Bardasht +
              "\n آخرین واریزی = "+ bank1.Variz;
        MessageBox.Show(str,"نمایش وضعیت");

    }
}
```

~~~~~~

فرض کنید دو سفینه فضایی می خواهیم ایجاد کنیم و مقدار سرعت آنها را به ترتیب ۵۰ و ۱۰۰ مشخص نماییم.

با زدن دکمه نمایش مشخصات سفینه ۱، پیام زیر مشاهده می شود که مقدار سوخت و سپر دفاعی صفر شده است.

```
SpaceShip mySpaceShip1;  // معرفی شئی سفینه اول
SpaceShip mySpaceShip2;  // معرفی شئی سفینه دوم
public Form1()
{
    InitializeComponent();
    mySpaceShip1 = new SpaceShip(); // ایجاد سفینه اول
    mySpaceShip2 = new SpaceShip(); // ایجاد سفینه دوم
    mySpaceShip1.Speed = 50; // تنظیم سرعت اولی
    mySpaceShip2.Speed = 100; // تنظیم سرعت دومی
}
```
~~~~~~



به هنگام ایجاد یک شیء، یک متد که به نام متد سازنده(Constructor)، معروف است به طور خودکار و پیش فرض اجرا می گردد و فیلدهای شیء را مقداردهی و معین می کند. اگر ما متدی را به عنوان متد سازنده معرفی نکنیم، هنگام ایجاد شیء اگر فیلدی مقداردهی اولیه نشده باشد مقدار پیش فرض به آن داده می شود. مقدار پیش فرض برای فیلدهای نوع عددی، عدد صفر و برای فیلدهای رشته ای مقدار تهی است.

متد سازنده(Constructor)

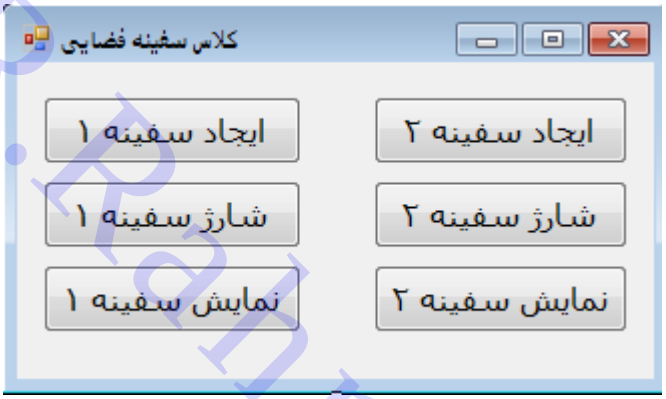
یکی از راه های مقداردهی اولیه به فیلدها و در واقع راه اصلی آن استفاده از متد سازنده است. متد سازنده، متدی است که در هنگام ایجاد شیء فراخوانی شده و عملیات آن انجام می شود. هرگاه بخواهیم در هنگام ایجاد یک شیء، عملیاتی انجام شود از متد سازنده استفاده می کنیم. این عملیات می تواند مقداردهی اولیه فیلدها یا هر دستور دیگری باشد.

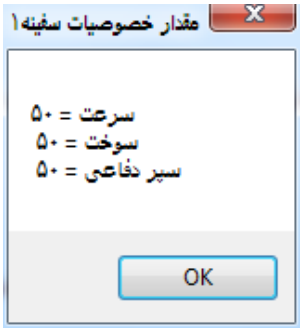
- نام متد سازنده همنام با نام کلاس است
- نوعی به عنوان خروجی نباید برای آن معین شود
- متد سازنده می تواند با پارامتر و یا بدون پارامتر باشد.
- می توان چندین متد سازنده تعریف کرد. یعنی چندین متد همنام با نام کلاس داشته باشیم که تفاوت آنها در پارامترهای ورودی متد است.
- هنگام ایجاد اشیاء، بصورت خودکار فراخوانی و اجرا می گردد.
- کاربرد: مقدار دهی اولیه به فیلدهای شیئ مورد نظر

مثلاً تعریف سازنده کلاس SpaceShip بصورت زیر خواهد بود:

```
class SpaceShip // تعریف کلاس سفینه
{
    private int fuel;   // مقدار سوخت
    private int shield; // مقدار سپر
    private int speed;  // مقدار سرعت


    public SpaceShip() // سازنده
    {
        fuel = 50; // مقدار دهی به فیلدها به کمک سازنده
        shield = 50;
        speed = 50;
    }
```

حال اگر دوباره برنامه را اجرا کنیم، با زدن دکمه نمایش مشخصات سفینه ۱، پیام زیر مشاهده می شود که برای همه فیلدها مقدار ۵۰ منظور شده است:





سوال: چگونه به کمک سازنده مقدار اولیه دلخواهی را برای فیلدها تنظیم کنیم؟

جواب: می توان برای سازنده، پارامتر ورودی تعریف کرد و متد سازنده، این پارامترهای ورودی را در فیلدها قرار دهد:

```
public SpaceShip(int fu , int sh, int sp) // سازنده دارای پارامتر ورودی
{
    fuel = fu; // مقدار دهی به فیلدها از طریق پارامتر سازنده
    shield = sh;
    speed = sp;
}
```

حالا در هنگام ایجاد اشیاء می توان مقادیر ورودی را ارسال نمود:

```
SpaceShip mySpaceShip1;  // معرفی شئی سفینه اول
SpaceShip mySpaceShip2;  // معرفی شئی سفینه دوم
public Form1()
{
    InitializeComponent();
    mySpaceShip1 = new SpaceShip(50, 75 ,95); // سازنده پارامتر دار
    mySpaceShip2 = new SpaceShip(); // سازنده بدون پارامتر

}
```

تمرین: پروژه را به شکل زیر کامل کنید

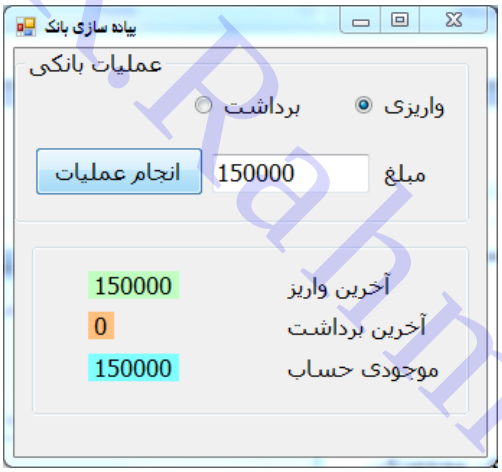
دستورات داخل Form1 بصورت زیر است:

```
public partial class Form1 : Form
{
    SpaceShip mySpaceShip1;  // معرفی شئی سفینه اول
    SpaceShip mySpaceShip2;  // معرفی شئی سفینه دوم
    public Form1()
    {
        InitializeComponent();
    }

    private void ShowBtn_Click(object sender, EventArgs e)
    {
        string s = "سرعت = " + mySpaceShip1.Speed +
                   "\n سوخت = " + mySpaceShip1.Fuel +
                   "\n سپر دفاعی = " + mySpaceShip1.Shield;
        MessageBox.Show(s, "مقدار خصوصیات سفینه۱");
    }
    private void ShowBtn2_Click(object sender, EventArgs e)
    {
        string s = "سرعت = " + mySpaceShip2.Speed +
                   "\n سوخت = " + mySpaceShip2.Fuel +
                   "\n سپر دفاعی = " + mySpaceShip2.Shield;
        MessageBox.Show(s, "مقدار خصوصیات سفینه۲");
    }

    private void Charge1_Click(object sender, EventArgs e)
    {
        mySpaceShip1.recharge(); // شارژ سفینه ۱
    }

    private void Charge2_Click(object sender, EventArgs e)
    {
        mySpaceShip2.recharge();// شارژ سفینه ۲
    }

    private void Create1_Click(object sender, EventArgs e)
    {
        mySpaceShip1 = new SpaceShip(50, 75, 95); // ایجاد سفینه ۱
    }

    private void Create2_Click(object sender, EventArgs e)
    {
        mySpaceShip2 = new SpaceShip(60,70,80); // ایجاد سفینه ۲
    }
}
```

ساختار کلاس سفینه که در ادامه کد بالا نوشته می شود، بصورت زیر نوشته می شود.

```
class SpaceShip // تعریف کلاس سفینه
{
    private int fuel;   // مقدار سوخت
    private int shield; // مقدار سپر
    private int speed;  // مقدار سرعت

    public SpaceShip() // سازنده
    {
        fuel = 50; // مقدار دهی به فیلدها به کمک سازنده
        shield = 50;
        speed = 50;
    }

    public SpaceShip(int fu , int sh, int sp) // سازنده دارای پارامتر ورودی
    {
        fuel = fu; // مقدار دهی به فیلدها از طریق پارامتر سازنده
        shield = sh;
        speed = sp;
    }
    public void recharge() // متد مقدارهی
    {
        fuel = 100; // مقدار دهی به فیلدها به کمک متد
        shield = 100;
        speed = 100;
    }

    public int Fuel //  دسترسی به مقدار سوخت
    {
        get { return fuel; }
        set
        {
            if ((value >= 0) && (value <= 100))
                fuel = value;
            else
                fuel = 0;
        }
    }
    public int Shield  //  دسترسی به مقدار سپر حفاظتی
    {
        get { return shield; }
        set
        {
            if ((value>=0) && (value<=100))
                shield = value;
              else
                shield = 0;
        }
    }
    public int Speed //  دسترسی به مقدار سرعت
    {
        get { return speed; }
        set
        {
            if ((value >= 0) && (value <= 100))
                speed = value;
            else
                speed = 0;
        }
    }
}
```

تمرین(۱): ساختار فرم بانک را بصورت زیر طراحی کنید و سپس تغییرات زیر را در کلاس آن ایجاد نمایید.



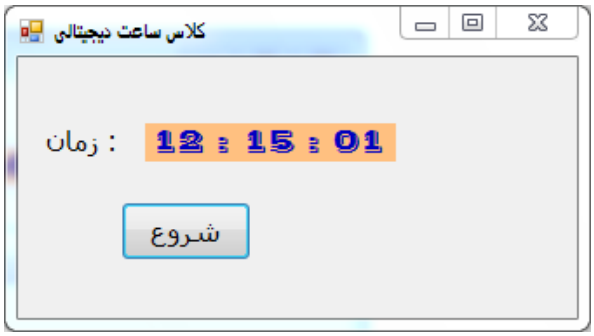
برای کلاس بانک دو متد سازنده تعریف کنید.

- **سازنده اول** بدون پارامتر و مقادیر همه فیلدهای موجودی، برداشت و واریزی را صفر کند
- **سازنده دوم** که نقش افتتاح حساب دارد، دارای یک پارامتر به عنوان مبلغ واریزی شروع داشته باشد و مقدار موجودی را نیز به مقدار واریزی تغییر دهد و برداشت صفر منظور شود.
- **در داخل کلاس فرم**، دو شیئ (bank1 , bank2) را بسازید که شیئ bank1 سازنده اول و شیئ bank2 سازنده دوم را استفاده نماید.
- یک متد به نام UpdateMoney به کلاس فرم اضافه نمایید که یک پارامتر مبلغ را داشته و بسته به وضعیت انتخاب دکمه های رادیویی واریز یا برداشت، عملیات مورد نظر را انجام دهد.
- عملیات بانک را فقط روی شیئ bank1 انجام دهید و بعد از انجام هر عمل، نتایج در سه برچسب پایین فرم نمایش یابد.

مثال:

۱) کلاسی (CTime)برای زمان تعریف کنید و اجزای زیر را پیاده نمایید.

- داده ها: ساعت، دقیقه و ثانیه(hour, minute , second)
- سازنده بدون پارامتر که مقادیر زمان ساعت، دقیقه و ثانیه را صفر نماید
- سازنده با سه پارامتر برای مقداردهی ساعت، دقیقه و ثانیه
- سه مشخصه(property) به نامهای myHour, myMinute,mySecond برای دسترسی و تغییر ساعت، دقیقه و ثانیه
- متد getTime() که مقدار زمان را بصورت مقابل به قالب یک رشته string برگشت دهد 11:58:45
- متد addTime برای افزایش ثانیه به مقدار پارامتر آن و کنترل مقادیر ثانیه، دقیقه و ساعت. یعنی اگر ثانیه به ۶۰ برسد باید ثانیه صفر شود و دقیقه اضافه شود و ...



۲) یک پروژه بسازید و کنترلهای timer، دکمه، برچسب و ... اضافه نموده و در رویداد یک دکمه زمان فعلی سیستم را خوانده و مقادیر آنرا در شیئ کلاس CTime به کمک سازنده ذخیره نمایید.

۳) رویداد timer باید هر یک ثانیه یکواحد به مقدار ثانیه شیئ اضافه نموده و نتیجه را نمایش دهد(ساعت دیجیتالی)

نکته: برای دسترسی به ساعت سیستم از دستور زیر استفاده کنید:

```
int h = DateTime.Now.Hour; // ساعت : بدست آوردن زمان سیستم
int m = DateTime.Now.Minute;// دقیقه : بدست آوردن زمان سیستم
int s = DateTime.Now.Second;// ثانیه : بدست آوردن زمان سیستم
```

کد و دستورات فرم برنامه بصورت زیر است:

```
public partial class Form1 : Form
{
    CTime myTime; // تعریف شیئ زمان
    public Form1()
    {
        InitializeComponent();
    }

    private void timer1_Tick(object sender, EventArgs e)//timer رویداد

    {
        myTime.addTime(1);// افزایش یک ثانیه به زمان
        lblTime.Text = myTime.getTime(); // بدست آوردن زمان و نمایش آن
    }

    private void button1_Click(object sender, EventArgs e) // شروع
    {
        int h = DateTime.Now.Hour; // ساعت : بدست آوردن زمان سیستم
        int m = DateTime.Now.Minute;// دقیقه : بدست آوردن زمان سیستم
        int s = DateTime.Now.Second;// ثانیه : بدست آوردن زمان سیستم
        myTime = new CTime(h, m, s); // ذخیره مقادیر در شیئ به کمک سازنده
        timer1.Enabled = true;
    }
}
```

کد ساختار کلاس به صورت زیر خواهد بود:

```
class CTime { // کلاس زمان
    private int hour;
    private int second;
    private int minute;

    public CTime() // سازنده بدون پارامتر
    {
        hour = minute = second = 0;
    }
    public CTime(int h , int m , int s) // سازنده سه پارامتری
    {
        myHour = h;
        myMinute = m;
        mySecond = s;
    }
    public int myHour // مشخصه دسترسی به ساعت
    {
        get { return hour; }
        // کنترل محدوده مجاز ساعت بین 0 تا 23
        set { hour = ((value >= 0) && (value <= 23)) ? value : 0; }
    }
    public int myMinute // مشخصه دسترسی به دقیقه
    {
        get { return minute; }
        // کنترل محدوده مجاز دقیقه بین 0 تا 59
        set { minute = ((value >= 0) && (value <= 59)) ? value : 0; }
    }
    public int mySecond // مشخصه دسترسی به ثانیه
    {
        get { return second; }
        // کنترل محدوده مجاز ثانیه بین 0 تا 59
        set { second = ((value >= 0) && (value <= 59)) ? value : 0; }
    }
    public string getTime() { // متد نمایش زمان طبق قالب
        string s1="", s2="", s3="";
        if (hour < 10) s1 = "0"; // برای مقدار یک رقمی، صفر به اول اضافه شود
        if (minute < 10) s2 = "0";
        if (second < 10) s3 = "0";
        return s1 + hour.ToString() + " : " +
               s2 + minute.ToString() + " : " +
               s3 + second.ToString();
    }
    public void addTime(int s) { // افزودن زمان بر حسب ثانیه به شیئ زمان
        second += s;
        if (second > 60) // کنترل مقدار ثانیه
        {
            minute ++; // افزایش دقیقه
            second -= 60; //  اصلاح ثانیه
        }
        if (minute > 60)
        {
            hour ++;
            minute -= 60;
        }
        if (hour > 23)
            hour = 0;
    }

}
```

فصل سیزدهم – فایل

فایل چیست؟

دنباله ای(Stream) از بایت ها را که در روی حافظه های جانبی تحت یک نام نگهداری می شود، فایل می نامند. فایل ها درون حافظه های جانبی ذخیره می شوند تا برای دسترسی های بعدی مورد استفاده قرار بگیرند.

انواع مختلف فایل ها از نظر محتوا:

الف) فایل متنی(Text File):

- فایلی که محتوای آن، کاراکترهای چاپ شدنی است
- در انتهای فایل های متنی، یک بایت به عنوان پایان فایل نشانه گذاری می شود که به آن کاراکتر پایان فایل (End of file) می گویند.
- محتوای فایل های متنی با یک ویرایشگر متنی مانند NotePad یا Word قابل مشاهده است.

ب) فایل دودویی(Binary file):

- داده های موجود در متغیّرهای یک برنامه که شامل اعداد و رشته ها است، به همان شکل که در حافظه قرار دارند، در فایل دودویی ذخیره می شوند.
- فایلی که محتوای آن، کاراکترهای چاپ شدنی نیست.
- محتوای فایل های متنی با یک ویرایشگر متنی مانند NotePad یا Word به درستی قابل مشاهده نیستند.

انواع مختلف فایل ها از نظر ترتیب ذخیره سازی اطلاعات و یا نحوه دسترسی به اطلاعات درون یک فایل عبارتند از فایل با دسترسی ترتیبی (Sequential access) و فایل با دسترسی مستقیم (Direct access)

در کتابخانه .NET در فضای نامی System.IO کلاس FileStream برای کار روی فایلها تعریف شده است. همچنین کلاسهای File, Directory, Path که شامل متدهای استاتیک هستند، به ترتیب برای انجام عملیات مختلف بر روی فایل ها، فولدرها و همچنین مسیر دسترسی به فایل ها و فولدرها به کار می رود. چند متد استاتیک از کلاس File عبارتند از:

- Exists( ) : بررسی می کند که آیا فایل وجود دارد یا نه. اگر فایل وجود داشته باشد مقدار خروجی متد برابر با true است و گرنه مقدار false را برگشت می دهد.

بررسی وجود فایل ← *System.IO.File.Exists("مسیر فایل")*

مثلاً دستور زیر بررسی می کند که آیا فایل *test.txt* در مسیر درایو *d:* وجود دارد یا خیر؟

*System.IO.**File**.Exists("d:\test.txt")*

WriteAllText( ) : یک فایل متنی را ایجاد نموده، متنی را در آن نوشته و سپس فایل را می بندد. اگر فایل مورد نظر قبلاً موجود باشد، اطلاعات قبلی پاک می شود و اگر موجود نباشد، آنرا ایجاد می نماید.

نوشتن داده رشته ای در فایل ← *System.IO.File.WriteAllText ("مسیر فایل", داده رشته ای)*

مثلاً دستور زیر محتوای کادر متنی *textBox1* را در فایل *test.txt* در مسیر درایو *d:* ذخیره می کند.

System.IO.File.WriteAllText ("d:\test.txt", textBox1.Text)

- AppendAllText( ) : یک فایل متنی را باز نموده، متنی را با آن اضافه کرده و سپس فایل را می بندد. اگر فایل مورد نظر قبلاً موجود باشد، اطلاعات قبلی محفوظ مانده و متن جدید به آن اضافه می شود و اگر موجود نباشد، آنرا ایجاد می نماید.

نوشتن داده رشته ای در انتهای فایل ← *System.IO.File. AppendAllText ("مسیر فایل", داده رشته ای)*

مثلاً دستور زیر محتوای کادر متنی *textBox1* را به فایل *test.txt* در مسیر درایو *d:* اضافه و ذخیره می کند.

System.IO.File. AppendAllText ("d:\test.txt", textBox1.Text)

- ReadAllText( ) : یک فایل متنی را باز نموده، محتویات آنرا خوانده و بصورت رشته ای بر می گرداند و سپس فایل را می بندد. اگر فایل مورد نظر قبلاً موجود باشد، اطلاعات آنرا می خواند و اگر موجود نباشد، خطا اعلام می شود.

خواندن داده های فایل ← *متغیر رشته ای = System.IO.File. ReadAllText ("مسیر فایل")*

مثلاً دستور زیر محتوای فایل را خوانده و در کادر متنی *textBox1* قرار می دهد.

textBox1.Text = System.IO.File. ReadAllText ("d:\test.txt")

مثال: نام کاربری و رمز کاربر و تکرار رمز را دریافت کنید و در فایل متنی ذخیره نمایید. به کمک کنترل CheckBox نمایش یا عدم نمایش رمز را کنترل نمایید. توجه کنید درستی اطلاعات ورودی را کنترل کنید:

- نام وارد شده باشد
- رمز وارد شده باشد
- تکرار رمز وارد شده و با رمز یکی باشد.

حل: ساختار فرم را به شکل زیر طراحی نمایید. نام کادرهای متنی عبارتند از: *userNameTxt* و *passwordTxt* و *verifiedPassTxt*

نکات قابل توجه:

- مشخصه PasswordChar را برای کنترل کادر متنی تنظیم نمایید تا رمز وارد شده نمایان نباشد. مثلاً علامت * قرار دهید تا به جای حروف تایپی نمایش داده شود.

- در کد نویسی می توان مشخصه PasswordChar را تنظیم کرد یا پاک نمود:

```
passwordTxt.PasswordChar = '*'; → علامت رمز تنظیم شود
passwordTxt.PasswordChar = '\u0000'; → علامت رمز پاک شود
passwordTxt.PasswordChar = (char) 0 ; → علامت رمز پاک شود
```

دقت کنید که روش اول پاک کردن علامت رمز، کارکتر تهی با کد **'\u0000'** را قرار می دهد و روش دوم عدد صفر را به کد کارکتر تبدیل می کند(**(char) 0**) که اثر هردو یکسان است.

- ویژگی Length از مشخصه Text کادر متنی، طول رشته ورودی را نشان می دهد که با بررسی آن می توان ورود مقدار را بررسی نمود.

```
if(userNameTxt.Text.Length ==0){ // بررسی اینکه نام کاربر وارد شده است یا نه
```

- متد ResetText( ) از کادر متنی، متن آنرا پاک می کند:

```
userNameTxt.ResetText();
```

- متد Trim( )، فواصل خالی اول و آخر متن را حذف می کند:

```
userNameTxt.Text.Trim();
```

- دستور return، باعث می شود که اجرای متد متوقف شود و از ادامه دستورات بعدی صرف نظر گردد.

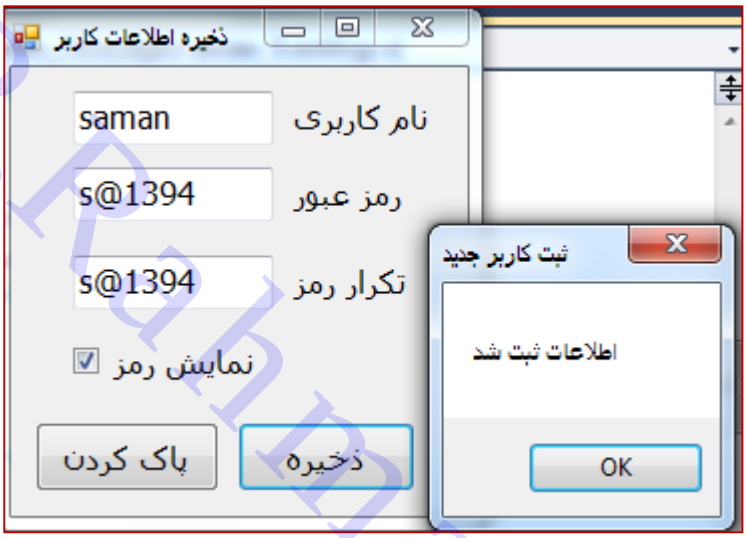
حال کد دستورات برنامه کادر انتخاب «نمایش رمز» و دکمه «پاک کردن» بصورت زیر خواهد بود:

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)// نمایش رمز
{
    if (passwordTxt.PasswordChar == '*') // اگر قبلاً تنظیم شده
    {
        passwordTxt.PasswordChar = '\u0000'; // علامت رمز پاک شود
        verifiedPassTxt.PasswordChar = (char)0;// علامت رمز پاک شود
    }
    else
    {
        passwordTxt.PasswordChar = '*'; // علامت رمز تنظیم شود
        verifiedPassTxt.PasswordChar = '*';
    }
}
private void clearBtn_Click(object sender, EventArgs e) // دکمه پاک کردن
{
    userNameTxt.ResetText();
    passwordTxt.ResetText();
    verifiedPassTxt.ResetText();
}
```

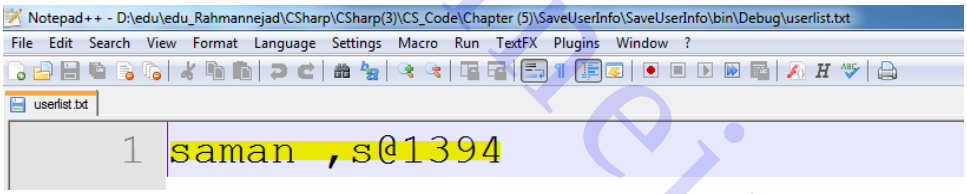
کد برنامه دکمه ذخیره طبق دستورات زیر خواهد بود:

```
private void writeBtn_Click(object sender, EventArgs e) //ذخیره
{
    string errorTitle=" خطا در ورود اطلاعات"; // عنوان پیام
    userNameTxt.Text.Trim() ; //  حذف فواصل خالی اول و آخر متن
    if(userNameTxt.Text.Length ==0){ // اگر طول رشته صفر یا خالی باشد
        MessageBox.Show(" لطفاً نام کاربری را وارد کنید ",errorTitle);
        return; // خروج از متد
    }
    if(passwordTxt.Text.Length ==0){
        MessageBox.Show(" لطفاً کلمه عبور را وارد کنید ",errorTitle);
        return;
    }
    if(verifiedPassTxt.Text.Length ==0){
        MessageBox.Show(" لطفاً تکرار کلمه عبور را وارد کنید ",errorTitle);
        return;
    }
    if (verifiedPassTxt.Text != passwordTxt.Text)//اگر رمز با تکرار آن یکسان نباشد
    {
        MessageBox.Show(" کلمه عبور با تکرار آن باید یکسان باشد ",errorTitle);
        return;
    }
    string textData = userNameTxt.Text +" ," + passwordTxt.Text;
    string fileName = "userlist.txt";
    System.IO.File.WriteAllText (fileName , textData);
    MessageBox.Show(" اطلاعات ثبت شد","ثبت کاربر جدید" );
}
```

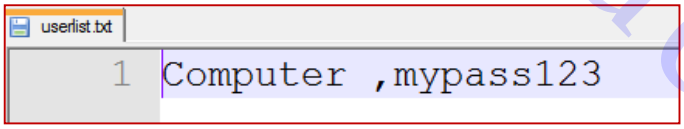
نمونه اجرا و خروجی:



حال اگر به مسیر محل ذخیره برنامه بروید و پوشه bin و سپس پوشه Debug را باز کنید می بینید که فایل متنی userlist.txt ایجاد شده است. می توانید با ویرایشگر notepad++ محتویات فایل را ببینید:



اگر بار دیگر نام و رمز را ذخیره کنید، محتویات فایل قبلی حذف شده و با مقدار جدید جایگزین می گردد.

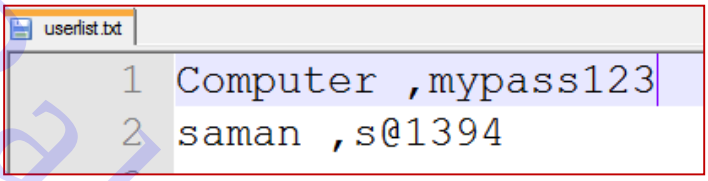


دلیل: چون دستور WriteAllText( )، محتوای قبلی فایل را حذف می کند.

اگر به جای دستور WriteAllText( )، متد AppendAllText( ) را در رویداد دکمه «ذخیره»، جایگزین کنیم، با هر بار ذخیره، اطلاعات جدید به آخر فایل اضافه شده و محتوای قبلی محفوظ است:

```
private void writeBtn_Click(object sender, EventArgs e) //ذخیره
{
    .
    .
    .
    string textData = userNameTxt.Text +" ," + passwordTxt.Text;
    string fileName = "userlist.txt";
    System.IO.File.AppendAllText (fileName , textData+ " \r\n");
    MessageBox.Show(" اطلاعات ثبت شد","ثبت کاربر جدید" );
}
```

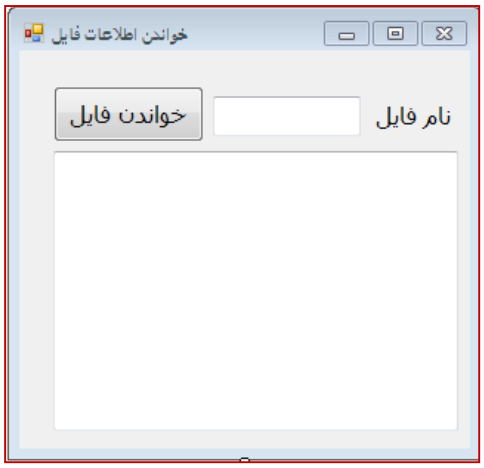
نکته: رشته کارکتر "\r\n" به عنوان کارکتر خط جدید در فایل متنی عمل کرده و سبب می شود پس ازنوشتن اطلاعات کاربر در فایل، خط جدیدی ایجاد شود تا اطلاعات کاربر بعدی در ابتدای خط جدید نوشته شود.



تمرین ۱: با استفاده از کنترل SaveFileDialog، مسیر و نام فایل مورد نظر را برای ذخیره تعیین نمایید و سپس اطلاعات را ذخیره کنید. دقت کنید که مشخصه Filter را طوری تنظیم نمایید که فقط فایلها با پسوند متنی(*.txt)، قابل تعیین باشد.

```
if (saveFileDialog1.showDialog() == DialogResult.ok)
{
   System.IO.File.WriteAllText(saveFileDialog1.FileName, textData);
   MessageBox.Show("اطلاعات بطور موفقیت‌آمیز ثبت شد", "ثبت کاربر جدید");
}
```

مثال: برنامه ای طراحی کنید که فایل متنی موجود در مسیر ذخیره پروژه را باز نموده و اطلاعات آنرا در یک textbox نمایش دهد. نام کادرهای متن نام فایل و محتوا به ترتیب fileNameTxt و fileContentTxt می باشد.




```
// دکمه خواندن فایل
private void readFileBtn_Click(object sender, EventArgs e)
{
    string fileName = fileNameTxt.Text; //دریافت نام فایل
    if (System.IO.File.Exists(fileName)) // اگر فایل موجود است
        // محتویات فایل را به کادر متن بخوان //
        fileContentTxt.Text = System.IO.File.ReadAllText(fileName);
    else  // فایل موجود نیست
        fileContentTxt.Text = "Error: File \'" + fileName +
                              "\' does not Exist!";
}
```

تمرین ۲: به جای اینکه کاربر نام فایل را در کادر متنی تایپ کند، یک کادر محاوره ای باز کردن فایل (OpenFileDialog) ظاهر می شود و کاربر فایل دلخواه خود را برای خواندن بازنماید.

فصل چهاردهم- اتصال برنامه کاربردی به پایگاه داده

پایگاه داده چیست(Database)؟

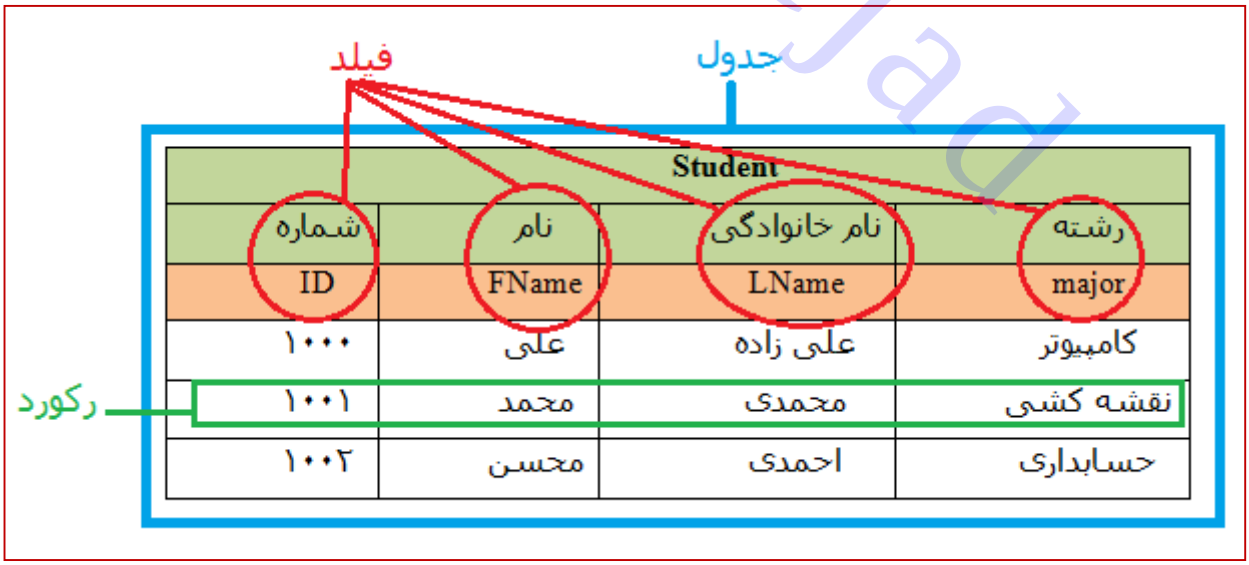
پایگاه داده حجم انبوهی از اطلاعات است که به طور منظم و سازماندهی شده بر روی حافظه های جانبی به صورت فایل نگهداری می شود. اطلاعات درون یک پایگاه داده به سادگی و با سرعت، قابل دستیابی است. معمولاً در یک پایگاه داده اطلاعات در یک یا چند جدول قرار می گیرند .

جدول چیست(Table) ؟

هر جدول شامل اطلاعاتی در مورد یک موضوع است. مثلاً پایگاه داده یک مدرسه که اطلاعات درسی دانش آموزان را نگهداری می کند، می تواند شامل چندین جدول اطلاعات شناسنامه ای دانش آموزان و نمرات آنها باشد. در جدول دانش آموزان، اطلاعاتی نظیر نام، نام خانوادگی، شمارهٔ ملی و آدرس مربوط هر یک از دانش آموزان مدرسه نگهداری می شود.

رکورد(Record) : هرسطر از یک جدول یک رکورد است

فیلد(Field): هر یک از اجزای تشکیل دهندهٔ رکورد، فیلد نامیده می شود. مانند نام، نام خانوادگی، شمارهٔ ملی و آدرس.



سیستم مدیریت پایگاه داده(Database Management System : DBMS)

برنامه ای که اطلاعات درون یک پایگاه داده را سازماندهی و مدیریت می نماید. برنامه Access نمونه ای از یک DBMS است که همراه مجموعه Office وجود دارد. این برنامه برای نگهداری اطلاعات و داده های شرکت ها و مؤسسات کوچک مناسب است. نمونه های دیگر SQL Server و Oracle می باشند توسط مؤسسات بزرگ استفاده می گردند.

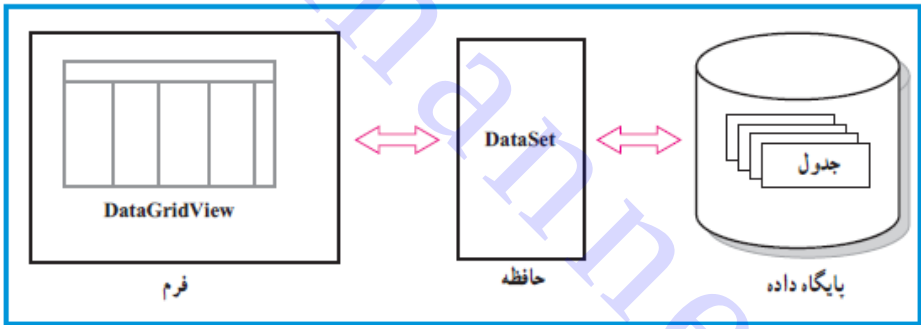
ایجاد پایگاه داده توسط Visual Studio و استفاده از آن

به طور کلی می توان اطلاعات درون یک پایگاه داده را در پروژه نرم افزار کاربردی استفاده کرد و یا ویرایش کرد .به عبارت دیگر از طریق کنترل های موجود در یک فرم، می توان داده های درون پایگاه داده را مشاهده نمود. همچنین داده هایی که کاربر در روی فرم وارد می کند و یا اطلاعات دستکاری شده، می تواند در پایگاه داده، ذخیره شود

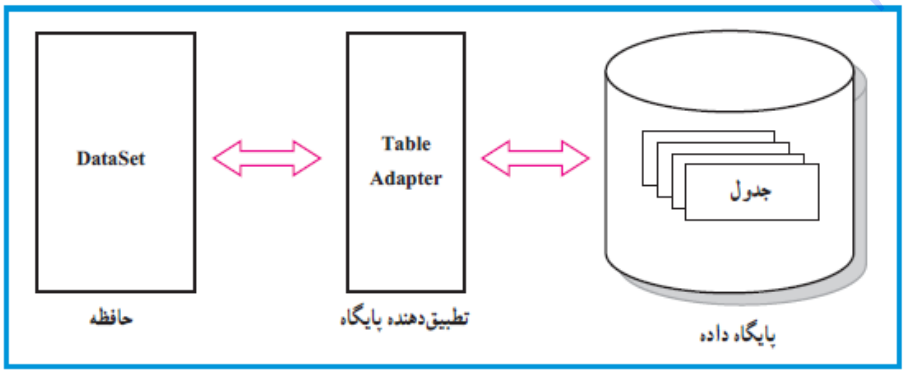
عملیات لازم برای دسترسی به اطلاعات درون یک پایگاه داده

۱) ابتدا اطلاعات مورد نظر از داخل پایگاه داده، به حافظه RAM کامپیوتر منتقل می شود به طوری که قابل مدیریت باشد .این عمل به وسیلهٔ ابزاری به نام DataSet انجام می شود

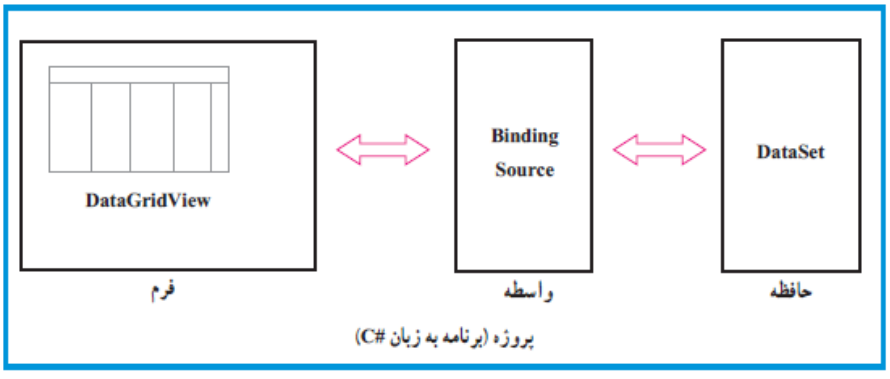
۲) سپس باید ابزاری برای نمایش اطلاعات روی فرم مورد استفاده قرار گیرد. اگر بخواهیم اطلاعات یک جدول را روی فرم نمایش دهیم، از ابزار DataGridView استفاده می شود. برای نمایش اطلاعات یک یا چند فیلد از ابزارهای دیگری مانند کنترل های برچسب یا کادر متنی می توان استفاده کرد.
شکل زیر ارتباط اجزا تا این مرحله از کار را نشان می دهد.



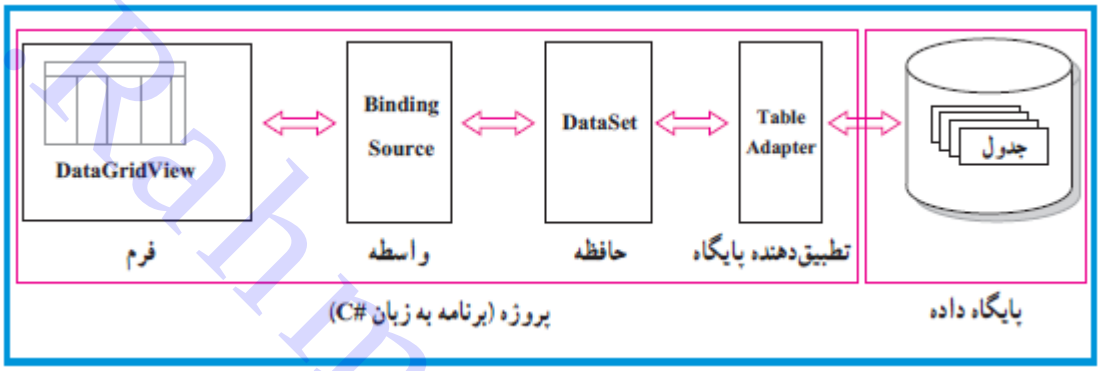
۳) یک واسطه به نام تطبیق دهنده جدول (TableAdapter) باید بکار رود تا برنامه بتواند به اطلاعات انواع پایگاه داده مستقل از نوع آن متصل گردد. بنابراین برای هر پایگاه داده، یک تطبیق دهنده مخصوص طراحی می شود که از آن برای انتقال اطلاعات به DataSet استفاده گردد.



۴) برای اینکه بین اطلاعات حافظه RAM یعنی واسط DataSet و کنترل نمایش جدول (DataGridView)، ارتباط برقرار گردد، از یک واسطه دیگری به نام BindingSource استفاده می گردد.



۵) برای هر یک از عملیات بالا کلاس و یا ابزاری تهیه شده است که از آنها در برنامه استفاده می شود و خلاصه ارتباطات در شکل زیر مشهود است.



در کتابخانه .NET مجموعه ای از کلاس های قدرتمندی برای اتصال با پایگاه داده و انجام عملیات مختلف بر روی داده، تحت عنوان ADO.NET تعریف شده است. در محیط برنامه نویسی Visual Studio با استفاده از ابزارهای تصویری و مرحله به مرحله می توانید تمام اشیاء لازم برای اتصال به یک پایگاه داده را به دست آورید بدون اینکه لازم شود دستوراتی را بنویسید.

انواع کلاس ها و ابزارهای کار با پایگاه داده

- **DataGridView**: یک ابزار واسطه گرافیکی است که برای نمایش و به روز رسانی رکوردهای اطلاعاتی استفاده می شود.
- **BindingSource**: کلاسی است که داده های نمایش داده شده بر روی فرم را به منبع آن اتصال می دهد.
- **DataSet**: کلاسی است برای ذخیره داده های پایگاه داده در حافظه، شیئ ساخته شده از این کلاس می تواند اطلاعات یک یا چند جدول را ذخیره نماید.
- **TableAdapter**: کلاسی است که ارتباط بین پایگاه داده با نرم افزار کاربردی را مهیا می کند.

مراحل ساخت یک نرم افزار متصل به پایگاه داده:

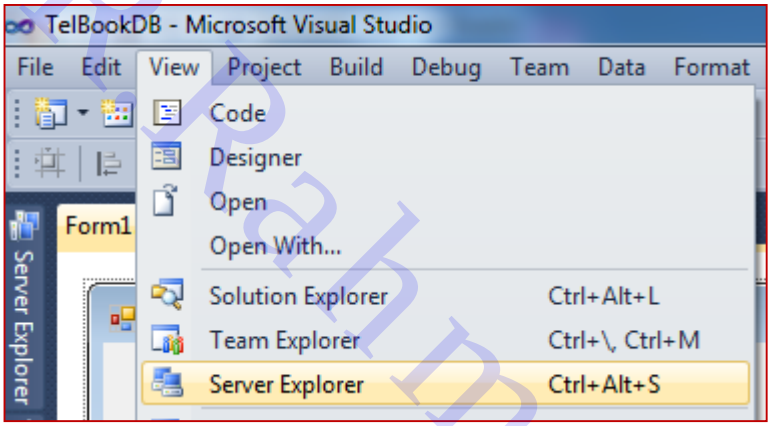
۱) ایجاد الگوریتم و نقشه کار مناسب
۱) طراحی پایگاه داده مورد نیاز
۲) طراحی واسط کاربری مناسب
۳) ایجاد رابطه بین نرم افزار و پایگاه داده
۴) تکمیل کد ها و برنامه
۵) آزمایش و رفع اشکال برنامه

مثال: طراحی یک دفترچه تلفن

حل: یک پروژه جدید حاوی فرم ایجاد نمایید و مسیر ذخیره پروژه را تنظیم کنید.

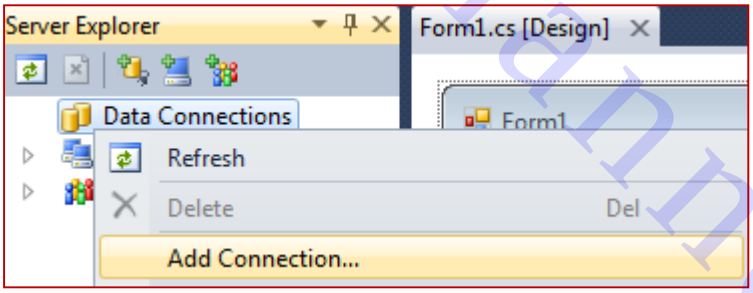
نام فیلد (انگلیسی)	نام فیلد (فارسی)
ID	شماره
Fname	نام
Lname	نام خانوادگی
City	شهر
Address	آدرس
Tel	تلفن
Mobile	موبایل

الف) طراحی پایگاه داده و جدول مربوطه

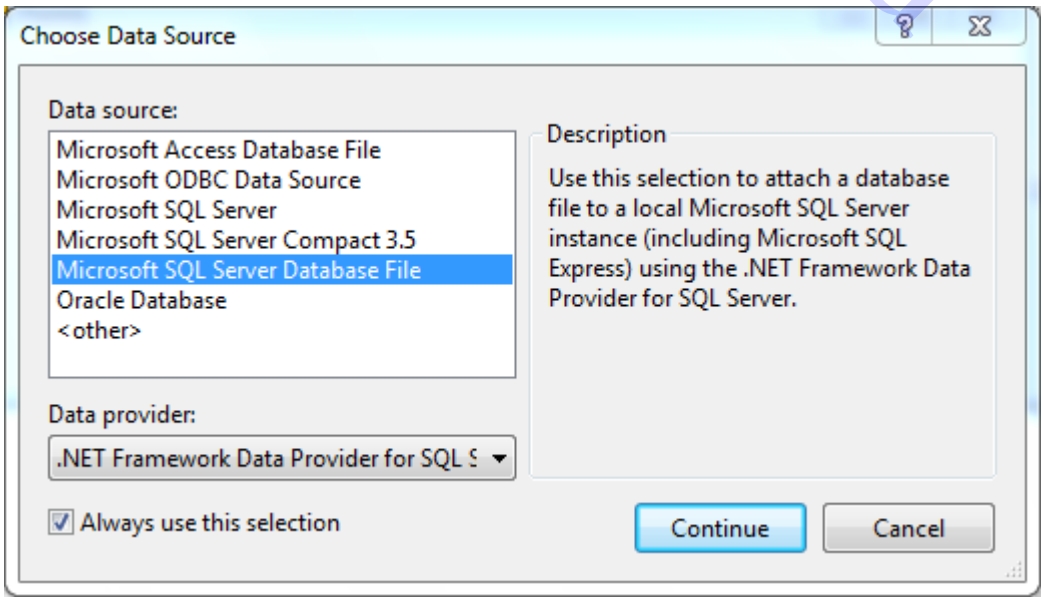


۱) پنجره Server Explorer را از منوی View فعال نمایید یا کلید ترکیبی Ctrl+Alt+S را بزنید.

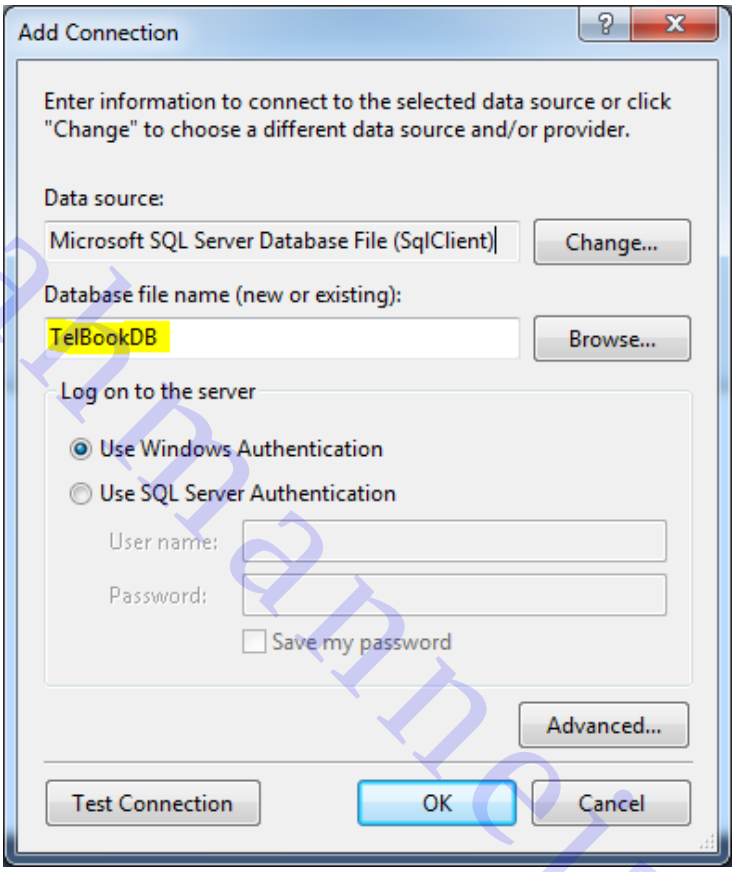
۲) روی Data Connection کلیک راست نموده و گزینه Add Connection را انتخاب نمایید. در این قسمت مشخصات پایگاه داده تعیین می گردد.



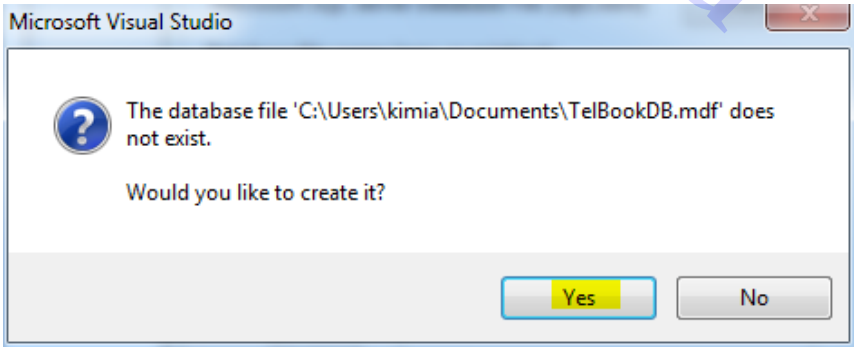
۳) در پنجره ظاهر شده انواع بانکهای اطلاعاتی قابل ایجاد را مشاهده می کنید. برای ایجاد بانک اطلاعاتی از نوع SQL Server گزینه Microsoft SQL Server DataBase File را انتخاب نموده و دکمه Continue را بزنید.



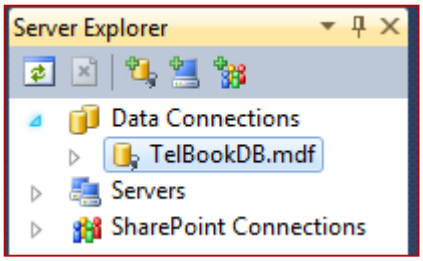
۴) در پنجره Add Connection ظاهر شده باید مسیر و نام پایگاه داده را تعیین نماییم. برای آنکه فایل در محل پیش فرض(پوشه documents) ایجاد شود در کادر Database file name نام دلخواه را برای پایگاه داده می نویسیم، اما اگر می خواهید در مکان دیگری فایل را ذخیره نمایید یا اگر فایل پایگاه داده را قبلاً ساخته اید می توانید آن فایل را با زدن دکمه Browse انتخاب کنید



۵) با زدن دکمه OK پایگاه داده مربوطه ساخته می شود. در این مرحله پیامی نمایش داده خواهد شد که می گوید فایل پایگاه داده وجود ندارد، آیا می خواهید آنرا ایجاد کنید. دکمه Yes را برای تأیید بزنید

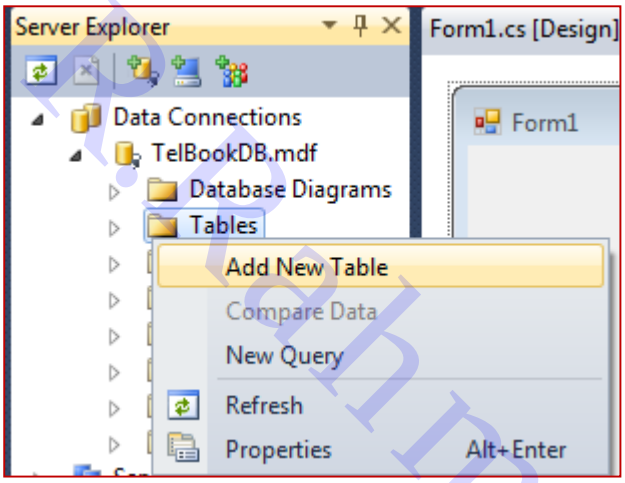


پس از ساخته شدن فایل در پنجره Server Explorer خواهیم دید یک اتصال به پایگاه داده جدید، ایجاد شده است.

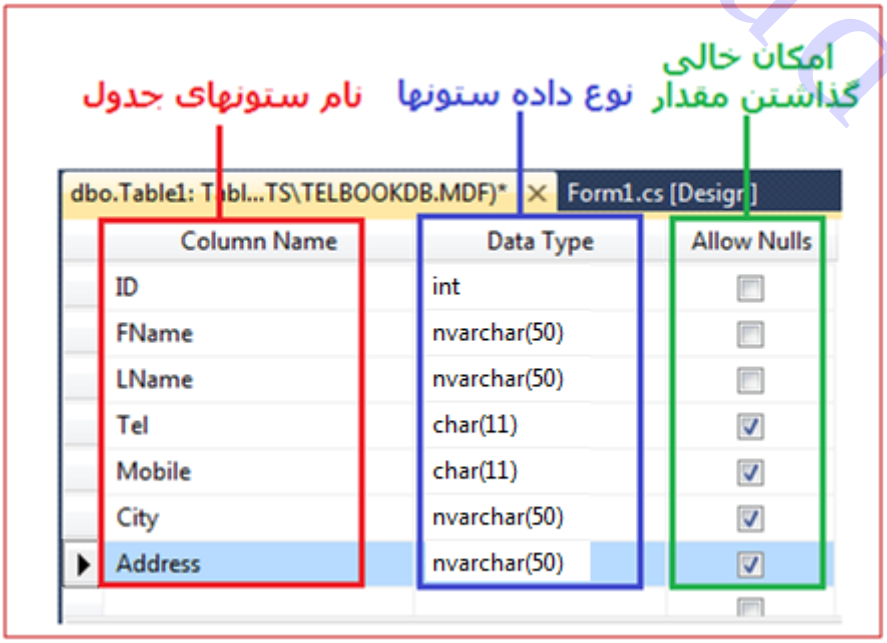


۶) روی علامت مثلث ▷ کنارنام پایگاه داده کلیک کنید تا اجزاء آنرا مشاهده نمایید. سپس بر روی شاخه Tables کلیک راست نموده و گزینه Add New Table را انتخاب کنید.

۷) در پنجره ای که ظاهر می گردد می توانید نام و مشخصات جدول جدید را وارد کنید.



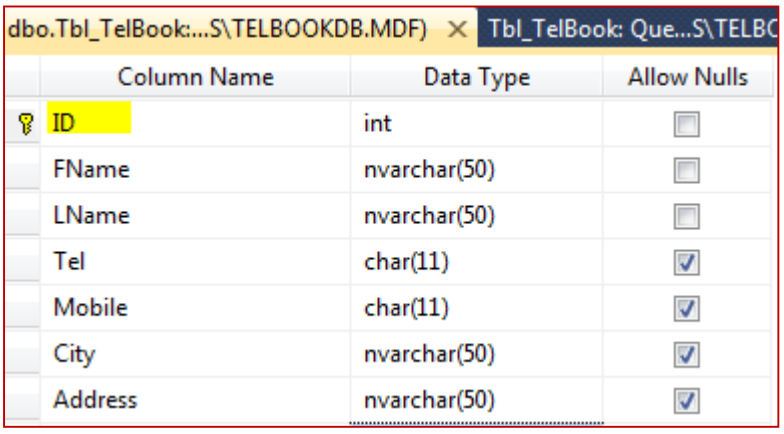
- در قسمت Column Name نام ستونهای جدول را وارد نمایید.
- در قسمت Data Type نوع داده ای که داده های ستون باید در خود ذخیره کنند، مشخص نمایید.
- در قسمت Allow Nulls، تعیین می کنید که آیا ستون مورد نظر می تواند دارای مقدار نباشد. اگر گزینه آن تیک زده شود ☑، به این معناست که ستون می تواند خالی باشد و مقدار نگیرد. اما اگر تیک نخورد ☐ یعنی مقدار ستون در هنگام ورود اطلاعات به جدول، باید حتماً وارد شود. مثلاً یک نفر ممکن است شماره تلفن ثابت را ارائه نکرده باشد، پس این فیلد خالی بوده و علامت ☑ برای آن در قسمت Allow Nulls فعال شده است.



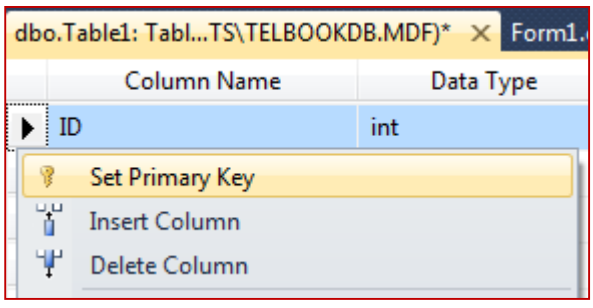
برای آنکه نوع داده یک فیلد را مشخص کنید باید اطلاعاتی را دربارهٔ انواع داده ها در SQL Server داشته باشید. تعداد محدودی از انواع داده ها را در جدول زیر می بینید:

نوع	توضیح
char	برای فیلدهای متنی غیر یونیکد درنظر گرفته می‌شود. بنابراین اگر متنی که می‌خواهید وارد کنید انگلیسی است از این نوع استفاده کنید. همچنین این نوع مناسب برای رشته‌هایی با طول معین می‌باشد.
nchar	برای فیلدهای متنی یونیکد استفاده می‌شود. بنابراین برای وارد کردن متنی که حروف فارسی دارد این نوع استفاده می‌شود. این نوع نیز برای رشته‌های با طول معین، مناسب است.
nvarchar	برای فیلدهای متنی یونیکد این نوع در نظر گرفته می‌شود. مقدار بایت در نظر گرفته شده برای هر فیلد بستگی به تعداد حرف متن ورودی دارد. این نوع برای رشته‌های متنی با طول نا معین مناسب است.
int	برای مقادیر عددی صحیح، از این نوع استفاده می‌شود.

- ستون ID عددی است و نوع آن int تعیین شده است
- ستون نام، نام خانوادگی، شهر و آدرس چون کارکتری با طول مختلف هستند، نوع nvarchar می توانند باشند. عدد داخل پرانتز برای نوع داده، حداکثر تعداد حروف آنرا مشخص می کند. مثلاً نام nvarchar(20) تعیین شده است یعنی به فیلد نام حداکثر ۲۰ حرف اختصاص یابد.
- ستونهای Tel,Mobile چون دارای ۱۱ کارکتر ثابت هستند (به همراه صفر شروع کد شهرستان و موبایل)، از نوع char(11) تعیین شده اند. شماره تلفن و موبایل چون پردازش عددی ندارند و می خواهیم صفر اول را هم ذخیره کنیم، هرچند از ارقام تشکیل شده اند، اما از نوع char تعریف شده اند.

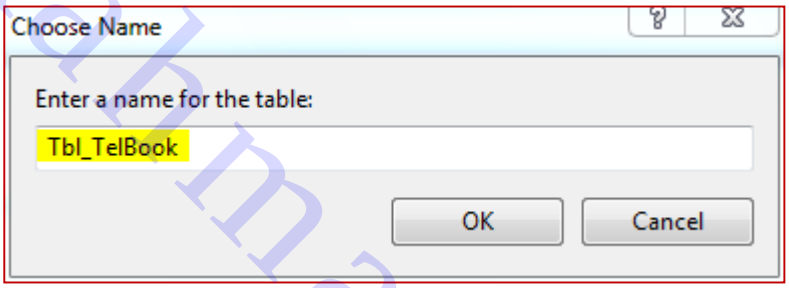


۶) هر جدول معمولاً یک کلید اصلی دارد. برای تعیین آن می توانید روی ستون مورد نظر کلیک راست نموده و گزینه Set Primary Key را انتخاب نمایید تا علامت کلید کنار آن ظاهر شده و به عنوان کلید اصلی فرض شود.

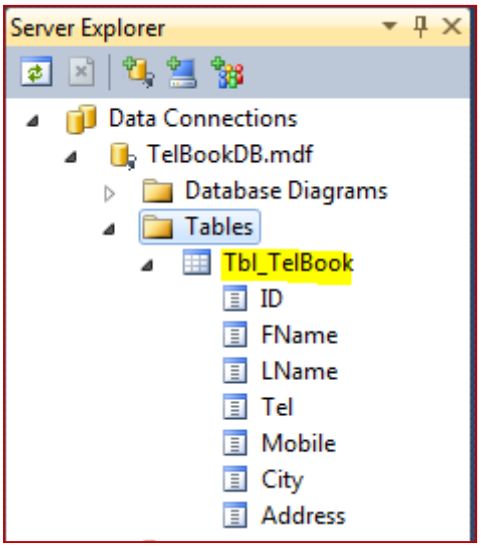


می توان ترکیبی از چند فیلد را با نگه داشتن Shift و کلیک ماوس انتخاب کرده و مجموعه را به عنوان کلید معرفی نمود.

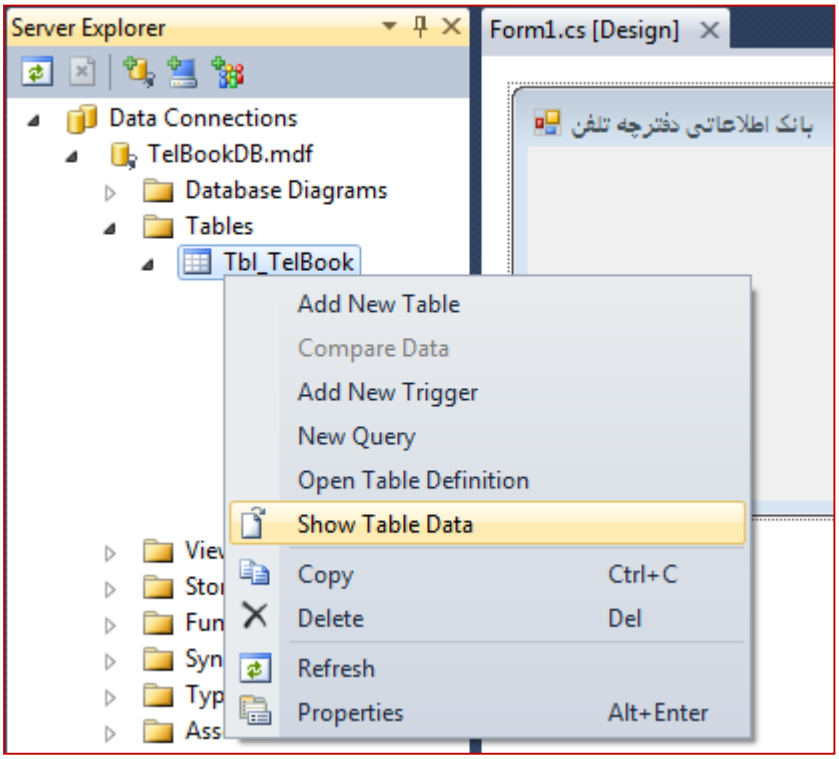
۸) بعد از طراحی جدول بانک اطلاعاتی، با زدن Save از نوار ابزار، کادر تعیین نام جدول ظاهر می گردد. نام جدول را تایپ نموده و دکمه Ok را بزنید.



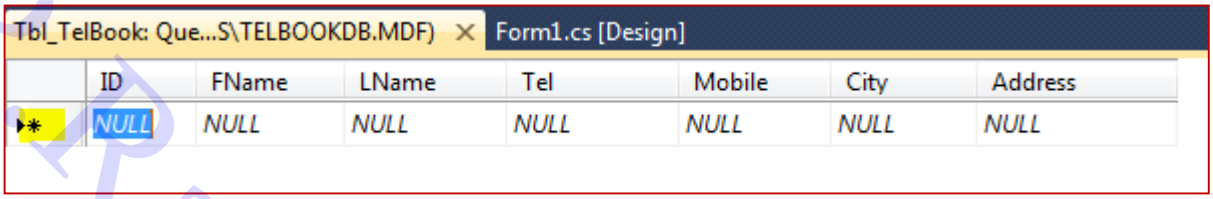
حال جدول مربوطه به بانک اطلاعاتی اضافه شده است.



۹) برای اینکه بتوان تعدادی رکورد در جدول وارد کرد، به پنجره Server Explorer برگشته و روی جدول کلیک راست نموده و گزینه Show Table Data را انتخاب نمایید.

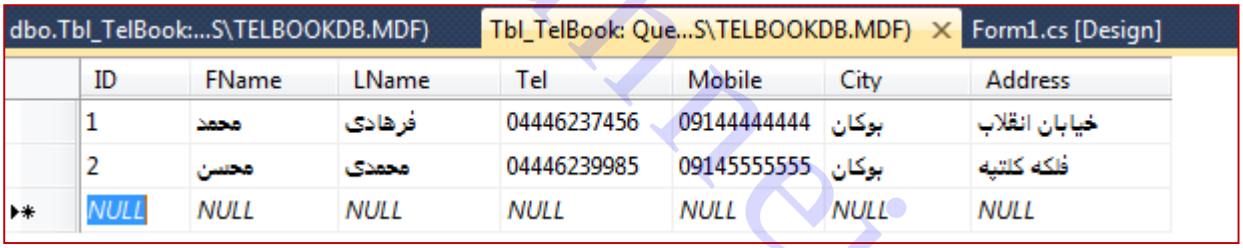


در پنجره ظاهر شده علامت * در کنار سطر به منزله ورود سطر جدید است.



کلمه NULL در هریک از ستونها به منزله عدم وجود مقدار برای آن فیلد است. دقت نمایید که NULL مقدار نیست و فقط علامتی است برای نشان دادن اینکه مقداری وارد نشده است.

۱۰) حال می توانید اطلاعات تماس چند نفر را وارد نمایید. لزومی ندارد اطلاعات تمام افراد را وارد کنید، چون بعداً نیز از طریق برنامه می توان اطلاعات دیگری اضافه نمود.

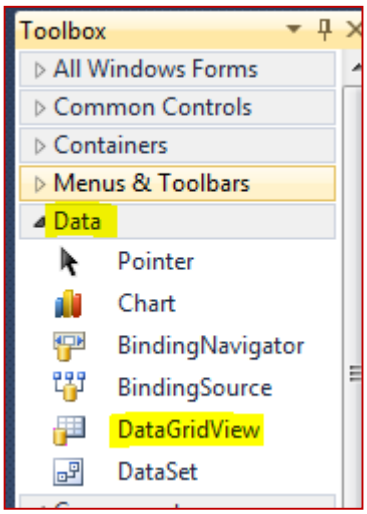


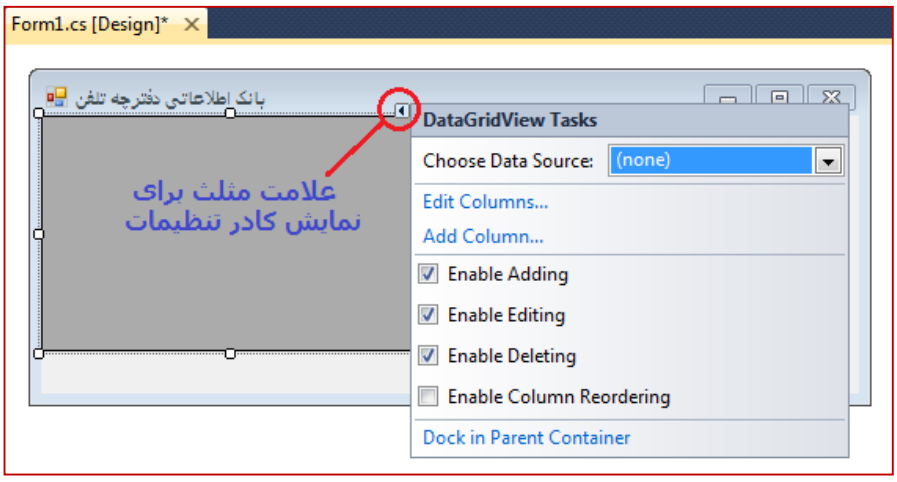
ب) طراحی واسط کاربری

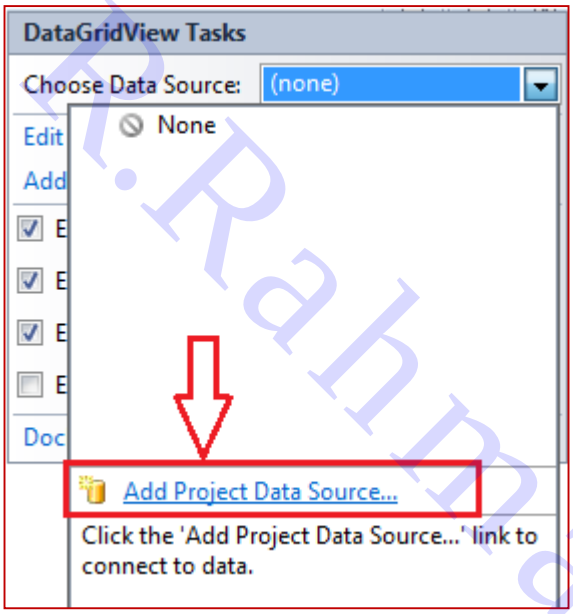
بعد از طراحی بانک اطلاعاتی، حال نوبت به طراحی واسط کاربری برای ارتباط با بانک اطلاعاتی می باشد.

۱) تنظیمات فونت و ظاهر فرم را تعیین نمایید و ویژگی RightToLeft آنرا به True تغییر دهید تا از راست به چپ باشد.

۲) برای اینکه بتوان اطلاعات موجود در یک جدول پایگاه داده را، به صورت یک جدول نمایشی در روی فرم مشاهده کرد از ابزار DataGridView استفاده می کنیم. این ابزار در قسمت Data از ToolBox قرار دارد. این ابزار را به فرم اضافه کنید و ویژگی RightToLeft آنرا به True تغییر دهید تا از راست به چپ باشد.



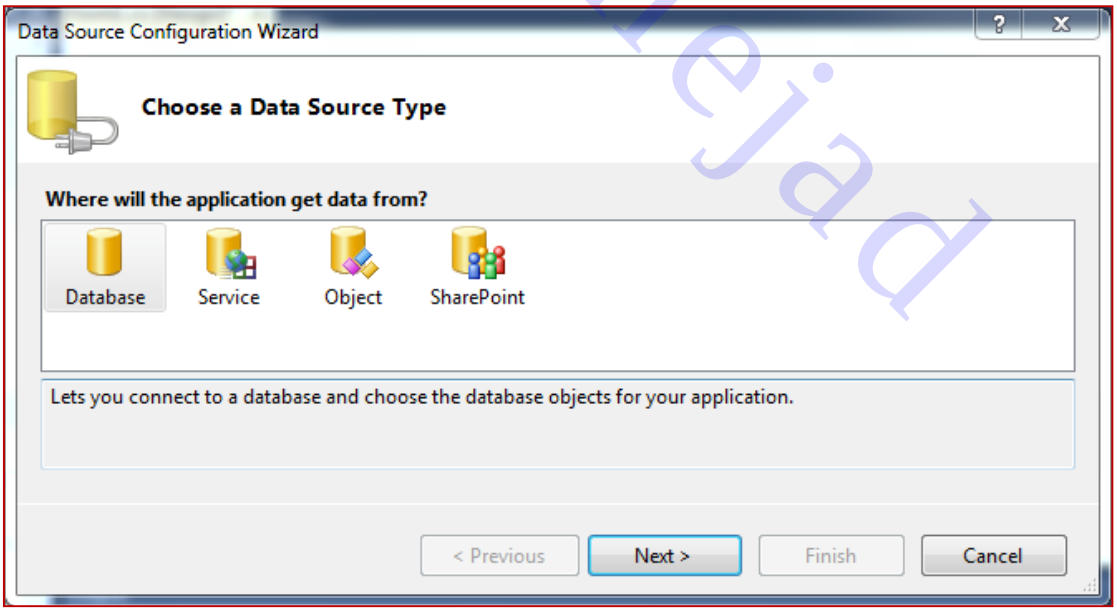


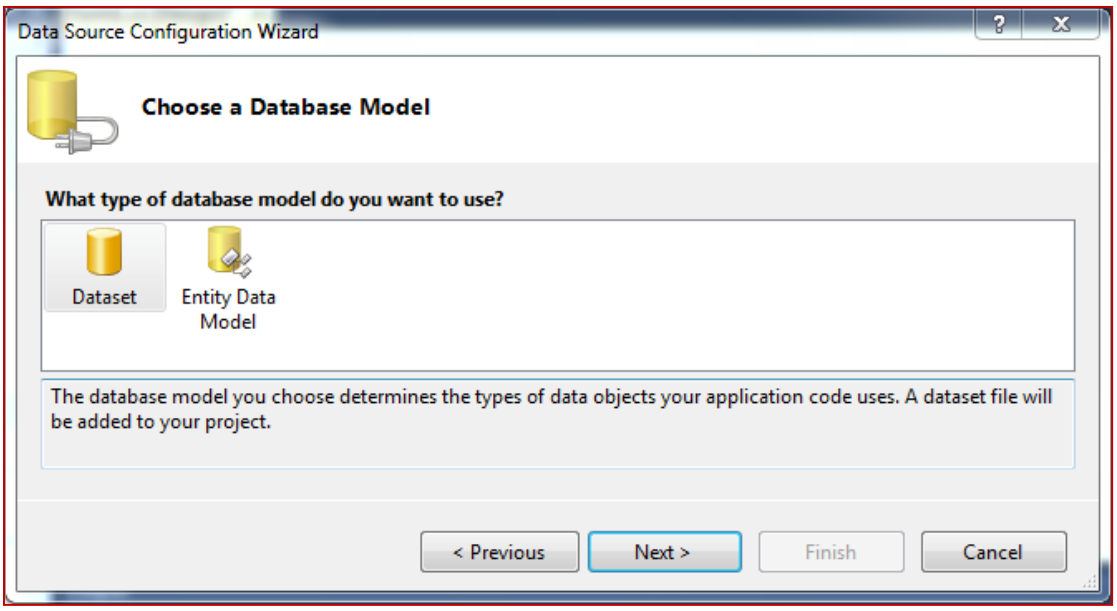


۳) با اضافه شدن کنترل DataGridView به فرم، کادری در کنار آن ظاهر می گردد. البته با زدن علامت مثلث در گوشه راست و بالای آن نیز می توان این کادر را نمایان کرد.

۴) روی لیست کشویی Choose Data Source کلیک کنید و در کادر ظاهر شده، گزینه *Add Project Data Source* را انتخاب کنید

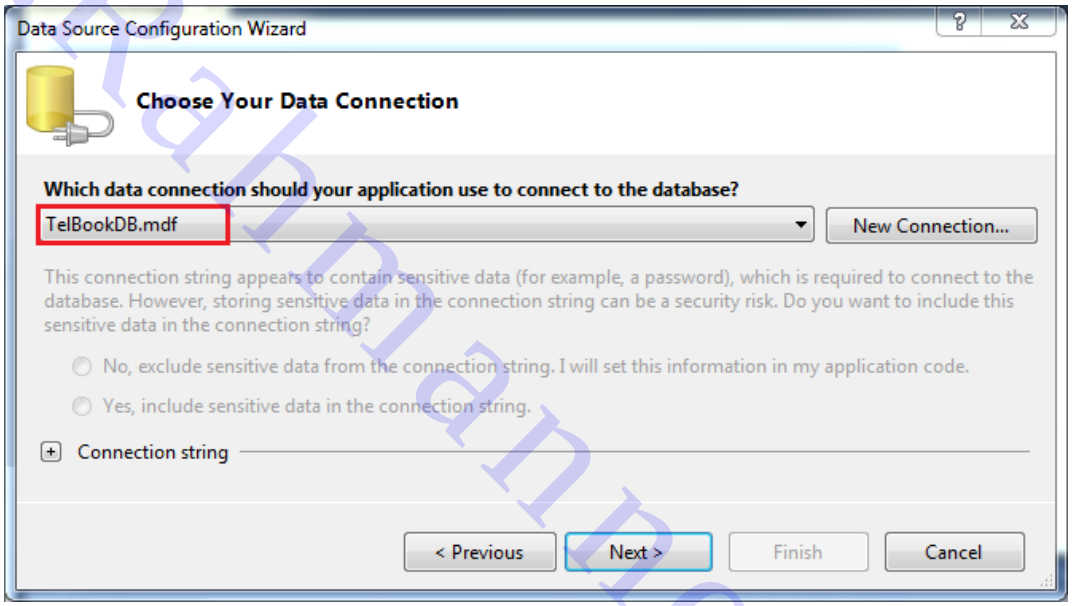
۵) در پنجره Choose a Data Source Type، گزینه DataBase را انتخاب و روی دکمه Next کلیک نمایید.



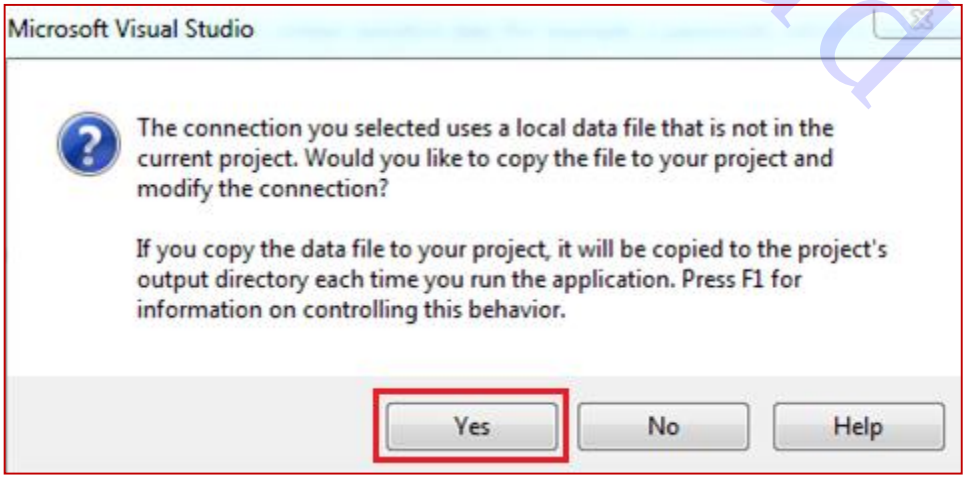


۶) در پنجره Choose Database Model، گزینه DataSet را انتخاب نموده و روی دکمه Next کلیک کنید.

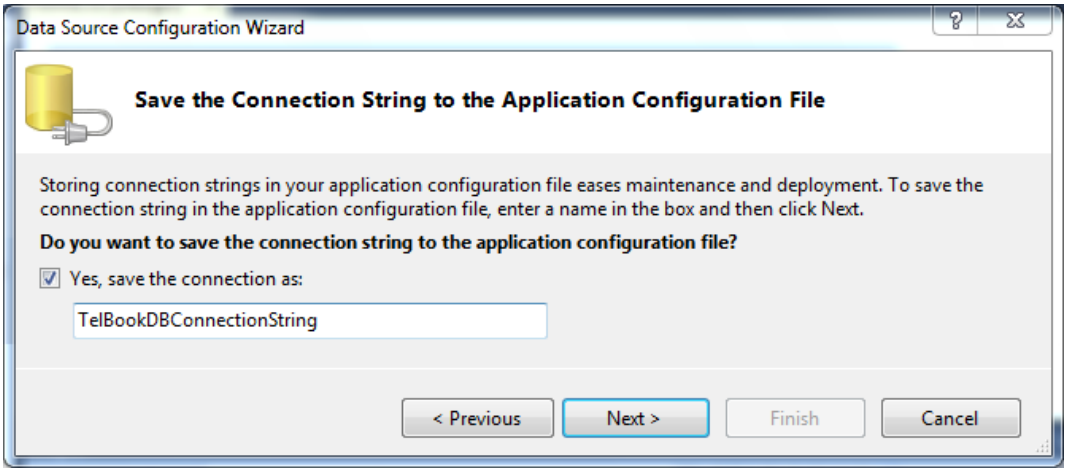
۷) در پنجره Choose Your Data Connection، نام بانک اطلاعاتی را که قبلاً ساخته ایم، انتخاب می کنیم که بصورت پیش فرض در لیست می باشد



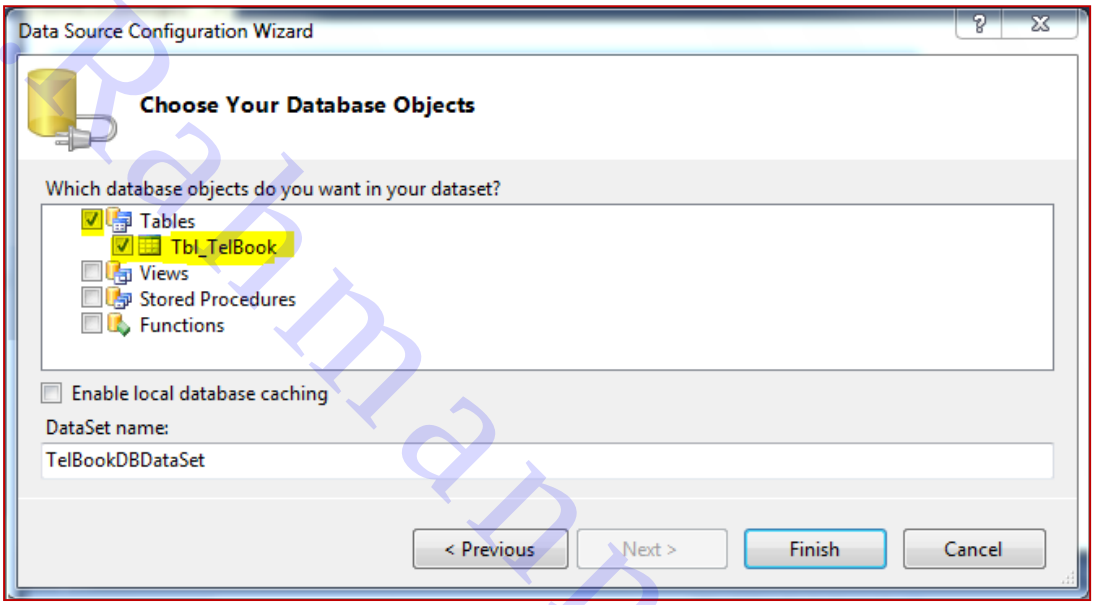
با زدن دکمه Next، کادر پیامی به نمایش در می آید که اگر دکمه Yes را انتخاب کنید، با این کار فایل پایگاه داده در پروژه کپی می شود و جزئی از پروژه محسوب می شود. بنابراین به صورت خودکار با ساختن فایل خروجی، در مسیر پوشه خروجی، فایل پایگاه داده نیز کپی می شود



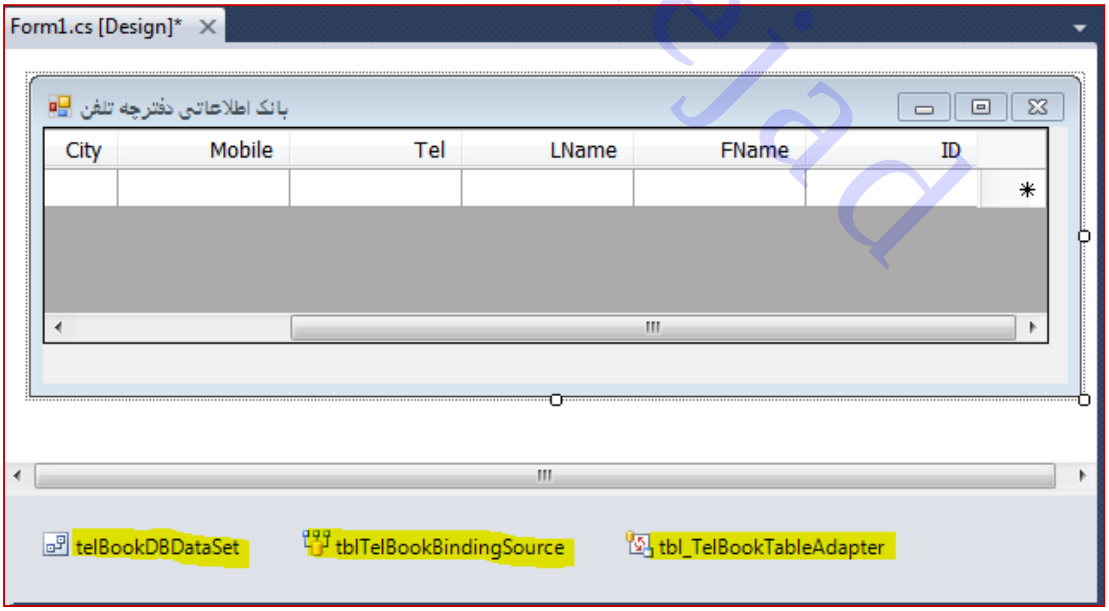
۸) در مرحله بعدی تنظیم های انجام شده در برنامه ذخیره می شود.



۹) با زدن Next در مرحله بعد می توان، جدول(یا جدولهای مربوطه) را برای اتصال به DataSet تعیین کرد. در نهایت دکمه Finish را بزنید.

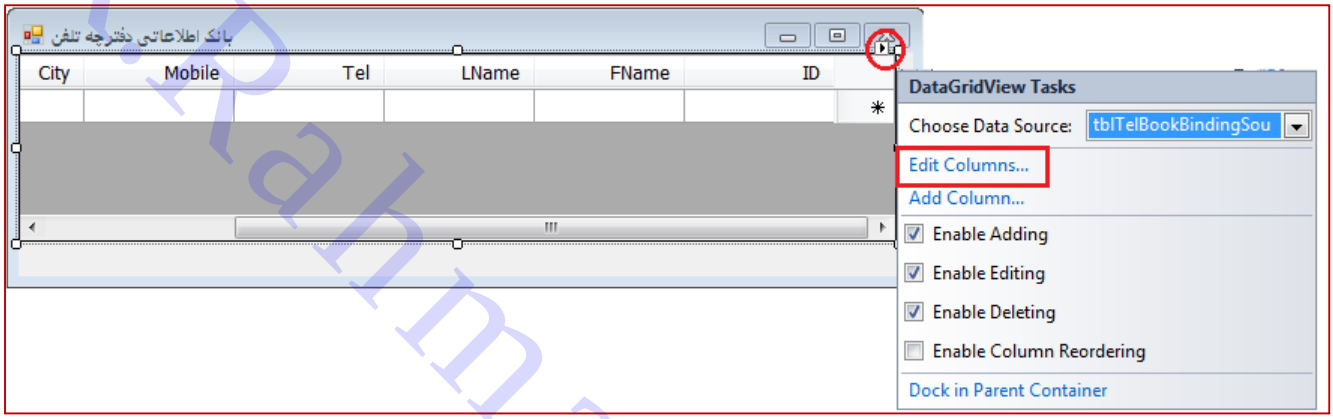


۱۰)به این ترتیب تمام واسطه های مورد نیاز به طور خودکار و بدون برنامه نویسی ساخته می شود.

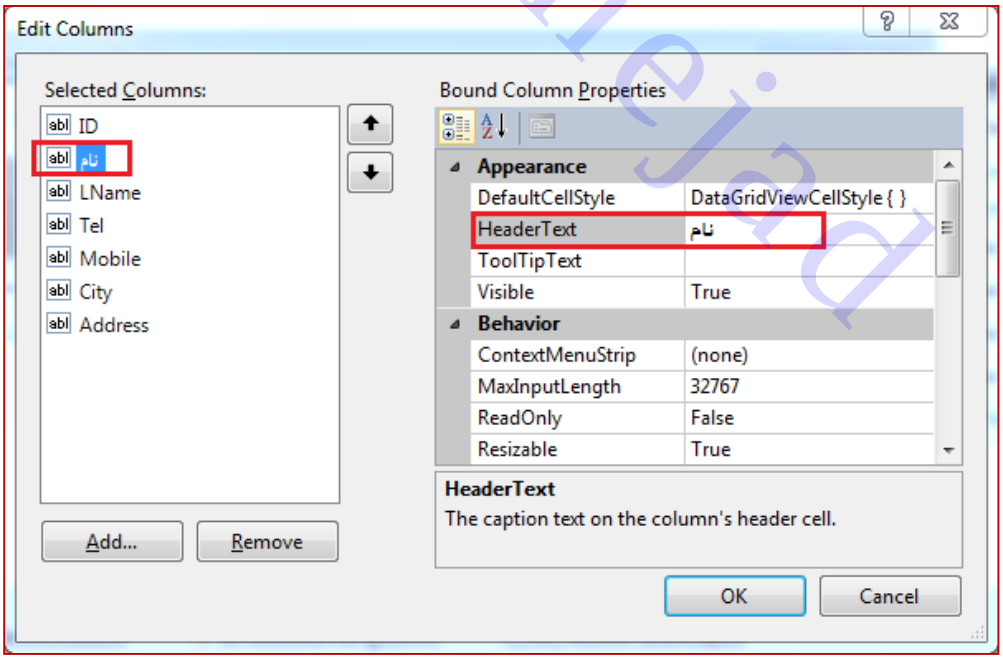


حال اگر برنامه را اجرا کنیم، خروجی زیر را خواهیم داشت:

۱۱)برای تنظیم عنوان ستونها(مثلاً به جای عنوان FName، عنوان «نام» نمایش داده شود)، علامت مثلث کوچک در کنترل GridView را زده و گزینه Edit Column را انتخاب نمایید.



سپس در قسمت سمت چپ پنجره نام فیلد یا ستون را انتخاب نموده و در بخش Header Text عنوان دلخواه خود را وارد نمایید. برای همه فیلدها این کار را انجام دهید. در قسمت Width هم می توانید اندازه عرض هر ستون را تنظیم نمایید.



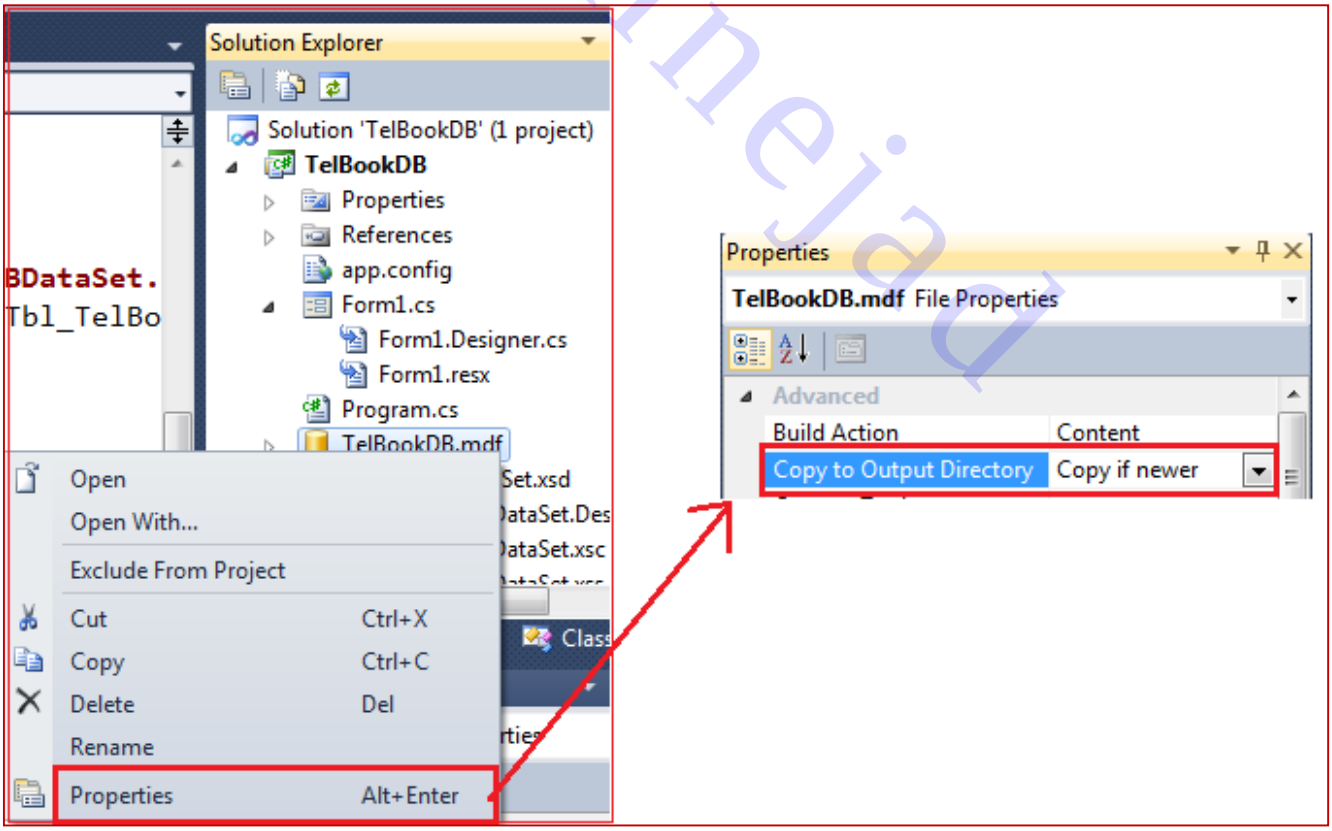
۱۲)در سطر آخر که علامت * وجود دارد می توانید اطلاعات سطر جدیدی درج نمایید.

نکته(۱) : برای ذخیره تغییرات باید متد Update از رابط TableAdapter را فراخوانی کنید. برای این منظور، دکمه ای با عنوان «ثبت تغییرات» به فرم اضافه نمایید و دستور زیر را در آن بنویسید.

```
private void btnSave_Click(object sender, EventArgs e)
{
    tbl_TelBookTableAdapter.Update(telBookDBDataSet.Tbl_TelBook);
}
```

Table Adaptor نام رابط.Update(DataSet نام . نام جدول);

نکته(۲) : برای اینکه در هر بار اجرا، فایل پایگاه داده قبلی(طراحی اولیه) جایگزین فایل پایگاه داده اطلاعات جدید نشود، باید تنظیمات زیر را انجام دهید. در پنجره Solution Explorer روی نام پایگاه داده کلیک راست نموده و گزینه Properties را انتخاب نموده و سپس مشخصه Copy to Output Directory را به مقدار Copy if newer تغییر دهید.



نکته(۳) : در رویداد Form_Load از فرم اصلی برنامه، متد Fill از شیئ TableAdapter، اطلاعات درون پایگاه داده به داخل حافظه و در شیء DataSet قرار می دهد.

```
private void Form1_Load(object sender, EventArgs e)
{
    this.tbl_TelBookTableAdapter.Fill(this.telBookDBDataSet.Tbl_TelBook);
}
```

مثال: فرمی طراحی نمایید که بر اساس نام وارد شده، در اطلاعات وارد شده در دفتر تلفن جستجو انجام داده و نتیجه را نمایش دهد.

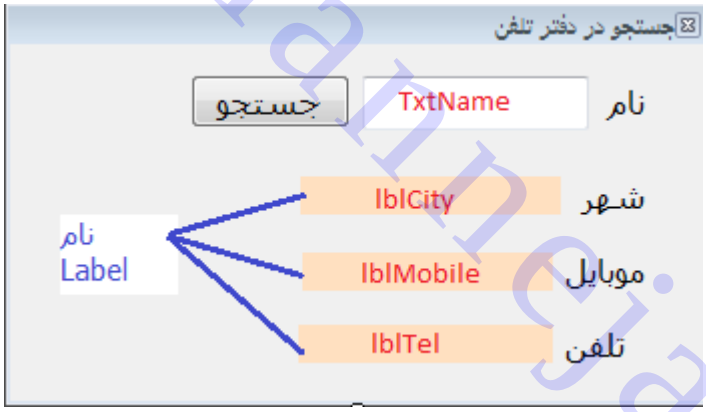
۱) فرم جدیدی به پروژه اضافه نمایید.

زدن دکمه Add→ انتخاب Windows Form → Add Windows Form منوی Project

یا

گزینه Add→ کلیک راست روی نام پروژه در Solution Explorer → Windows Form

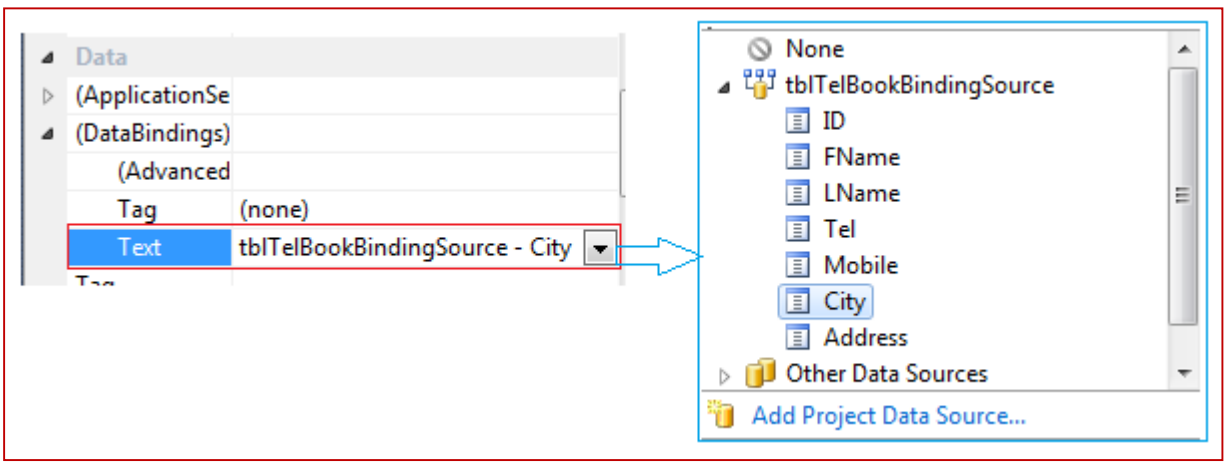
ویژگی FormBorderStyle فرم را به FixedToolWindow تغییر داده و مشخصات فونت و RightToLeft آنرا تنظیم نمایید. نام کنترلهای کادر متنی و برچسبها را مطابق شکل تنظیم کنید.



۲) در رویداد دکمه جستجو در فرم اول (Form1) دستورات زیر را بنویسید تا فرم دوم (Form2) را ظاهر سازد.

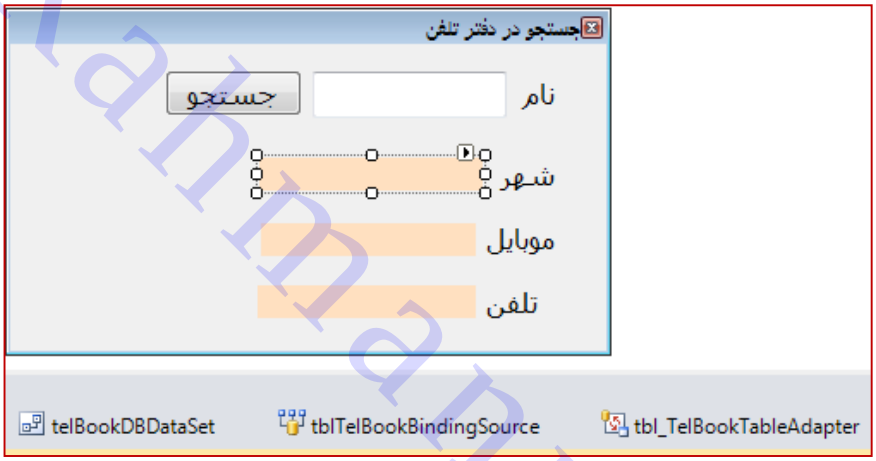
```
private void btnSearch_Click(object sender, EventArgs e)
{
    Form2 f2 = new Form2() ;
    f2.ShowDialog() ;
}
```

۳) حال باید کنترلهای برچسب را به فیلدهای بانک اطلاعاتی مرتبط نمایید. برای این کار در پنجره Properties و در گروه Data، علامت ▸ کنار مشخصه (DataBindings) را بزنید و با کلیک ▾ در کادر Text فیلد مربوطه را از جدول انتخاب کنید. مثلاً برای برچسب(Label) شهر(lblCity) فیلد City را انتخاب نمایید.



برای بقیه فیلدهای مورد نیاز همین کار را انجام دهید.

با انجام تنظیمات فوق برای برچسبها، بصورت خودکار شیئ های BindingSource، TableAdapter و DataSet به فرم اضافه می شوند.



۴) با اجرای برنامه، و زدن دکمه جستجو در فرم اول، فرم دوم باز می شود و مشخصات اولین رکورد در برچسب ها ظاهر می گردد. برای اینکه قبل از انجام جستجو، برچسبها چیزی نشان ندهند و به مقید شدن آنها به داده های بانک اطلاعاتی فعلاً به حالت تعلیق درآید، در متد سازنده مربوط به فرم دوم، باید دستور زیر را اضافه نمود:

```
public partial class Form2 : Form
{
    public Form2()
    {
        InitializeComponent();
        tblTelBookBindingSource.SuspendBinding();
    }
    :
    :
```

نکته ۴: متد SuspendBinding، مقید سازی کنترلها با داده های بانک اطلاعاتی توسط tblTelBookBindingSource را به حالت تعلیق در می آورد و فعلاً داده ها را نشان نمی دهد و با اجرای اول مقدار برچسبها، خالی خواهد بود.

۵) حال برای انجام جستجو، در رویداد دکمه جستجو در فرم دوم (Form2) دستورات زیر را بنویسید:

```
private void btnSearch_Click(object sender, EventArgs e)
{
    // انجام عمل جستجو براساس مقدار یک فیلد
    int find = tblTelBookBindingSource.Find("FName", TxtName.Text);
    if (find >= 0) // اگر پیدا شد
    {
        tblTelBookBindingSource.ResumeBinding(); // مقید سازی دوباره به داده ها
        tblTelBookBindingSource.Position = find; // نمایش رکورد یافت شده
    }
    else // یافت نشد
        tblTelBookBindingSource.SuspendBinding();// تعلیق مقید سازی به داده ها
}
```

شرح دستورات:

برای انجام جستجو بر اساس مقدار یک فیلد از دستور زیر استفاده می کنیم:

متغیر عددی = نام شیئ مقیدساز داده ها . Find("نام فیلد" , مقدار مورد جستجو);

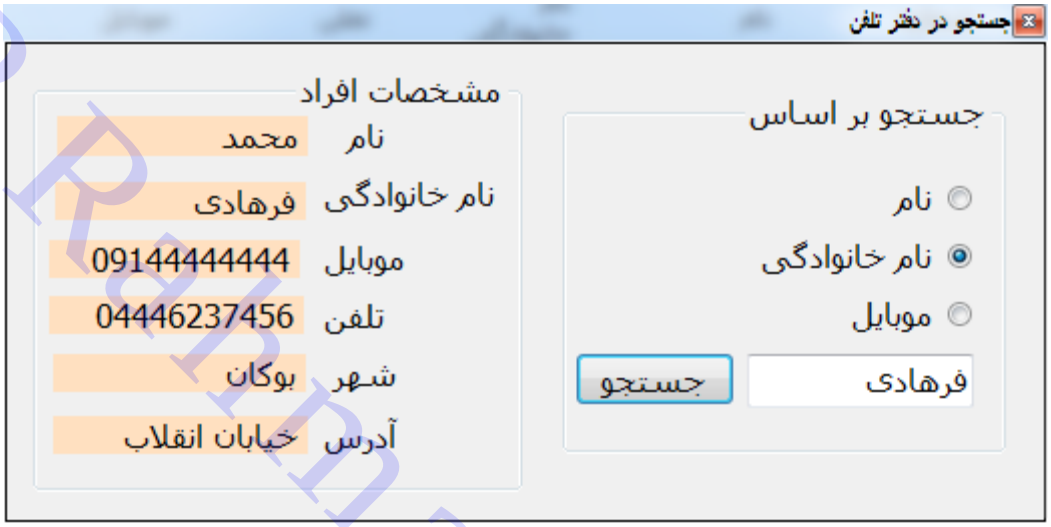
```
int find = tblTelBookBindingSource.Find("FName", TxtName.Text);
```

- در اینجا، نام فیلد مورد جستجو، FName یعنی نام است.
- مقدار مورد جستجو از کادر متنی TxtName تأمین می شود.
- نام شیئ مقیدسازی به داده ها در این مثال، *tblTelBookBindingSource* است.
- نتیجه جستجو، عددی صحیح است و نشانگر شماره اولین رکورد یافت شده در جدول است که مقدار فیلد FName با مقدار مورد جستجو یکسان است. عدد بزرگتر یا مساوی صفر به منزله یافت شدن رکورد می باشد. صفر به اولین رکورد اشاره دارد. اگر عدد ۱- برگردانده شود، به منزله این است که رکوردی با مشخصات داده شده پیدا نشده است.
- متد ResumeBinding( )، مقید سازی و اتصال مجدد به داده ها را برای کنترل های نمایش داده انجام می دهد.
- مشخصه Position اشاره گر رکوردها را تعیین می کند و دستور زیر اشاره گر را به محل یافت شدن(مقدار متغیر find)، منتقل می کند.

```
tblTelBookBindingSource.Position = find;
```

- دستور آخر بعد از else تعیین می کند که اگر رکورد با مشخصات جستجو یافت نشد، مقیدسازی را به حالت تعلیق درآورد و چیزی نمایش داده نشود.

تمرین: برچسبهای دیگری برای نام و نام خانوادگی و آدرس نیز به فرم اضافه کنید(مشابه تلفن و موبایل و شهر). سپس سه کنترل RadioButton (با عناوین نام، نام خانوادگی و موبایل)به فرم اضافه نمایید و دستورات جستجو را طوری تغییر دهید که براساس مورد انتخاب شده در RadioButton ها، جستجو انجام شود




```
private void btnSearch_Click(object sender, EventArgs e) // دکمه جستجو
{
    string fieldName= "";
    if (radFName.Checked)
        fieldName = "FName";
    else if(radLName.Checked)
        fieldName = "LName";
    else if (radMobile.Checked)
        fieldName = "Mobile";
    // انجام عمل جستجو براساس مقدار یک فیلد
    int find = tblTelBookBindingSource.Find(fieldName, TxtSearch.Text);
    if (find >= 0){ // اگر پیدا شد
        tblTelBookBindingSource.ResumeBinding(); // مقید سازی دوباره به داده ها
        tblTelBookBindingSource.Position = find; // نمایش رکورد یافت شده
    }
    else{  // یافت نشد
        MessageBox.Show("رکورد مورد نظر یافت نشد!");
        tblTelBookBindingSource.SuspendBinding();// تعلیق مقید سازی به داده ها
    }

}
```

نکته: برای انجام جستجو بر اساس یک فیلد که با وارد کردن بخشی از عنوان، نتایج را نشان دهد، می توان از خصوصیت Filter از شیئ مقید سازی داده ها(BindingSource)استفاده نمود:

نام شیئ مقیدساز داده ها . Filter = "نام فیلد LIKE ' %مقدار جستجو % ' " ;

دقت شود که در اول و آخر مقدار جستجو علامت ' ' قرار داده شده و علامت % جایگزین هر تعداد کارکتر در اول یا آخر می باشد.

- اگر علامت ٪ در اول قرار گیرد رکوردهایی را نمایش می دهد که مقدار فیلد به مقدار مورد جستجو ختم می شود.

tblTelBookBindingSource.Filter = " LName LIKE '%یان' "; → نام خانوادگی به «یان» ختم شود

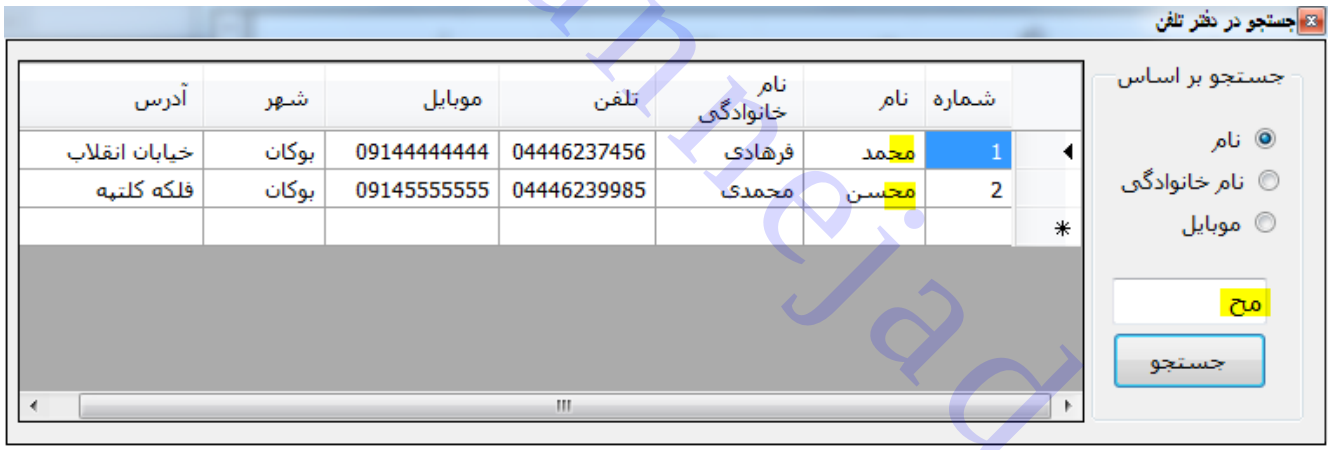
- اگر علامت ٪ در آخر قرار گیرد رکوردهایی را نمایش می دهد که مقدار فیلد با مقدار مورد جستجو شروع می شود.

نام خانوادگی با «مح» شروع شود → tblTelBookBindingSource.Filter = " LName LIKE 'مح٪' ";

- اگر علامت ٪ در اول و آخر قرار گیرد رکوردهایی را نمایش می دهد که مقدار مورد جستجو در هر جای مقدار فیلد قرار دارد.

نام خانوادگی شامل «مح» باشد → tblTelBookBindingSource.Filter = " LName LIKE '%مح٪' ";

در مثال زیر، بر اساس فیلد انتخابی و مقدار وارد شده، جستجو انجام می شود و فقط رکوردهایی را نمایش می دهد که شامل متن وارد شده باشند.




```
private void btnSearch_Click(object sender, EventArgs e) // دکمه جستجو
{
    string fieldName= "";
    if (radFName.Checked)
        fieldName = "FName";
    else if(radLName.Checked)
        fieldName = "LName";
    else if (radMobile.Checked)
        fieldName = "Mobile";
    tblTelBookBindingSource.Filter = fieldName + " LIKE "+ "'%"+TxtSearch.Text+"%'";
    tblTelBookBindingSource.ResumeBinding();
}
```

نکته ۲: برای انجام جستجو بر اساس یک فیلد که دقیقاً با مقداری مشخص یکسان باشد، و تمام رکوردهای مورد نظر را نمایش دهد، از خصوصیت Filter از شیئ مقید سازی داده ها(BindingSource) می توان استفاده کرد:

نام شیئ مقیدساز داده ها . Filter = "نام فیلد = 'مقدار جستجو' " ;

```
tblTelBookBindingSource.Filter = "FName  = 'محمد' "; // نمایش تمام کسانی که نامشان محمد است
```

منابع و مأخذ:

۱- یادداشت ها و تجربیات تدریس

۲- برنامه سازی (۱)،(۲)، (۳)- کربلایی مجید، دفتر تألیف کتاب های درسی فنی و حرفه ای و کاردانش، ۱۳۹۴، www.chap.sch.ir

۳- تولید محتوای الکترونیک و برنامه سازی،سازمان پژوهش و برنامه ریزی آموزشی، دفتر تألیف کتاب های درسی فنی و حرفه ای و کاردانش ۱۳۹۶، www.chap.sch.ir

۴- توسعه برنامه سازی و پایگاه داده، سازمان پژوهش و برنامه ریزی آموزشی، دفتر تألیف کتاب های درسی فنی و حرفه ای و کاردانش ۱۳۹۶، www.chap.sch.ir

4. Microsoft® Visual C#® 2010 Step by Step, John Sharp, Microsoft Press, 2010