نمره نهایی

- ✓امتحان میان‌ترم
- ✓امتحان عملی
- ✓تمرین
- ✓پروژه
- ✓امتحان پایان ترم
- ✓مطالب در سایت damrudi.ir

2014 *M. Damrudi*

C# به عنوان یک استاندارد جهانی

- C# در سال ۲۰۰۱ توسط شرکت مایکروسافت به همراه بسته Net.برای اولین بار مطرح و ارائه شد. که بعد ها توسط ISOو ECMA به عنوان یک زبان برنامه نویسی استاندارد مورد تائید قرار گرفت.
- انجمن تولید کنندگان کامپیوتر اروپا ECMA در سوم اکتبر سال ۲۰۰۱ زبان C# را به عنوان یک استاندارد پذیرفت.
- زبان برنامه‌نویسی C# توسط تیمی به مدیریت اندرز هایلزبرگ که قبلا تجربه ارائه زبان های برنامه‌نویسی موفقی همچون توربو پاسکال و دلفی را داشت، ایجاد شد.

2014 *M. Damrudi*

.Net Framework

- برنامه‌های تولید شده برای .Net Framework توسط Common Language Runtime اجراء می‌شوند.
- CLR یا همان Common Language Runtime سرویس‌هایی مهمی از قبیل Garbage Collection Exception Handling و Memory Management را ارائه می‌دهد.

2014 *M. Damrudi*

.Net framework

- با وجود Garbage Collection در زبان برنامه‌نویسی C# دیگر نگرانی از جهت مدیریت اشیاء در حافظه وجود ندارد به این معنی که وقتی شما شیء ایجاد می‌کنید بعد از اینکه دیگر از آن شی ء استفاده نکردید آن شیء به‌طور اتوماتیک از حافظه پاک خواهد شد. این کار توسط GC یا همان Garbage Collector انجام می‌شود.

2014 *M. Damrudi*

.Net framework

- روش کار GC به این ترتیب است که تا موقعی که Referenceی به یک object وجود داشته باشد آن شیء در حافظه باقی خواهد ماند اما در صورتیکه Referenceی به آن شیء وجود نداشته باشد بعد از یک بازه زمانی نامشخص آن شیء به صورت اتوماتیک از حافظه پاک خواهد شد.

2014 *M. Damrudi*

.Net Framework

- برخلاف زبان های برنامه نویسی دیگر وقتی کد شما در زبان برنامه‌نویسی C#(یا هریک از زبان های دیگر .Net) کامپایل می‌شود به یک زبان دیگری به نام Language Intermediate یا همان IL تبدیل می‌شود.
- در موقعی که درخواست برای اجراء آن داده می‌شود توسط یک مکانیزمی به نام Just In Time Compiler که در CLR موجود است به کد اجرایی خاص آن ماشین تبدیل شده و اجراء می‌گردد.

2014 *M. Damrudi*

C#

- C# یک زبان شی‌گرا است.
- C# یک زبان Case-sensitive است.
- ترکیب زبان C# شبیه زبان C++ و java است.

2014 *M. Damrudi*

فایل‌های پروژه C#

- ⇦ code and text Editor محتوای فایل‌های source را نشان می‌دهد.
- ⇦ Solution Explorer نام فایل‌های پروژه را نشان می‌دهد.
- ⇦ هر solution شامل یک یا چند پروژه است. این فایل‌ها با پسوند sln. ذخیره می‌شوند.
- ⇦ هر فایل پروژه یک یا چند فایل دارد. این پروژه‌ها با پسوند csproj. ذخیره می‌شوند.

2014 *M. Damrudi*

فایل‌های پروژه #C

⇦ فولدر Properties شامل فایلی به نام AssemblyInfo.cs است. این فایل، فایل مخصوصی است که می‌توانید از آن برای اضافه کردن صفت‌ها (Attribute) به یک برنامه استفاده کنید. این صفت‌ها شامل:

⇦ نام نویسنده

⇦ تاریخ نوشتن برنامه

⇦ version برنامه

⇦ ....

2014 *M. Damrudi*

فایل‌های پروژه #C

⇦ فولدر References شامل کدهای گردآوری شده است که برنامه می‌تواند از آنها استفاده کند.

⇦ فایل Program.cs فایل C# Source است. هنگامی‌که پروژه‌ای اولین بار ایجاد می‌شود محتوای این فایل در پنجره Code and Text Editor نمایش داده می‌شود. در این فایل کدهایی از قبل وجود دارند که به صورت خودکار ایجاد شده‌اند. کافی است کدهای برنامه خود را در بین این کدها قرار دهید.

2014 *M. Damrudi*

تکنولوژی IntelliSense

- لیستی شامل تمام کلمات کلیدی و انواع داده‌های معتبر C# است. به جای تایپ کردن می‌توان از این لیست کلمه مورد نظر را پیدا کرد.
- IntelliSense نام هر عضو از کلاس را نمایش می‌دهد و در سمت چپ آن آیکن‌هایی قرار دارد که نوع عضو را مشخص می‌کند.

2014 *M. Damrudi*

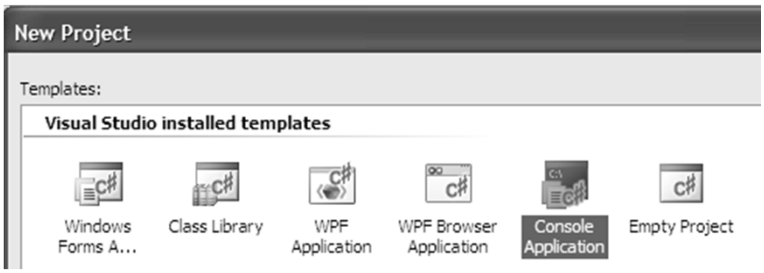
تکنولوژی IntelliSense

آیکون	معنی
	کلمه کلیدی
	متد
	خاصیت
	کلاس
	Struct
	Enum
	Interface
	Delegate
	NameSpace

2014 *M. Damrudi*

اولین برنامه C#

- یک پروژه جدید ایجاد کنید.



- Console Application محیطی شبیه DOS و بدون طراحی GUI می‌باشد.

2014 *M. Damrudi*

اولین برنامه C#

◈ در برنامه‌های بزرگ دو مشکل وجود دارد. اولا درک و نگهداری کدها مشکل می‌شود، دوم اینکه نام‌ها، داده‌ها، متدها و کلاس‌ها بیشتر می‌شوند، namespace این مشکل را حل می‌کند.

◈ namespace دسته‌بندی برای کلاس‌ها و مدیریت کدنویسی است. برای نوشتن کلاس‌ها مجبوریم نام کلاس را به‌همراه namespace مربوطه آن‌ها بنویسیم. می‌توان با یک‌بار نوشتن نام namespace مربوط به کلاس با استفاده از using از تکرار آن جلوگیری کرد.

◈ عبارت using یک namespace را در یک scope قرار می‌دهد.

2014 *M. Damrudi*

اولین برنامه C#

- کدهای زیر به صورت خودکار نوشته شده‌اند:

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

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

2014 *M. Damrudi*

اولین برنامه C#

- برای نمایش اطلاعات در خروجی از متد Writeline از کلاس Console استفاده می‌شود.

```
System.Console.Writeline ("hi");
```

namespace Class method

- اگر با استفاده از using شیء System در scope قرار داده شود (در ابتدای برنامه) برای نمایش یک متن کافی‌است بنویسیم:

```
Console.Writeline("hi");
```

2014 *M. Damrudi*

اولین برنامه C#

- WiteLine و ReadLine (دریافت اطلاعات از ورودی) جزء توابع کتابخانه‌ای کلاس System می‌باشند.
- اسم namespace و class را می‌توان به اختیار تعیین کرد. تمام برنامه‌ها در داخل namespace و class نوشته می‌شود تا بتوانیم به راحتی به قسمتهای مختلف برنامه دسترسی داشته باشیم.
- اگر به عنوان مثال نام‌های تکراری برای کلاس در دو namespace جدا داشته باشیم باید نام کامل شناسایی را به‌کار بریم تا مشکلی ایجاد نشود.

2014 *M. Damrudi*

اولین برنامه C#

- C# تمام کارش با کلاس‌هاست. بنابراین باید در تعریف و استفاده از آنها دقت کافی داشت.
- بدون متد Main برنامه‌های C# قادر به اجرا نخواهند بود.
- هر متد باید مقداری را برگرداند. نوع متد در اینجا voidتعریف شده است یعنی هیچ مقداری را برنمی‌گرداند. اگر برنامه حاوی متد Main نباشدبعنوان توابع سیستمی همانند dllهای ویندوز درنظر گرفته می‌شود.

2014 *M. Damrudi*

اولین برنامه #C

- متدها پارامترهایی را در پرانتز جلوی نام متد دارند. متد Main در این مثال یک پارامتر args از نوع آرایه رشته‌ای دارد.
- Methodها همان رفتارهایی هستند که ما در شئ‌گرایی از کلاس ها انتظار داریم.
- تمام فضاهای نام به‌صورت پیش‌فرض public هستند و در خارج از آن قابل دسترسی هستند. نحوه استفاده از آن‌ها به‌صورت زیر می‌باشد.
- ProjectName.namespace.ClassName.MemberName

2014 *M. Damrudi*

دومین برنامه #C

- یک پروژه جدید ایجاد کنید.
- این بار Windows Forms Application را انتخاب کنید.
- در این نوع برنامه می‌توان با ابزار موجود واسط کاربر گرافیکی ایجاد کرد.
- در این حالت دو view وجود دارد: Design view و code view
- یک Button، یک Text boxو یک label روی فرم قرار دهید.
- با استفاده از Properties می‌توان خواص کنترل‌ها را مشاهده یا تغییر داد.

2014 *M. Damrudi*

دومین برنامه #C

- خاصیت Text دکمه را به OK و خاصیت Text مربوط به label را به :Enter your age تغییر دهید.
- در solution Explorer فایل form1 را انتخاب کرده و با راست کلیک view code را انتخاب کنید تا source آن ظاهر شود.
- نام پروژه به‌عنوان namespace تعیین می‌شود.
- نام فرم به‌عنوان class تعیین می‌شود.

2014 *M. Damrudi*

دومین برنامه #C

- متدی به نام Initialize component برای Form1 نوشته شده است که خواص کنترل‌هایی را که ما به فرم اضافه کرده‌ایم تعیین می‌کند.
- در فایل Program.cs نیز کدهایی نوشته شده است، از آن جمله سه عملیات زیر هستند که مربوط به namespace System.Windows.Forms می‌باشند.

Application.Run(new Form1)؛ ایجاد و نمایش فرم

```
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);
```

2014 *M. Damrudi*

دومین برنامه C#

```
private void btn1_Click(object sender, EventArgs e)
{
    MessageBox.Show("hello " + txtbox1.Text + " years old");
}
```

- MessageBox کلاسی است که متد show از آن، برای نمایش پیغام به کار می‌رود.
- در این برنامه کاربر سن خود را در کادر متن نوشته و دکمه OK را کلیک می‌کند و در مقابل پیغام زیر نمایش داده می‌شود (با فرض ورود ۱۹ برای سن):
- Hello 19 years old
- برای استفاده از متن کادر متن کافیست از خاصیت Text آن استفاده نمود.

2014 *M. Damrudi*

انواع داده در C#

نوع داده	اندازه	مثال
int	۳۲	test1=42;
long	۶۴	test2=42L;
float	۳۲	test3=0.33F;
double	۶۴	test4=0.54;
decimal	۱۲۸	test5=0.45M;
string	۱۶ بیت در هر کاراکتر	test6=77;
char	۱۶	test7=4;
bool	۸	test8=false;

2014 *M. Damrudi*

متد Tostring

- کنترل‌های Text box خاصیتی به‌نام Text دارند که به شما امکان می‌دهد تا محتویات آن‌ها را دستیابی کنید.
- Tostring برای تبدیل مقادیر به نوع داده string معادل به کار می‌رود.
- برای تبدیل انواع داده به یکدیگر از کلاس Convert استفاده می‌شود.
- System.Int32.Parse(): متد Parse از کلاس System.Int32 یک پارامتر رشته‌ای را می‌گیرد و آن را به مقدار int تبدیل می‌کند.

2014 *M. Damrudi*

عملگرها در C#

- عملیات + ، -، *، /، %، ++، -- مشابه همان عملیات در C++ است.
- رفتار عملگرهای ترکیبی نیز همانند C++ است. به عنوان مثال:

Test=Test+1 ⇨ Test+=1

2014 *M. Damrudi*

متد در C#

❖ دستور زبان متد در C# به صورت زیر است:

```
returnType  methodName(parameters)
{
// body statements
return ;
}
```

✍ returnType: تعیین می‌کند چه نوع اطلاعاتی توسط متد بازگردانده می‌شود. اگر متد مقداری را بازنمی‌گرداند از نوع void تعریف می‌شود.

✍ methodName: نام متد که برای فراخوانی متد است.

✍ parameters: نام و نوع مقادیر ارسالی به متد می‌باشند.

2014 *M. Damrudi*

متد در C#

✍ تمام متدها در داخل کلاس‌ها تعریف می‌شوند.

✍ مقادیر بازگشتی توسط دستور return برگشت داده می‌شوند.

✍ اگر متد مقداری را بازنمی‌گرداند می‌توان دستور return را حذف کرد.

✍ می‌توان برای خارج شدن فوری از متد از دستور return استفاده کرد.

✍ در هر فراخوانی متد باید از پرانتز استفاده شود حتی اگر هیچ آرگومانی نداشته باشد.

2014 *M. Damrudi*

متد در C#

```
int result(int a, int b)
{
return a+b;
}
```

متدها در C# مشابه C++ است.

Overload کردن متد در C#

- می‌توان متدهای هم‌نام با تعداد پارامترهای متفاوت و یا نوع پارامتر متفاوت داشت. به این عمل سربارگذاری (Overloading) گفته می‌شود.
- این عمل اصولا زمانی مفید است که نیاز به اجرای عملیات مشابه روی نوع داده‌های متفاوت داشته باشید.
- ترکیب تعداد و نوع پارامترهای متد را signature گویند.
- نمی‌توان نوع بازگشتی متد را Overload کرد به این معنی که نمی‌توان دو متد هم نام داشت که فقط در نوع بازگشتی بایکدیگر متفاوت باشند.

Scope در C#

- {} در یک متد، محدوده آن را مشخص می‌کند و هر متغیری در درون این بدنه تعریف شود، در محدوده آن متد قرار می‌گیرد.
- این متغیرها تا زمانی وجود دارند که متد پایان بپذیرد و تنها کدهای درون آن متد می‌توانند از آن استفاده کنند. این متغیرها را محلی می‌نامند.
- نمی‌توان از متغیرهای محلی برای به اشتراک گذاشتن اطلاعات بین متدها استفاده کرد.

2014 *M. Damrudi*

Scope در C#

```
class test
{
        void firstMethod()
        {
                int t1;
        }
        void secondMethod()
        {
                t1=66;
        }
}
```

Not in scope

2014 *M. Damrudi*

Scope در C#

- {} در یک کلاس، محدوده آن را مشخص می‌کند و هر متغیری در درون این بدنه تعریف شود، در محدوده آن کلاس قرار می‌گیرد.
- متغیرهای تعریف‌شده به‌وسیله‌کلاس، فیلد نامیده می‌شوند.
- می‌توان از فیلدها برای به اشتراک گذاشتن اطلاعات بین متدها و کلاس‌ها استفاده کرد.
- در یک متد باید یک متغیر را قبل از استفاده تعریف کرد ولی در مورد فیلدها این‌گونه نیست. متد می‌تواند از یک فیلد قبل از دستوری که آن فلید را تعریف می‌کند، استفاده کند.

2014 *M. Damrudi*

Scope در C#

```
class test
{
        void firstMethod()
        {
                t1=55;
        }
        void secondMethod()
        {
                t1=45;
        }
        int t1=0;
}
```

2014 *M. Damrudi*

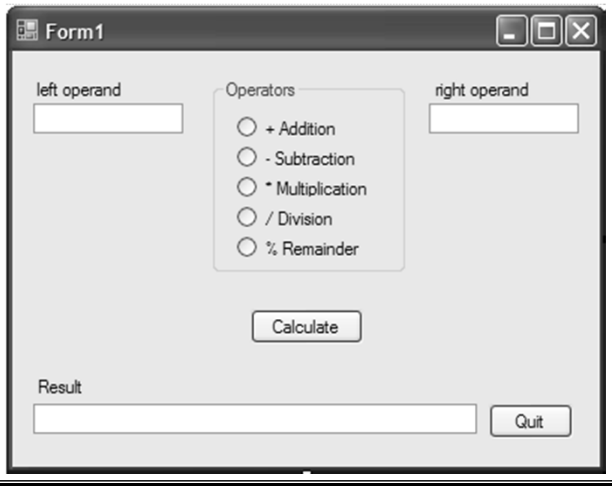
if در C#

✍ if در C# مشابه C++ است. در صورتی که تعداد دستورات بیش از یک دستور باشد برای مشخص کردن محدوده دستورات از {} استفاده می‌شود.

```
if ( booleanExpression)
        statement1;
else
        statement2;
```

2014 *M. Damrudi*

مثال



2014 *M. Damrudi*

مثال

```
namespace Methods
{
   partial class Form1 : Form
{
      public Form1()
        {
         InitializeComponent();
        }

      private void calculate_Click(object sender, System.EventArgs e)
        {
         int calculatedValue = 0;

            int X1 = System.Int32.Parse(leftOperand.Text);
            int X2 = System.Int32.Parse(rightOperand.Text);
```


مثال

```
if (addition.Checked)
              {
               calculatedValue = addValues(X1, X2);
               showResult(calculatedValue);
              }
     else if (subtraction.Checked)
               {
               calculatedValue = subtractValues(X1, X2);
               showResult(calculatedValue);
               }
    else if (multiplication.Checked)
               {
               calculatedValue = multiplyValues(X1, X2);
               showResult(calculatedValue);
               }
```

مثال

```
    else if (division.Checked)
              {
               calculatedValue = divideValues(X1, X2);
               showResult(calculatedValue);
              }
     else if (remainder.Checked)
              {
               calculatedValue = remainderValues(X1, X2);
               showResult(calculatedValue);
              }
}
private int addValues(int X1, int X2)
{
        return X1 + X2;
}
```

2014 *M. Damrudi*

مثال

```
private int subtractValues(int X1, int X2)
       {
         return X1 - X2;
       }
private int multiplyValues(int X1, int X2)
      {
         return X1 * X2;
       }
private int divideValues(int X1, int X2)
      {
         return X1 / X2;
      }
```

2014 *M. Damrudi*

مثال

```
private int remainderValues(int X1, int X2)
       {
         return X1 % X2;
       }
private void showResult(int answer)
       {
         result.Text = answer.ToString();
        }
 private void quit_Click(object sender, System.EventArgs e)
       {
         Application.Exit();
       }
    }
}
```

2014 *M. Damrudi*

Switch در#C

✍ اگر دستوراتِ if مشابه داشته باشیم می‌توان از switch استفاده کرد.

```
switch(variable)
{
case  constant1;
          statement1;
          break;
…
default:
          statements;
          break;
}
```

مشابه C++

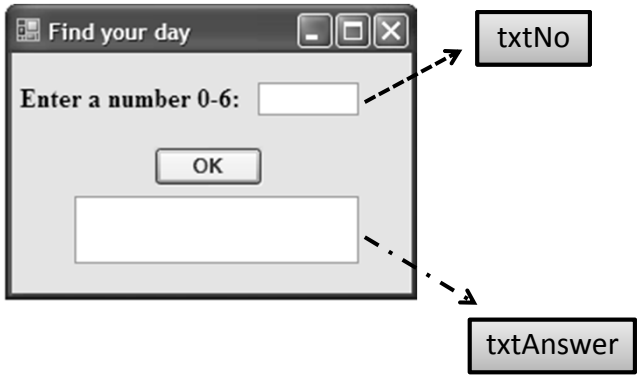
2014 *M. Damrudi*

switch در C#

- برای هر case نیاز به یک break داریم.
- دستور default اختیاری است.
- اگر هیچ‌یک از مقادیر caseها با variable برابر نشد دستور بعد از default اجرا می‌شود.
- دستور switch را تنها می‌توان روی char و int و string استفاده کرد.
- مقدار بعد از case باید مقدار ثابتی باشد.
- دو case نمی‌توانند مقدار مشابهی داشته باشند.

2014 M. Damrudi

مثال



2014 M. Damrudi

مثال

```
using System;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
   public partial class frmMain : Form
{
      public frmMain()
       {
         InitializeComponent();
       }
      private void btnDay_Click(object sender, EventArgs e)
       {
         int x;
         string s;
         x=System.Int32.Parse(txtNo.Text);
```


مثال

```
switch (x)
{
            case 0:
               s = "saturday";
               break;
            case 1:
               s = "sunday";
               break;
            case 2:
               s = "monday";
               break;
            case 3:
               s = "tuesday";
               break;
            case 4:
               s = "wednesday";
               break;
            case 5:
               s = "thursday";
               break;
            case 6:
               s = "friday";
               break;
            default:
               s = "wrong day";
               break;
         }
         txtAnswer.Text = s.ToString();
         }
      }
   }
```

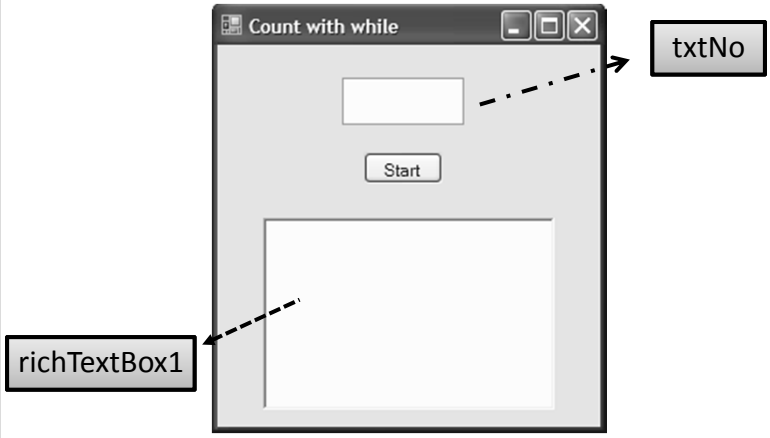
while در C#

✍ دستورات تا زمانیکه عبارت شرطی true باشد اجرا خواهند شد.

```
while(Expression)
{
        statements;
}
```

2014 M. Damrudi

مثال



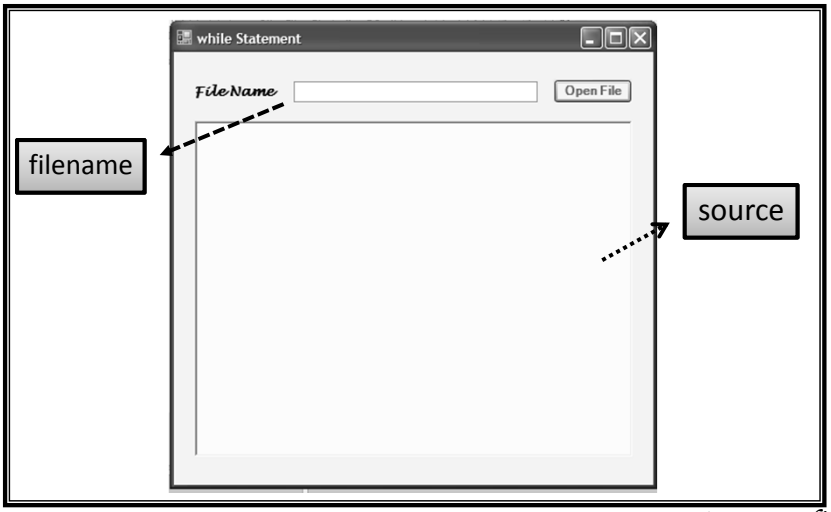
2014 M. Damrudi

مثال

```
private void button1_Click(object sender, EventArgs e)
{
        int x = 0;
        int y=System.Int32.Parse(txtNo.Text);
        while (x != y)
         {
           richTextBox1.Text+= x.ToString() + "\n";
           x++;
         }
}
```

2014 M. Damrudi

مثال



2014 M. Damrudi

مثال

```
using System;
using System.IO;
using System.Windows.Forms;
namespace WhileStatement
{
   partial class Form1 : Form
}
      public Form1()
       {
         InitializeComponent();
       }
      private void openFile_Click(object sender, System.EventArgs e)
       {
         openFileDialog.ShowDialog();
       }
```


مثال

```
private void openFileDialog_FileOk(object sender, System.ComponentModel.CancelEventArgs e)
      {
         string fullPathname = openFileDialog.FileName;
         FileInfo src = new FileInfo(fullPathname);
         filename.Text = src.Name;
         source.Text = "";
         TextReader reader = src.OpenText();
         string line = reader.ReadLine();
         while (line != null)
          {
            source.Text += line + "\n";
            line = reader.ReadLine();
          }
         reader.Close();
         }  }  }
```

مثال

- ShowDialog(): یک جعبه گفتگو را باز می‌کند.
- OpenFileDialog: می‌تواند جعبه گفتگویی برای بازکردن فایل ایجاد کند.
- FileInfo: کلاسی را برای ایجاد، حذف، کپی، حرکت دادن و بازکردن فایل‌ها فراهم می‌کند.
- TextReader: این کلاس اشیائی ایجاد می‌کند که می‌تواند مجموعه ترتیبی از کاراکترها را از منابعی مانند فایل‌ها بخواند.
- OpenText(): فایل متنی را برای خواندن باز می‌کند و یک object از نوع TextReader برمی‌گرداند.
- ReadLine(): یک خط از کاراکترها را از جریان ورودی می‌خواند.

2014 *M. Damrudi*

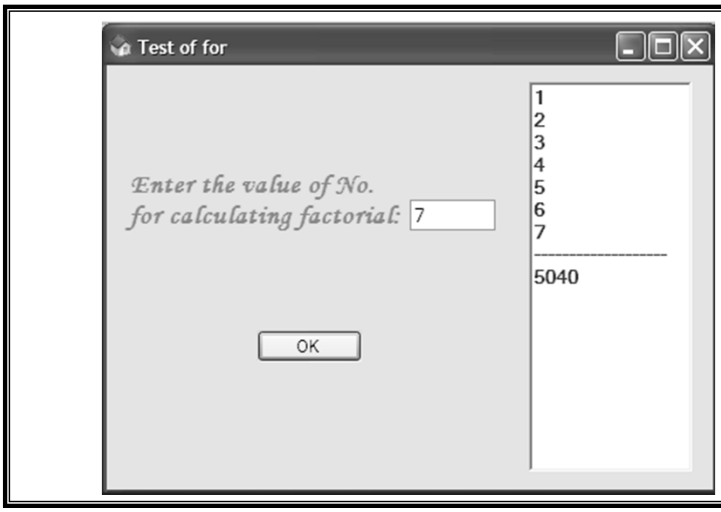
for در C#

- initialization فقط یک‌بار در شروع حلقه انجام می‌پذیرد.
- پس از اجرای دستورات متغیر کنترلی تغییر می‌کند و شرط چک می‌گردد.
- تا زمانیکه شرط درست باشد دستورات اجرا می‌شوند.

```
for(initialization;Boolean expression; increment/decrement)
{
        statements;
}
```

2014 *M. Damrudi*

مثال



2014 M. Damrudi

مثال

```
private void btnCal_Click(object sender, EventArgs e)
{
        rtxtAns.Text="";
        int factorial=1;
        int no = Int32.Parse(txtValue1.Text);
        for (int i = 1; i <= no; i++)
          {
           factorial = factorial * i;
           rtxtAns.Text += i + "\n";
          }
        rtxtAns.Text += "--------------------\n";
        rtxtAns.Text += factorial.ToString();
}
```

2014 M. Damrudi

do در C#

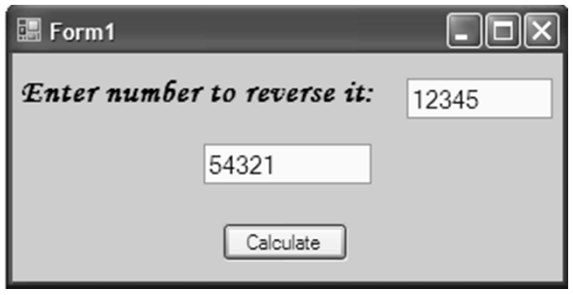
✍ دستور while و for هردو عبارت شرطی خود را در شروع حلقه چک می‌کنند. اگر false باشد بدنه اجرا نمی‌شود.

✍دستور do عبارت شرطی را در پایان حلقه بررسی می‌کند و بدنه حلقه حداقل یکبار اجرا می‌شود.

```
do
{
        statements;
}
while (booleanExpression);
```

2014 *M. Damrudi*

مثال



2014 *M. Damrudi*

مثال

```
private void btnCalculate_Click(object sender, EventArgs e)
{
        txtAnswer.Text = "";
        int m=0,a,b;
        a = Int32.Parse(txtNo.Text);
        do
         {
           b = a % 10;
           m =m* 10 + b;
           a /= 10;
         }
        while (a > 0);
        txtAnswer.Text = m.ToString();
}
```

2014 *M. Damrudi*

break و continue در C#

✍ break برای خروج از بدنه حلقه استفاده می‌شود.

✍ continue برای اجرای دور بعدی حلقه به کار می‌رود به‌صورتی دستورات بعد از continue اجرا نمی‌شوند.

```
int  x=0;
while (true)
{
Console.WriteLine(x);
x++;
if (x!= 5)
        continue;
else
        break;
}
```

output:
0
1
2
3
4

2014 *M. Damrudi*

break و continue در C#

✍ break برای خروج از بدنه حلقه استفاده می‌شود.

✍ continue برای اجرای دور بعدی حلقه به کار می‌رود به‌صورتی دستورات بعد از continue اجرا نمی‌شوند.

```
int  x=0;
while (true)
{
  x++;
  if (x!= 5)
        continue;
  else
        Console.WriteLine(x);
        break;
}
```

output:
5

2014 *M. Damrudi*

تمرین

برنامه‌های زیر را با استفاده از Console Application بنویسید.

۱- برنامه‌ای بنویسید که یک عدد را دریافت کند و تعداد ارقام زوج آن را نشان دهد.

۲-برنامه‌ای بنویسید که ضرایب یک معادله دو مجهولی را دریافت کند، ریشه‌های آن را در صورت وجود نمایش دهد.

۳- برنامه‌ای بنویسید که ۲۰ عدد را دریافت کرده max، min و میانگین اعداد را نمایش دهد.

2014 *M. Damrudi*

خطا در C#

☼ به دلایل مختلفی ممکن است خطا رخ دهد. برنامه‌ها باید قابلیت تشخیص و کنترل خطا را داشته باشند.

☼ C# با استفاده از استثنا و اداره کننده استثنا، کدی که برنامه اصلی را پیاده سازی می‌کند را از کد مدیریت خطا جدا می‌کند.

☼از دستورات try و catch برای تشخیص و مدیریت خطا استفاده می‌شود.

2014 *M. Damrudi*

خطا در C#

☼ تمام دستورات اجرایی را در یک بلوک try قرار دهید.

☼ اگر دستورات بدون خطا اجرا شوند، اتفاقی رخ نمی‌دهد.

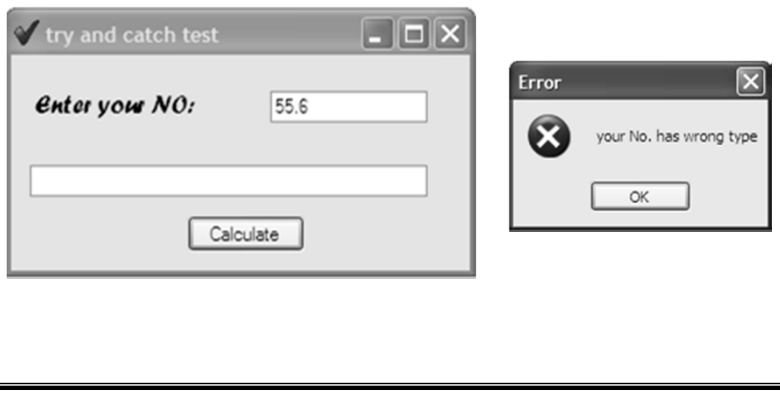
☼ اگر خطایی رخ دهد، اجرا به خارج از بلوک try و به یک اداره کننده catch پرش می‌کند.

☼ یک یا چند اداره‌کننده catch برای مدیریت خطا پس از بلوک try نوشته می‌شود.

☼ اگر هریک از دستورات درون try باعث ایجاد خطا شوند، runtime یک استثنا را تولید می‌کند. سپس اداره‌کننده‌های catch پس از try را بررسی می‌کند و کنترل را به یک اداره کننده متناظر با خطا انتقال می‌دهد.

2014 *M. Damrudi*

مثال



مثال

```
private void btnCalc_Click(object sender, EventArgs e)
{
        try
          {
           int x = Int32.Parse(txt1.Text);
           if (x % 2 == 0)
              txtans.Text = "The Number is Even";
           else
              txtans.Text = "The Number is Odd";
          }
        catch
          {
           System.Windows.Forms.MessageBox.Show ("your No. has
           wrong type","Error",MessageBoxButtons.OK
           ,MessageBoxIcon.Error);
          }
}
```

انواع Exception در#C

☼در مثال قبل از آنجایی که برای catch ، Exception خاصی را قرار ندادیم، حالت عمومی دارد. برای دسترسی به جزئیات بیشتر در مورد خطا، در حالت کلی هم بهتر است به صورت زیر عمل کرد تا بتوان جزئیات خطای ایجاد شده را مشاهده کرد.

☼ فیلد Message از Exception شرح متنی خطارا برمی‌گرداند.

```
catch (Exception t)
{
 System.Windows.Forms.MessageBox.Show(t.Message,
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

2014 *M. Damrudi*

انواع Exception در#C

☼می‌توان اداره‌کننده catch را با نوع خطایی که رخ می‌دهد مشخص کرد. درصورتی که بخواهید مدیریت خطای دقیق‌تری را انجام دهید از این روش استفاده کنید.

☼ در مثال قبل می‌توان catch های مختلفی داشت. به عنوان مثال یک catch برای اینکه اگر کاربر فرمت عدد را اشتباه وارد کرد(عدد اعشاری) و یک catch برای اینکه اگر عدد وارد شده بزرگتر از نوع داده تعیین شده باشد.(اسلاید بعد)

☼ اگر بیش از یک Exception وجود داشته باشد که مطابق با خطای رخ داده شده باشند. هر کدام که بالاتر باشند اجرا خواهند شد و بقیه نادیده گرفته می‌شوند. بهتر است catchهای مخصوص بالاتر نوشته شوند.

2014 *M. Damrudi*

انواع Exception در#C

```
catch (FormatException  f)
{
System.Windows.Forms.MessageBox.Show(f.Message + "\n
Enter an integer", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
}
catch (OverflowException  o)
{
System.Windows.Forms.MessageBox.Show(o.Message + "\n
Enter smaller No.", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
}
```

2014 *M. Damrudi*

checked و unchecked در#C

☼ انواع داده دارای محدوده ثابتی هستند که بیشتر یا کمتر از محدوده مورد نظر پاسخ اشتباه به شما می‌دهد.

☼ int دارای محدوده ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷ است و اگر مقدار نتیجه محاسبه شما بیش از مقدار فوق باشد پاسخ اشتباه (منفی) تولید می‌شود و در واقع overflow رخ داده است.

☼ از آنجایی که عملیات ریاضی رایج‌ترین عملیات است و چک کردن overflow برای هر محاسبه کارایی را پایین می‌آورد، اغلب ریسک را می‌پذیرند و آن را چک نمی‌کنند.

2014 *M. Damrudi*

checked و unchecked در C#

☼می‌توانید تنظیمات چک کردن سرریز را به روش زیر فعال کنید:

☼Project ➔ ? Properties ➔ Build ➔ Advanced ➔ check for arithmetic overflow/underflow

☼ می‌توانید از کلمات checked و unchecked برای فعال یا غیرفعال کردن چک کردن سرریز در هر قسمت استفاده کرد.

☼ اگر می‌خواهید ماکزیمم و مینیمم int را مشاهده کنید از Int32.Minvalue و Int32.Maxvalue استفاده کنید.

2014 *M. Damrudi*

checked و unchecked در C#

❖ در مثال زیر یک overflowException رخ می‌دهد و دیگر پاسخ اشتباه نمی‌دهد.

```
int x = Int32.MaxValue;
checked
{
    int y = ++x;
    Console.WriteLine(y);
}
```

❖تنها عملیات محاسباتی موجود در بلوک چک می‌شوند و درصورتی که فراخوانی متد وجود داشته باشد، چک در داخل متد انجام نمی‌گیرد. اگر unchecked جای checked قرار داده شود دیگر کنترل صورت نمی‌گیرد.

2014 *M. Damrudi*

checked و unchecked در C#

❖ توجه داشته باشید روش بررسی overflow ذکر شده فقط روی اعداد صحیح انجام‌پذیر است.

❖ می‌توان از checked و unchecked در عبارات محاسباتی (-،+، *، /، ++، --)نیز استفاده کرد.

```
int x = 2147483647;
int y = 7;
int z1 = checked(x + y);
Console.WriteLine(z1);
int z2 = unchecked(x + y);
Console.WriteLine(z2);
```

2014 *M. Damrudi*

ایجاد استثنا در C#

❖ اگر متدی برای ماه داشته باشید برای اعداد غیر از ۱ تا ۱۲ نباید مقداری بازگرداند. دراین حالت متد باید یک استثنا ایجاد کند.

❖ کتابخانه کلاس .NET Framework شامل بسیاری از کلاسهای استثنا است که می‌توان از آنها استفاده کرد.

❖ در مثال بعد اگر عددی غیر از ۱تا ۱۲ وارد شود یک Exception خارج از محدوده رخ می‌دهد و پیغام مناسب را نمایش می‌دهد.

❖ اگر تابعی که در یک کلاس فراخوانی می‌شود از نوع static تعریف نشود، حتما باید هنگام فراخوانی تابع آن را new کرد.

2014 *M. Damrudi*

مثال

```
static void Main(string[] args)
{
        int i;
        i = int.Parse(System.Console.ReadLine());
        try
         {
          System.Console.WriteLine(monthfind(i));
          }
        catch (Exception e)
          {
           Console.WriteLine(e.Message);
          }
}
```

System.Console.WriteLine((new Program()).monthfind(i));

تابع static نیاز ندارد.

2014 M. Damrudi

مثال

```
public static string monthfind(int month)
{
          switch (month)
 {
             case 1:
                return "january";
                break;
             case 2:
                return "februrary";
                break;
            . . .
             case 12:
                return "december";
                break;
             default:
                throw new ArgumentOutOfRangeException("bad no");
} }
```

2014 M. Damrudi

استفاده از بلوک finally در C#

- همیشه نمی‌توان تضمین کرد که یک دستور اجرا شود زیرا ممکن است دستور قبلی یک Exception ایجاد کند.
- برخی مواقع اگر یک دستور ویژه اجرا نشود مشکل بزرگی به وجود می‌آید.
- مثال خواندن متن فایل را در نظر بگیرید اگر منابع باز شده را نبندید ممکن است پس از مدتی نتوانید فایل‌های بیشتری باز کنید.
- برای آن‌که یک دستور همیشه اجرا شود(چه Exception اتفاق افتد چه اتفاق نیفتد)آن را درون یک بلوک finally بنویسید.

2014 *M. Damrudi*

استفاده از بلوک finally در C#

- بلوک finally پس از try یا پس از آخرین catch قرار می‌گیرد.
- حتی اگر Exception اتفاق افتد باز هم بلوک finally را اجرا خواهد کرد. در این صورت ابتدا Exception و سپس finally اجرا می‌شود.
- بلوک finally همیشه اجرا می‌شود.
- مثال ذکر شده همراه با finally در ادامه آمده است.

2014 *M. Damrudi*

مثال

```
TextReader reader = null;
try
        {
          TextReader reader = src.OpenText();
          string line;
          while ((line = reader.ReadLine()) != null)
          {
            source.Text += line + "\n";
          }
         }
finally
{
        if (reader !=null)
           {
             reader.Close();
           }
}
```

2014 *M. Damrudi*

کلاس در C#

- کلاس مکانیزم راحتی برای مدل کردن واحدهای به کار رفته به وسیله برنامه را فراهم می‌کند.
- کلاس کلمه ریشه‌ای طبقه‌بندی است.
- کپسوله‌سازی اصل مهمی در هنگام تعریف کلاس است.
- برنامه‌ای که از یک کلاس استفاده می‌کند نباید نگران اتفاقات درون کلاس باشد. به عنوان مثال زمانی‌که از Console.WriteLine استفاده می‌کنید نیازی نیست جزئیات کلاس Console را بدانید.
- هدف کپسوله‌سازی پنهان کردن اطلاعات است.

2014 *M. Damrudi*

کلاس در C#

- کپسوله‌سازی دو هدف مهم دارد:

۱- ترکیب متد و داده درون یک کلاس: پشتیبانی از طبقه بندی

۲- کنترل دستیابی به متدها و داده‌ها: کنترل استفاده از کلاس

تعریف کلاس به صورت زیر است:

```
class Circle
{
  public double Area()
  {
     return 3.14*radius*radius;
   }
 private double radius:
}
```

2014 *M. Damrudi*

کلاس در C#

- کلاس فوق شامل متد Area و فیلد radius است. به متغیر در یک کلاس فیلد گفته می‌شود.

```
Circle c;
```

متغیری از نوع Circle تعریف می‌کند.

```
c= new Circle;
```

مقداردهی متغیر c.

- new یک نمونه از کلاس را ایجاد می‌کند.
- یک متد یا فیلد را خصوصی (private) گویند اگر و تنها اگر از درون کلاس قابل دستیابی باشد.
- یک متد یا فیلد را عمومی(public) گویند اگر هم از داخل و هم خارج کلاس قابل دستیابی باشد.

2014 *M. Damrudi*

سازنده در C#

- هنگامی که از new برای ایجاد object استفاده می شود CLR آن Object را با تعریف کلاس بسازد. قسمتی از حافظه سیستم‌عامل را گرفته و با فیلدهایی که توسط کلاس تعریف شده‌اند، پرمی‌کند.
- سازنده یک method خاص است و همان نام کلاس را دارد. می‌تواند پارامتر بگیرد اما نمی‌تواند مقداری را بازگرداند(حتی void).
- هر کلاس یک سازنده دارد و اگر آن را ننویسید کامپایلر به صورت اتوماتیک یک سازنده پیش‌فرض تولید می‌کند. سازنده پیش فرض بدون پارامتر است.
- مثال بعد سازنده‌ای ایجاد کرده که فیلد radius را با ۰ مقداردهی کرده است.

2014 *M. Damrudi*

مثال

```
class Circle
{
  public Circle()
  {
    radius=0.0;
  }
  public double Area()
  {
     return 3.14*radius*radius;
   }
 private double radius:
}
```

2014 *M. Damrudi*

Overloadکردن سازنده در#C

- سازنده از نوع public است، اگر نوع ذکر نشود private است.
- برای فراخوانی متد یک از object از . استفاده می‌شود.

```
Circle c;
c=new Circle();
double a=c.Area();
```

2014 M. Damrudi

Overloadکردن سازنده در#C

- در مثال قبل همواره مساحت صفر است. می‌توان سازنده را Overload کرد.
- در Overload کردن متد باید پارامترهای متفاوتی قرار داد.
- ترتیب سازنده‌ها مهم نیست.

2014 M. Damrudi

Overloadکردن سازنده در#C

```
class Circle
{
  public Circle()
  {
   radius=0.0;
  }
  public Circle(double R)
  {
   radius=R;
  }
  public double Area()
  {
     return 3.14*radius*radius;
   }
 private double radius:
}
```


تمرین

❁کلاسی به نام person بنویسید:

❁برنامه استفاده کننده از این کلاس، اطلاعات نام، نام‌خانوادگی و سن ۳ نفر را دریافت کند.

❁اطلاعات را توسط متدی نمایش دهد.

تمرین

```
namespace Peoples
{
   class Program
      {
      static void Main(string[] args)
         {
          string n, f; int a;
          for (int i = 1; i <= 3; i++)
          {
             n = Console.ReadLine();
             f = Console.ReadLine();
             a = Int32.Parse(Console.ReadLine());
             Person p = new Person(n, f, a);
             Console.WriteLine(p.prints());
           } } }
```

2014 *M. Damrudi*

تمرین

```
class Person
{
       public Person(string x, string y, int z)
        {
          name = x;
          family = y;
          age = z;
        }
       public string  prints()
         {
          return name + "\n" + family + "\n" + age;
         }
       private string name, family;
       private int age;
}}
```

2014 *M. Damrudi*

تخریب کننده در#C

- تخریب کننده‌ها مشابه سازنده‌ها هستند با این تفاوت که در ابتدای نام آنها علامت ~ قرار می‌گیرد.
- هیچ پارامتری دریافت نکرده و هیچ مقداری برنمی‌گردانند.
- در هرنقطه از برنامه که نیاز به آزاد کردن منابع سیستم است می‌توان از تخریب کننده‌ها استفاده نمود.
- تخریب کننده معمولا زمانی فراخوانی می‌شود که Garbage Collector تصمیم به حذف شیء مورد استفاده برنامه و آزادسازی منابع سیستم گرفته باشد.
- به طور پیش‌فرض برنامه‌نویس نمی‌تواند خودش اشیاء را تخریب کند و این کار توسط GC انجام می‌گیرد. در صورت نیاز باید از کلاس IDisposable استفاده نمود.

2014 *M. Damrudi*

تخریب کننده در#C

```
class Person
{
      public Person(string x, string y, int z)
       { name = x;
         family = y;
         age = z;}
      public string  prints()
        {return name + "\n" + family + "\n" + age;}
      ~Person()
        {Console.WriteLine(“Destructor”);}
      private string name, family;
      private int age;
}
}
```

2014 *M. Damrudi*

static در C#

- اگر فیلدی static تعریف شود بین همه objectهای یک کلاس مشترک است. فیلدهای غیراستاتیک برای هر نمونه از یک object محلی هستند.
- فیلدی که const تعریف شود استاتیک است هرچند که از static در تعریف خود استفاده نمی‌کند. مقدار فیلد const هرگز تغییر نمی‌کند. فیلد در صورتی می‌تواند const باشد که مقدار عددی یا یک رشته باشد.

```
class Math
{
…
public const  double PI=3.141592;
}
```

2014 *M. Damrudi*

enum در C#

- enum مقادیر را به یک سری نام‌های سمبولیک محدود می‌کند.
- enum نوع شمارشی یک مقدار عددی را به یک عنصر مربوط می‌کند. شماره‌گذاری به صورت پیش‌فرض از صفر برای عنصر اول شروع می‌شود.

```
enum Season{ Spring, Summer, Fall, Winter}
```

- با تعریف enum می‌توان آن را مانند هر نوع داده دیگری استفاده کرد.
- پیش از استفاده از یک متغیر enum باید آن را با مقدار معتبری مقداردهی کنید.

2014 *M. Damrudi*

enum در C#

- می‌توانید به عناصر یک enum مقدار دهید.

```
enum Month{Jan=10, Feb=20, Mar=30}
```

- نحوه استفاده:

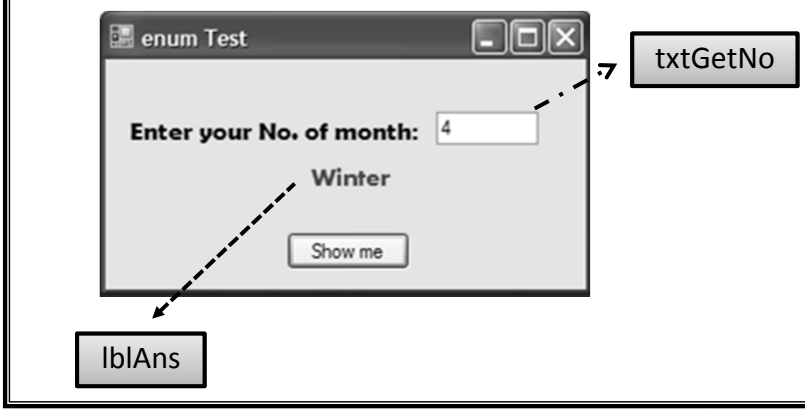
```
int x=Month.Jan
Month x=Month.Jan
int x=(int) Month.Jan          Casting
```

- اگر به عنصری مقدار ندهید، کامپایلر به آن مقداری می‌دهد که مقدار آن یک واحد از قبلی بیشتر باشد.

```
enum Season{Spring=1, Summer, Fall, Winter}
```

2014 *M. Damrudi*

مثال



2014 *M. Damrudi*

مثال

```
enum Season { Spring=1, Summer, Fall, Winter }
    private void btnShow_Click(object sender, EventArgs e)
{
    int x = Int32.Parse(txtGetNo.Text);
    switch(x){
      case (int)Season.Spring:
        lblAns.Text = "Spring";
        break;
      case (int)Season.Summer:
        lblAns.Text = "Summer";
        break;
      case (int)Season.Fall:
        lblAns.Text = "Fall";
        break;
      case (int)Season.Winter:
        lblAns.Text = "Winter";
        break;
    } }
```

2014 *M. Damrudi*

ساختار در C#

- با استفاده از ساختار می‌توان انواع داده جدید ایجاد کرد. ساختارها *مشابه و فقط مشابه* کلاس‌ها می‌باشند.
- برای یک structure نمی‌توان سازنده پیش‌فرض تعریف کرد. کامپایلر همیشه یک سازنده پیش‌فرض تولید می‌کند.
- پس از تعریف ساختار می‌توان از آن همانند انواع داده دیگر استفاده کرد.
- اگر یک سازنده در ساختار تعریف می‌کنید، مطمئن باشید که همه فیلدها مقداردهی می‌شوند.

2014 *M. Damrudi*

مثال

```
namespace ConsoleApplication1
{
   struct Time
  {
    public Time(int h, int m, int s)
     {
    hours=h%24;
    minutes=m%60;
    seconds=s%60;
     }
    public int Hour()
    {
      return hours;
    }
```


مثال

```
public int Minute()
{
      return minutes;
}
   public int Second()
{
      return seconds;
}
   private int hours;
   private int minutes;
   private int seconds;
}
```

```
class Program
 {
      static void Main(string[] args)
       {
         Time t1=new Time(34,435,43);
         Console.WriteLine(t1.Hour());
         Console.WriteLine(t1.Minute());
         Console.WriteLine(t1.Second());
       }
   }
}
```

تفاوت ساختار و کلاس در C#

- برای یک structure نمی‌توان سازنده پیش‌فرض تعریف کرد زیرا کامپایلر همیشه یک سازنده پیش‌فرض تولید می‌کند. اما در کلاس، کامپایلر در صورتی سازنده پیش‌فرض تولید می‌کند که شما آن را ننوشته باشید.
- یک کلاس می‌تواند از یک کلاس پایه ارث ببرد اما یک structure نمی‌تواند.
- کلاس نمونه‌ها را در heap قرار می‌دهد اما ساختار در پشته قرار می‌دهد.
- نوع در کلاس ارجاعی است اما در ساختار مقداری است.

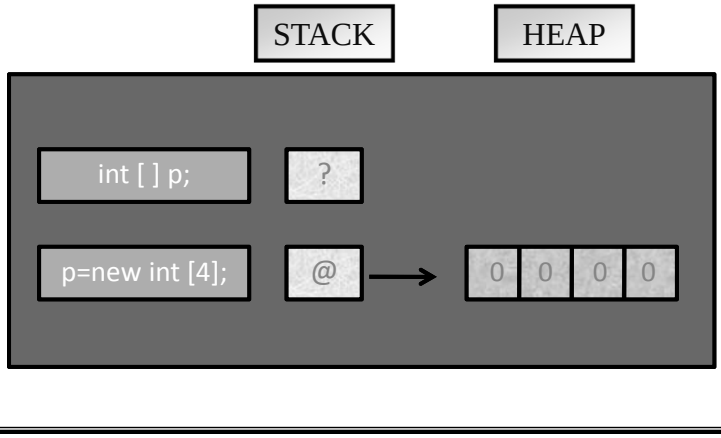
2014 *M. Damrudi*

آرایه در C#

- حالت کلی آرایه در C# به صورت زیر است:
- نوع آرایه [ ]نام آرایه;
- آرایه‌ها بدون توجه به نوع عناصرشان از انواع ارجاعی هستند. در واقع یک متغیر آرایه‌ای به یک نمونه آریه‌ای در heap اشاره می‌کند و عناصر خود را مستقیما روی پشته نگه‌داری نمی‌کند.
- هنگامی که یک متغیر آرایه‌ای تعریف می‌کنید، اندازه آن را مشخص نمی‌کنید.
- هنگامی اندازه آرایه را مشخص می‌کنید که می‌خواهید یک نمونه از آن ایجاد کنید.

2014 *M. Damrudi*

آرایه در C#



2014 M. Damrudi

آرایه در C#

- حالت کلی آرایه دوبعدی در C# به صورت زیر است:
- نوع آرایه [,] نام آرایه;
- مثال:
- int [,] table = new int [4,6];
- می‌توان مقادیر آرایه را در زمان اجرا و به صورت زیر انجام داد:
- int [ ] p= new int[4]={9,6,3,0};
- یا
- int [ ] p={9,6,3,0};

2014 M. Damrudi

آرایه در C#

- اولین اندیس آرایه صفر است.
- اگر اندیس آرایه کمتر از صفر یا بزرگتر یا مساوی طول آرایه باشد، کامپایلر IndexOutOfRangeException ایجاد می‌کند.

```
static void Main(string[] args)
{
        try
        {
           int[] a = { 2, 4, 6, 8 };
           Console.WriteLine(a[4]);
        }
        catch(IndexOutOfRangeException ex)
        {
           Console.WriteLine(ex.Message);
        } }
```

2014 *M. Damrudi*

آرایه در C#

- Length خاصیتی است که می‌توان از آن برای مشخص کردن تعداد عناصر آرایه استفاده کرد.

```
static void Main(string[] args)
      {
            int[] age = { 12, 14, 16, 18 , 20};
            for (int i = 0; i != age.Length; i++)
               Console.WriteLine(age[i]);
      }
```

2014 *M. Damrudi*

آرایه در C#

- از دستور foreach می‌توان برای دسترسی به عناصر آرایه استفاده کرد.

```
static void Main(string[] args)
    {
        string [ ] names = { "Sara" , "Tara", "Dina"};
        foreach (string  name in names )
            Console.WriteLine(name);
    }
```

2014 M. Damrudi

آرایه در C#

- دستور foreach:
  - ☆ در تمام آرایه حرکت می‌کند،
  - ☆ از صفر تا Length-1 حرکت می‌کند،
  - ☆ از اندیس عناصر آرایه اطلاعاتی وجود ندارد،
  - ☆ نمی‌توان عناصر را تغییر داد.
- در موارد فوق از for استفاده کنید.

2014 M. Damrudi

آرایه در C#

- آرایه‌ها از نوع ارجاعی هستند. یک متغیر آرایه یک ارجاع به یک نمونه از آرایه است. هنگامی‌که یک متغیر از نوع آرایه کپی می‌شود، دو ارجاع به یک نمونه از آرایه خواهید داشت.

```
int [ ] p={9,7,5,3};
int [ ] a=p;
```

- برای آنکه از آرایه کپی تهیه کنید به گونه‌ای که یک متغیر آرایه به آن ارجاع کند.

```
int [ ] p={9,7,5,3};
int [ ] c= new int[4]; یا int [ ] c= new int[p.Length];
  for (int i=0;i != c.Length; i++)
       c[i]=p[i];
```

2014 *M. Damrudi*

آرایه در C#

- استفاده از متد CopyTo برای کپی آرایه:

```
int [ ] p={ 4,3,2,1};
int [ ] c= new int[ p.Length];
p.CopyTo(c,0);
```

- استفاده از متد Copy برای کپی آرایه:

```
int [ ] p={ 4,3,2,1};
int [ ] c= new int[ p.Length];
Array.Copy(p,c,c.Length);
```

- استفاده از متد Clone برای کپی آرایه:

```
int [ ] p={ 4,3,2,1};
int [ ] c= (int[ ]) p.Clone();
```

2014 *M. Damrudi*