OPTİMİZASYON TEKNİKLERİ-1. Hafta

GİRİŞ

Mühendisliğin temel mantığı, günlük yaşamdaki problemlere, temel fen bilimlerini (fizik, kimya, matematik) kullanarak en optimum çözüm üretmesidir. Bu açıdan bakıldığın tüm mühendislerin bilmesi gereken bir konudur. Özelde daha çok endüstri mühendislerinin ilgilendiği bir alandır.

Burada amaç, belli bir hedef fonksiyonu, eldeki değişkenleri ayarlayarak minimize yada maksimize etmektir. Bunları gerçekleştirirken kısıtlamalar söz konusu olabilir.

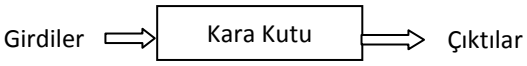
Optimum çözümü ararken bir takım algoritmalar kullanılabilir. Değişkenler zamana bağlı olarak değişiyorsa dinamik optimizasyon söz konusu olur. Tam olarak optimum çözümü bulmak bazen geliştirilen algoritmalarla yada bilgisayar hafıza yetersizlikleriyle çözmek mümkün olmayabilir. Bunun yerine optimal çözüme yakın sonuçlar üreten sezgisel yöntemler kullanılır.

Bu dersimizde tam çözüm bulan yada yaklaşık çözüm bulan sezgisel algoritmalardan örnekler göreceğiz ve bunlarla ilgili programlar yazarak gerçek hayattaki problemleri çözeceğiz. Bu tekniklerin kullanılabilmesi için iyi bir programlama alt yapısının olması gerekir. Bu amaçla bu derste kullanacağımız C# programının temel komutlarına ve fonksiyonlarına bakacağız.

C# PROGRAMININ TEMELLERİ

Problem Çözme Metodu

Programlamayı düşünmeden önce problemi çok iyi tanımak gerekir. Bu aşamada bizden ne istendiği (Çıktılar) ve bunun için giriş verileri ne olacak bunları iyi bilmemiz gerekir. Daha sonrasında giriş verilerinden çıkış verilerine ulaştıracak karakutunun içerisini program teknikleriyle çözmeliyiz. Program tekniklerinin gelişmesi için kullandığımız programlama dilinin alt yapısına yeter düzeyde hakim olmamız ve konumuzla alakalı algoritma yapılarına bilmemiz gerekir.

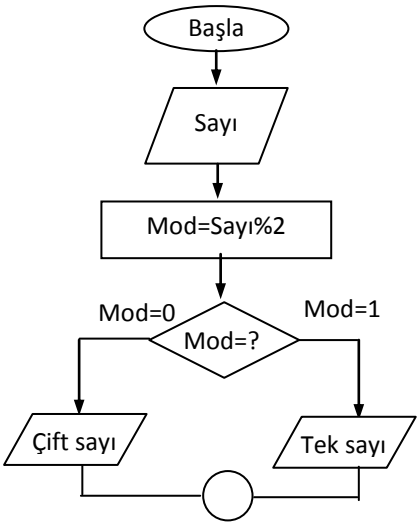
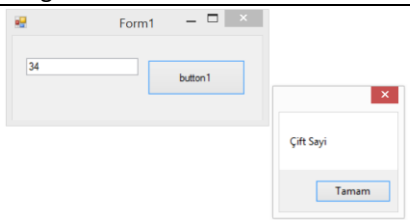


Problemin Çözümü (Kara kutu)
1) Programlama dili bilinmeli
2) Algoritma teknikleri bilinmeli

Çözüm metodunu tasalarken şu üç aşama üzerinde tasarım yapmak faydalı olacaktır. Günlük kullanılan dil ile adımların ifade edilmesi (algoritma), Bu ifadelerin şekillerle daha anlaşılır hale getirmek için görsel bir şekil oluşturma (akış diyagramları) ve ardından program kodlarını yazma. Çoğunlukla programcılar birinci ve ikinci aşamaları direk kafalarında oluşturup programlamaya geçmektedirler. Bu durum baştan bazı adımların gözden kaçmasına neden olmakta ve program ilerledikten sonra geriye dönük köklü değişiklikleri yapmayı gerektirmektedir. Bu nedenle baştan planlama yapmak önemlidir.

Örnek: Bir sayının tek mi çift mi olduğunu hesaplayan bir programı yazmak için bu üç aşamayı oluşturalım.

Yapay Kod (Algoritma)	Akış Diyagramı	Program Kodu

1. Başla
2. Dışarıdan sayıyı al
3. Sayının 2 ye göre Modunu hesapla
4. Mod değeri 0'a eşitse çift sayıdır, 1'e eşitse tek sayıdır
5. Sonucu ekrana yazdır.
6. Bitir
		
		`private void button1_Click(object sender, EventArgs e)`
		`{`
		`int Sayi = Convert.ToInt16(textBox1.Text);`
		`if (Sayi % 2 == 0)`
		`MessageBox.Show("Çift Sayi");`
		`else MessageBox.Show("Tek Sayi");`

	Bitir	}

C# ve Nesne Tabanlı Programlama

C# eski C dilinin daha gelişmiş hali olan herşeyin Nesne tabanlı olarak hazırlandığı bir programdır. Nesne tabanlı ne demek? şu şekilde açıklayabiliriz.

Örneğin toplama işlemi yapacak bir programa ihtiyacımız olsun. Bu program dışarıdan iki değeri alır, bunları toplar ve sonucu bize geri verir. Biz her seferinde toplama işlemi yaptıracağımız zaman bu kodları tekrar tekrar yazmak zorundayız. Oysa bu kodları alsak, bunlara bir isim versek ve bunları bir paket haline getirsek kenara bir yere saklasak. Bizim ne zaman bir toplama işlemine ihtiyacımız olursa, bu kodları yeniden yazmak yerine bu kenardaki paketin kopyasını alıp onu kullansak böylece içindeki kodları yazmakla tekrar uğraşmamış oluruz ve direk kullanırız. Bu paketi kullanırken girişte neleri vereceğiz çıkışta ne alacağız sadece bunları bilsek yeterlidir.

Yapılacak her iş bu şekilde paketler halinde (nesneler=sınıflar) hazırlanmış olsun. Binlerce paketin bir arada tutulması olayı karmaşık hale getirir (bir depoya atılmış binlerce ürünün eşyanın bir arada olması gibi). Bunun yerine daha düzenli hale getirmek için belli paketler belli odalara istiflenirse, böylece aradığımız paketi bulmamız daha kolaylaşacaktır.

İşte bu şekilde, bir iş yapan program parçacıklarına (paketlere) nesne diyeceğiz. Bu nesnelerin bir araya getirilip gruplandırıldığı odalara ise isim uzayı (namespace) diyeceğiz. Her nesnenin kendisine has özellikleri (properties) ve yaptığı bazı işler (metod=fonksiyon) vardır.

Program yazarken C# kendisinin daha önceden hazırladığı tüm komutlar birer nesne şeklinde (hazır alınıp kullanılacak paket) olduğu gibi kendimizde bir işi yapan nesneler yazabiliriz. Böylece aynı komutları tekrar tekrar yazmak derdinden kurtuluruz.

Yeni bir program yazmaya başladığımızda C# tüm komutları (nesneleri) bizim kullanımıza açmıyor. Öncelikle hangi komut gruplarını kullanacağımızı bizden istiyor ve onların bulunduğu odaların isimlerini bizden istiyor. Bu nedenle kullacağımız komutların hangi odalarda bulunduğunu bilmemizde önemlidir.

```
Form1.cs*   Form1.cs [Design]*
WindowsFormsApplication70.Form1        button1_Click(object ser
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;          Kütüphaneler
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication70      NameSpace (İsim Uzayı)
{
    public partial class Form1 : Form    Class (Sınıf)
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
                                         Method (Fonksiyon)
        }
    }
}
```

Bir nesnenin methodu çağrılırken şu sıra takip edilir. (NameSpace İsmi).(Class İsmi).(Method İsmi) yazılır. Eğer kütüphaneye NameSpace ismi eklenmiş ise Bu kısım yazılmaz direk Class isminden başlanabilir. Class dan yeni nesnemiz oluşturulduktan sonra Bu nesnemizin yanına yazacağımız nokta işaretinin devamında o nesneye ait (özellikler ve metodlar) ortaya çıkar.

Örnek:

1) Kütüphaneye komutun bulunduğu isim uzayı eklenirse.

```
using System.Data.OleDb;                          //kütüphaneye nesnenin bulunduğu namespace eklendi.
    .....
OleDbConnection baglanti = new OleDbConnection("BaglantiYolu");   // bağlantı nesnesi oluşturuldu.
Baglanti.Open();                                                  //Bağlantı nesnesinin metodu oldu.
```

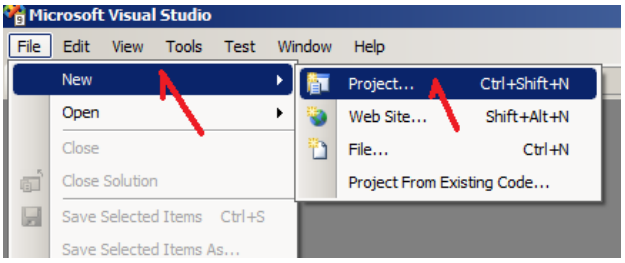
2) Kütüphaneye komutun bulunduğu isim uzayı eklenmez ise

```
System.Data.OleDb.OleDbConnection Baglanti = new System.Data.OleDb.OleDbConnection("BaglantiYolu");
```

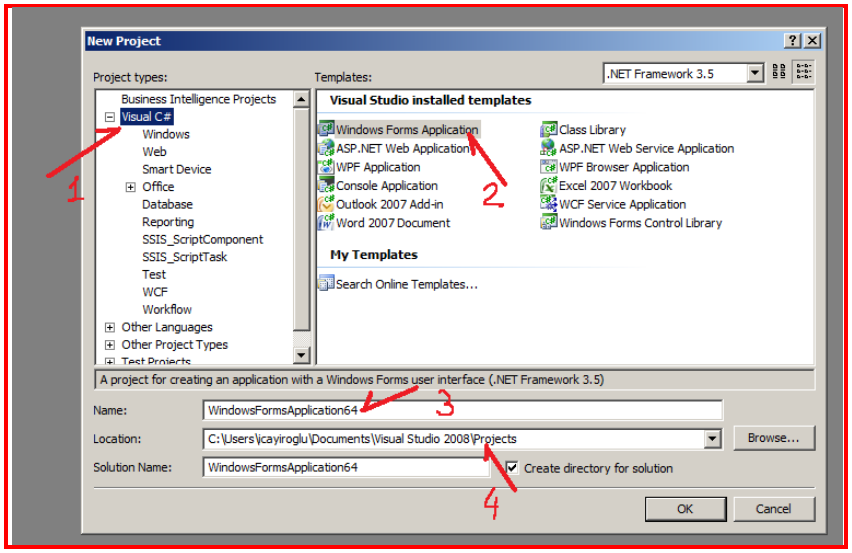
Baglanti.Open(); //Bağlantı nesnesinin metodu oldu.

Visual Studio'da Yeni Bir Proje Oluşturma

Visual Studio (VS) programını çalıştırdığımızda karşımıza boş bir ekran gelir. Yeni bir proje oluştururken File>New>Project yolu kullanılarak yeni bir proje başlatırız.

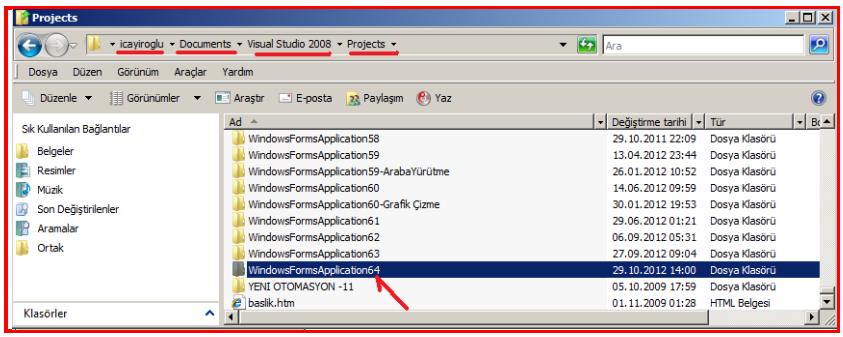


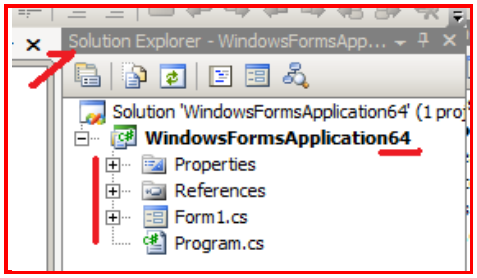
Karşımıza çıkan ekrandan C# dili seçili olmalı (1 nolu). Programımız masaüstü bir program olacağından ve Windows ortamında çalışan bir program olacağından "Window Forms Application" seçili olmalı. Projenin adı 3 nolu yerde gösterilen addır. Bu projenin bilgisayarımızda nerede kayıtlı olacağını gösteren yer ise 4 nolu yerdir. VS yi kapattıktan sonra hazırladığımız programı başka bir yere taşımak istiyorsak 4 nolu yere gidip orada 3 numara ile gösterilen ismin bulunduğu klasörü alıp kopyalayabiliriz.



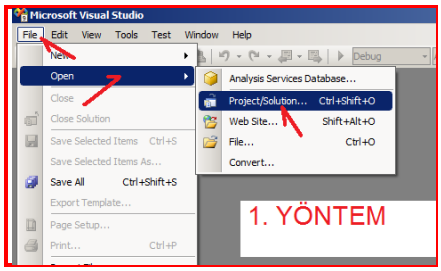
Ok dügmesine bastıktan sonra VS bize projemizin temelini oluşturan ilk yapıyı hazır olarak verecektir. Bunlarla ilgili ilk kodları projemizin içerisine atar ve bilgisayarımızdaki 4 numara ile gösterilen adresteki yere dosyaları kopyalar.

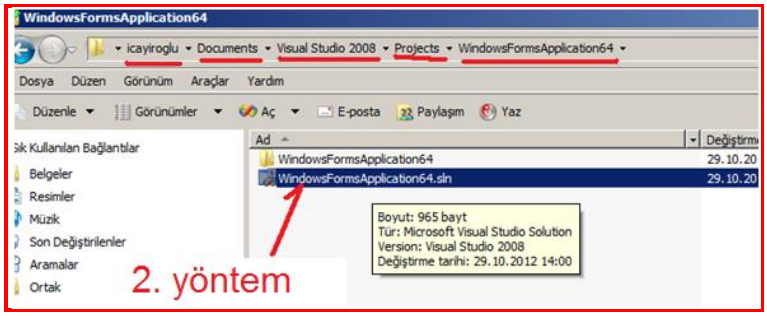
Şimdi bilgisayarımızdaki bu dosyaları görelim. Göreceğimiz gibi 64 numaralı proje oluşturulmuş durumda. Aynı proje VS içerisinde de şuan açık durumdadır. Biz VS içerisinde projenin dosyaları arasında gezerken VS nin kendi Gezgin (explorer) penceresini kullanırız. Buna "Solution Explorer" penceresi diyoruz.





Projemizde epey bir çalışma yaptığımızı düşünelim. Ertesi günü tekrar projemizi açmak istersek iki yolu kullanabiliriz. Ya VS nin içerisinden File>Open>Project yolunu kullanıp buradaki 64 numaralı projeyi açarız. Yada Windows'un kendi gezgin penceresinden gidip 64 nolu projenin ana dosyasına çift tıklayıp VS ile birlikte projenin açılmasını sağlayabiliriz.

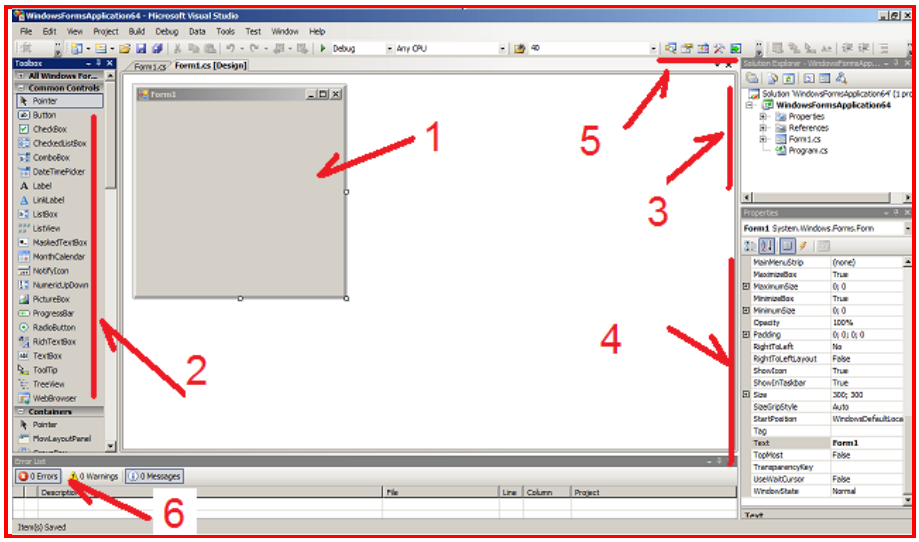




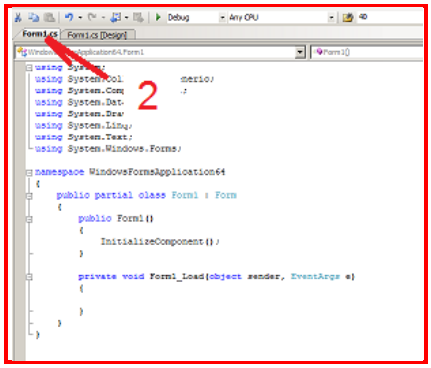
İlk projemizi New deyip oluşturduktan sonra karşımıza aşağıdaki gibi bir ekran gelecektir. Bu ekranda bize ilk olarak hazır bir nesne diyebileceğimiz Form nesnesi verilmiştir. Bu nesne programımızın zeminin oluşturan düz bir çerçevedir (1 nolu). Her nesnenin Özellikleri (properties) ve üzerinde gerçekleştirilebilecek olayları (events) vardır demiştik. Örneğin Form nesnesinin zemin rengini farklı bir renkte yapmak istiyoruz. Bunu nereden ayarlayabiliriz. Sağ taraftaki Properties penceresinden bu Bunu BackColor (arka renk) kısmında değiştirebiliriz. İşte bunun gibi projemize eklediğimiz hazır nesnelerin birçok özelliğini sağ taraftaki Preoperties penceresinde (4 nolu) ayarlayabiliriz.

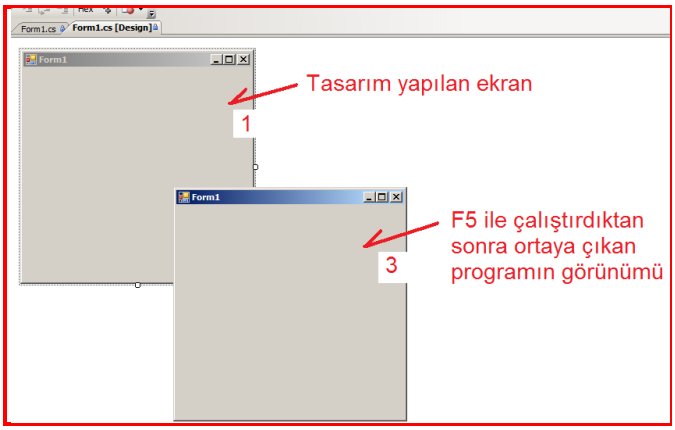
Dikkat edersek buradaki projemizde sadece bir tane hazır nesne (Form nesnesi) bulunmaktadır. Eğer istersek bu formun üzerine daha birçok hazır nesneleri ekleyebiliriz. Bu iş için sol taraftaki Toolbox (araçkutusu) (2 nolu) penceresini kullanabiliriz. Örneğin formun üzerine Textbox (yazı yazma kutucukları) button (buton) label (etiket) gibi daha bir çok nesneyi sol taraftan sürükleyip formun üzerine getirebiliriz.

Dikkat edersek ekranımızda birçok panel denilen pencereler bulunmaktadır. Bunları tekrar açıklarsak Toolbox hazır nesnelerin bulunduğu panel, Solution Explorer projemizin içinde bulunan dosyaları, Properties ise seçili olan nesnenin özelliklerini değiştirmeyi sağlayan kısımdır. Kodlarda bir hata var ise bu hatanın nerede olduğunu gösteren 6 nolu kısımda Errors panelimiz bulunmaktadır. İşte bütün bu paneller eğer ekranımızda gözükmeyip kaybolmuş ise 5 numara ile gösterilen yerdeki düğmeler tıklarsak ortaya çıkacaktır.



Projemizi hazırlarken üç tane ekran bizim için önemlidir. Bunlardan birincisi Design penceresi. Bu pencere projemizin nasıl görüneceğini bize gösterir (1 nolu) . Diğeri içeresine kod yazdığımız ekrandır. Bu ekranda C# kodlarını yazacağız (2 nolu). Birde Programı F5 ile çalıştırdıktan sonra Design penceresini çalışır halde gördüğümüz ekrandır (3 nolu).





Değişkenler

Verilerin tutulacağı değişkenlerin tanımlanması zorunludur. Değişken tanımlanırken hangi tip veri türü tutulacağı ve hangi aralıkta çalışacağına dikkat edilmelidir. Çalışacağı aralık içerisinde mümkün olduğunca en düşük hafıza tutan veri türünü tercih etmek gerekir. Değişkenlerin tanımlanması karmaşık program yapılarında bilgilerin karışmalarını engellemesi açısından ve en az ram kaynaklarını kullanmaya neden olduğu için kullanımı önemlidir. Değişken türleri aşağıdaki tabloda verilmiştir. Bir değişken hafızada tutmuş olduğu byte sayısı kadar bilgiyi tutabilir. Örneğin 2^8 = 1 byte = 256 kadar olan sayıları tutabilir. Yani 0 ile 256 arası sayıları tutabilir. Bu şekilde tanımlama işaretsiz (unsigned) tanımlama olur Eğer negatif bölgeye de geçiş yapılarsa bu sayı ikiye bölünür. -128 ile +128 arasında bilgiler tutulmuş olur. Bu tanımlama işaretli tanımlama olur.

Adı		Hafıza (byte)	Sınır Değerleri	İşaretsiz (unsigned) Değerleri		
Tamsayı	sbyte	1	-128 : + 127	byte	1	0 : + 255
	short	2	-32 768 : +32 767	ushort	2	0 : + 65 535
	int	4	-2 147 483 648 : + 2 147 483 648	uint	4	0 : +4 294 967 295
	long	8	-9 223 372 036 854 775 808 : +9 223 372 036 854 775 808	ulong	8	0 : +18 446 744 073 709 551 616
Ondalık	float	4	$\pm 3.6x10^{-38}$: $\pm 3.6x10^{+38}$ (tek duyarlık)	Yoktur		
	double	8	$\pm 1.8x10^{-308}$: $\pm 1.8x10^{+308}$ (çift duyarlık)	Yoktur		
	decimal	16	28 digit ondalık sayı tutar.	Yoktur		
metin	char	2	Unicode tek karakteri tutmak içindir.			
	string	2x	Birden fazla karakteri tutmak içindir.			
	bool	1 (bit)	0 : 1 (false – true)			

Değişkenlerin Yaşam Süreleri (Geçerli oldukları aralıklar)

Degişkenlerin tanım aralıkları ve hafızada tuttukları yerin yanında yaşam süreleri yada geçerli oldukları bölge hakkında da bilgi sahibi olmamız gerekir. Buna göre tanımlanan değişkenler dört farklı şekilde açıklanabilir.

a) Local (yerel) değişkenler: Bu değişkenler sadece tanımlandıkları fonksiyon içinde geçerlidirler. Tanımlandıkları fonksiyon dışından ulaşılmak mümkün değildir. Fonksiyon çağrıldığında hafızada oluşturulurlar, fonksiyondan çıkıldığında ise tekrar hafızadan silinirler.

Örnek: Uygun bir örnek geliştiriniz.

b) Global (genel) degişkenler: Tüm fonksiyonarın dışında tanımlanırlar. Dolayısı ile tüm fonksiyonlarda geçerli olurlar. Program çalışmaya başladığı anda hafızada yer alırlar ve program çalıştığı sürece hafızada kalırlar. Program sona erdiğinde hafızadan silinirler.

Örnek: Uygun bir örnek geliştiriniz.

c) Statik Degişkenler: Statik değişkenler fonksiyonların içinde tanımlanan değişkenlerdir. Local değişkenlere benzerler fakat bu değişkenler fonksiyondan çıkılsa bile içerisindeki değerleri korurlar ve fonksiyon tekrar çağrıldığında bir önceki çağrılmadaki son değerleri kullanırlar.

C# statik değişkenler class seviyesinde tanımlanırlar. Method (fonksiyon) içerisinde tanımlanamazlar.

Ödev: Uygun bir örnek geliştiriniz.

d) Register Degişkenleri: Program işletilirken, işlem sırası gelen değişkenlerin degerleri hafızadan CPU (merkezi işlemci) ya taşınırlar. İşlemden sonra içeriği tekrar hafızaya iade edilir. Eğer program içerisinde bir değişkene çok fazla ihtiyaç duyuluyorsa (çok fazla sayıda kullanılıyorsa) sürekli olarak bu işlemlerin yapılması zaman kaybettirici bir hal alır. Değişkenlerin bunun yerine Merkezi işlemciye ait Registerlarda tutulması değişkenlerin üzerindeki işlemleri hızlandırarak programın daha kısa sürede çalışmasını temin eder. Örneğin sürekli içeriği değişen döngü değişkenlerinin bu şekilde tanımlanması yararlı olacaktır.

Ödev: Uygun bir örnek geliştiriniz.

e) Pointer (işaretçi) kullanımı: İşaretçilerin diğer değişkenlerden tek farkı başka bir değişkenin bellek adresini içeriyor olmasıdır. İşaretçiler kitaplardaki indeksler olarak da düşünebiliriz. Bir kitabın sayfalarını bellekteki gözlerin adresleri ile ilişkilendirirsek, kitabın indeksindeki her bir başlık bir işaretçi olarak düşünülebilir. Bir işaretçiyi kullanmadan önce, bir bellek adresinin mutlaka işaretçiye atanması gerekir. Çünkü, işaretçi bellek adresleri ile işlem yapar. C dili işaretçilere bellek adresi atama işlemini, & işlemcisini kullanarak yapar. C'de işaretçilerle kullanılan iki farklı işlemci vardır. Bunlardan biri * , diğeri ise & işlemcisidir. Bu iki işlemcinin görevlerinin anlaşılması işaretçi kavramını anlamak açısından çok önemlidir. C# gösterici kullanımı pek anlamlı değildir. İşaretçilerin kullanımının zor olması ve kestirilmesi kolay olmayan hatalara yol açabilmesi muhtemeldir. Zaten C#'ta gösterici kullanacağımız zaman kodu **unsafe(güvensiz)** anahtar sözcüğü ile işaretlememiz gerekir. Aksi halde program derlenemeyecektir. Normal bir metot içinde gösterici kullanımı yasaklanmıştır. C#'ta pointer kullanabilmek için öncelikle Visual Studio'da unsafe kodlamaya izin vermemiz gerekmektedir. **Project->ProjeAdi.Properties** seçenegini açalim. oradan **Build** yaprağında **Use Unsafe** kısmını işaretleyelim.

* : Hangi işaretçi değişken adından önce kullanılırsa, o işaretçi değişkene adresi atanan değişkenin değerini verir.

& : Hangi değişken adından önce kullanılırsa, o değişkenin adresini verir.

```
private unsafe void button1_Click(object sender, EventArgs e)
{
    int sayi = 21;
    int *isrt = &sayi; //sayi degişkenin adresi *isrt1 işaretçisine aktarılmıştır.

    Hesapla(*isrt); //fonksiyona giderken *isrt1 gösterdiği adresteki değer götürülüyor.
    MesajGoster();

}

public void Hesapla(int deger)
{
    MessageBox.Show((deger).ToString());
}

private unsafe void MesajGoster()
{
    int sayi =21;
    int *isrt = &sayi; //n sayi değişkenin adresi *isrt işaretci değişkenine aktarılmıştır.
    MessageBox.Show((*isrt).ToString());
}
```

Otoperatörler

x=x+y	eşiti	x+=y
x=x-y	eşiti	x-=y
x=x*y	eşiti	x*=y
x=x+1	eşiti	x++
x=x-1	eşiti	x--
z=x%y	eşiti	(Modunu alır: x sayısının y ile bölümünde kalanı z ye atar.)
x=x%y	eşiti	x%=y

atama	=	a=b;	(b nin içindeki değer a nın içine atılır, sağ dan sola doğru)
eşit mi?	==	a==b;	(a içindeki değer b nin içindeki değere eşit mi?)
büyük mü?	>	a>b;	(a nın içindeki değer b nin içindeki değerden büyük mü?)
küçük mü?	<	a<b;	
büyük eşit mi?	>=	a>=b;	
küçük eşit mi?	<=	a<=b;	
Değil mi?	!=	a!=b;	(a nın değeri b nin değerine eşit degilse)
Ve	&&	a=3 && b=4;	(anın değeri 3 eşitse ve b nin değeri 4 eşitse)

Veya || a=3 || b=4; (a nın değeri 3 eşitse yada b nin değeri 4 eşitse)

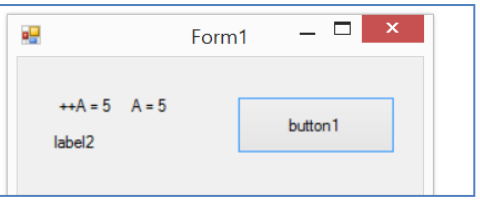
++a İlk olarak a nın değerini bir artır. Sonra a nın içindeki yeni değeri işlet. Aynısı (a--) içinde geçerlidir.

a++ İlk olarak a nın içindeki değeri işlet, sonra da a nın değerini artır. Aynısı (a--) içinde geçerlidir.

Örnek

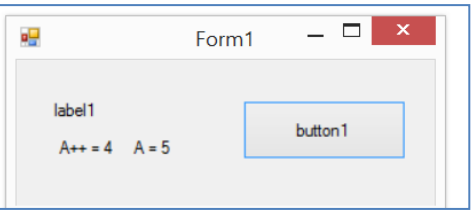
```
private void button1_Click(object sender, EventArgs e)
{
    int A = 4;

    label1.Text = " ++A = " + (++A).ToString() + "     A = " + (A).ToString();
    //label2.Text = " A++ = " + (A++).ToString() + "     A = " + (A).ToString();
}
```




```
private void button1_Click(object sender, EventArgs e)
{
    int A = 4;

    //label1.Text = " ++A = " + (++A).ToString() + "     A = " + (A).ToString();
    label2.Text = " A++ = " + (A++).ToString() + "     A = " + (A).ToString();
}
```



Karar Yapıları

If–Else yapısı: Belirlenen koşullar içinden hangi koşul gerçekleştiğinde onunla ilgili işlemleri yapmak için kullanılır. Koşullar için mantıksal operatörler kullanılır (eşit mi, büyük mü, degil mi vs).

```
if(şart)
{
   Komutlar;
}
else if (şart)
{
   Komutlar;
}
else if (şart)
{
   Komutlar;
}
….
else
{
   Komutlar;
}
```

Switch-Case Yapısı: Bu yapıda koşullar için eşit söz konusudur. Değişkenin aldığı değere göre ilgili durum çalıştırılır. Hiç bir durum gerçekleşmezse default bölümü çalıştırılır.

```
switch(değişken)
{
case kosul1:
      ...
   break;

case kosul2:
      ....
   break;
...

case kosuln:
      ...
   break;

default:
      .....
   break;
}
```

Döngüler

For Döngüsü: Belli sayıda döngü işlemleri gerektiğinde kullanılır.

```
for (ilk değer şartı ; son değer şartı ; artış)
{
....
}
```

Örnek: Uygun bir örnek geliştiriniz.

For-Each Döngüsü: Bir dizi içindeki tüm elemanların her biri için işlem yaparken kullanabiliriz. Burada dizinin kaç elemanı olduğunu bilmemize gerek yoktur.

```
foreach (degisken_tipi degisken in dizi)
{
...
}
```

Örnek: Uygun bir örnek geliştiriniz.

While Döngüsü: Şart döngüye girmeden kontrol edilir ve şart gerçekleştiği sürece döner.

```
while (şart)
{
}
```

Örnek: Uygun bir örnek geliştiriniz.

Do While Döngüsü: Bu döngü en az bir kez çalıştırılır, ondan sonra şartı kontrol eder. Şart gerçekleştiği sürece de dönmeye devam eder.

```
do
{
    Komutlar;
}while (şart)
```

Örnek: Uygun bir örnek geliştiriniz.

Matematik (Math) Kütüphanesi Fonksiyonları

Komutların çalışması için kütüphaneyi sayfanın en üstüne eklemeye gerek yoktur. Normal şartlarda kütüphanesi eklenmiş olarak gelir ve komutlar direk çalışır. Bazı önemli komutlar aşağıda verilmiştir.

Math.PI; // pi sayısını verir

Math.Sin(b); // b sayısının sin değerini alır

Math.Pow(b,c); // b^c demektir

Math.Sqrt(b); // Karekök değerini alır daha fazla kök için $a^{(2/3)}$, Math.Pow(a,(2/3))

Math.Ceiling(b); // Ondalık sayıyı üste yuvarlar, b=10.3 , 11 çıkar

Math.Floor(b); // Ondalık sayıyı aşağıya yuvarlar, b=10.3 , 10 çıkar

Math.Round(b); // En yakın tamsayıya yuvarlar, b=10.3 , 10 çıkar, b=10.7 den 11 olur. b=10.49864 sayısı , Dikkat b=10.5 sayısını 10 yuvarlar.

Math.Min(b,c); //iki sayıdan en küçük olanı verir. b ve c sayısından en küçük sayıyı verir. b=3 , c=4 ise sonuç 3 çıkar

Math.Max(b,c); //iki sayıdan en büyük olanını döndürür.

Math.Abs(b); // sayının mutlak değerini alır, yani tüm sayılar pozitif çıkar.

Math.Log10(b); // b sayısının 10 tabana göre logaritmasını alır. b=100 ise sonuç 2 çıkar b=102 => 2 çıkar.

Math.Log(b); // b sayısının In'ini almaktadır. e tabanına göre logaritmasını alır.

Math.Log(b,c); //c tabanında b sayısının logaritmasını alır. Örneğin b=8 ve c=2 ise sonuç 3 tür.

Math.Round(b,c); // b sayısının virgülden sonra C haneye kadar yuvarlatır. b=10.234567 TL sayısını şu şekilde yapmalıyız. b=Math.Round(b,2); şeklinde yazılmalıdır.

String Kütüphanesi Fonksiyonları

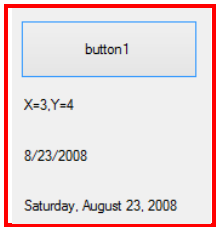
Metinler üzerinde işlem yaparken string fonksiyonları kullanırız. Kullanacağımız bazı önemli fonksiyonlar aşağıdadır.

String.Format() – String ifadeleri biçimlendirmek için kullanılır. Örneğin tarih formatı vermek için yada stringin sağına soluna boşluklar eklemek yada iki string ifadeyi bir araya istenilen şekilde getirmek için bu fonksiyonu kullanabiliriz.

Örnek:

```
int x = 3, y = 4;
label1.Text = String.Format("X={0},Y={1}", x, y);

DateTime date = new DateTime(2008, 8, 23);
label2.Text = String.Format("{0:d}", date);
label3.Text = String.Format("{0:D}", date);
```



Split() – String'i dizi halinde parçalamak

Bir string içerisindeki kelimeleri bazı karakterleri kullanarak ayırmak istiyor isek bu durumda **Split()** metodunu kullanmamız gerekir.**"2,3"** koordinat bilgisinden **2** ve **3** sayılarını elde etmek istiyor isek Split() metodu harika metoddur.

Aşağıdaki kod parçacığında x,y,z değerleri string ifadeden parçalanıp elde edilmektedir.

```
// Koordinat bilgisi string olarak tutuluyor.
string coord = "2,3,5";

// Bu komut sonrasında 3 string ifadeden oluşan bir dizi elde edilecektir.
// xyz[0] = "2"
// xyz[1] = "3"
// xyz[2] = "5"
string[] xyz = coord.Split(',');

int x = int.Parse(xyz[0]);
int y = int.Parse(xyz[1]);
int z = int.Parse(xyz[2]);

// Ekran Çıktısı : 2,3,5
Console.WriteLine("{0},{1},{2}", x, y, z);
```

SubString() – String içindeki alt stringleri elde etmek

Bir string ifadenin içinde,4.karakterden başlayıp 10 karakter elde etmek istiyor isek bu durumda kullanacağımız metod **SubString()** metodudur.

Aşağıdaki kod parçalarını inceleyelim;

```
string text = "Visual Studio 2005";
Console.WriteLine(text.Substring(7,4));     // Ekran Çıktısı : "Stud"
Console.WriteLine(text.Substring(7));       // Ekran Çıktısı : "Studio 2005"
```

Trim() – TrimEnd() – TrimStart() – Boşlukları kaldıran fonksiyonlar

Trim metodları, string içindeki boşluklarla bir derdiniz var ise çok işinize yarayacaktır.

Aşağıdaki kod örneğini ve açıklamaları inceleyelim;

```
string text = "    Visual Studio 2005    ";

/*
* Trim()       : Text'in başındaki ve sonundaki boşlukları kaldırır
* TrimEnd()    : Text'in sonundaki boşlukları kaldırır.
* TrimStart()  : Text'in başındaki boşlukları kaldırır.

/
Console.WriteLine("-{0}-", text.Trim());        // -Visual Studio 2005-
Console.WriteLine("-{0}-", text.TrimEnd());     // -    Visual Studio 2005-
Console.WriteLine("-{0}-", text.TrimStart());   // -Visual Studio 2005    -
```

Replace() – Yer değiştirme fonksiyonu

Bir string içindeki bir değeri başka bir değerle değiştirmek istiyor iseniz Replace() metodunu kullanmanız gerekmektedir.

Aşağıdaki kod parçacığını inceleyelim;

```
string text = "Visual Studio 2005";
string text2 = text.Replace("sual", "SORU");
Console.WriteLine(text2);   // "Ekran Çıktısı : ViSORU Studio 2005"
```

Diziler

Birden fazla değerleri tek bir değişken adı altında tanımlayarak bilgileri hafızada tutabilmek için kullanılır. İki farklı şekilde kullanılabilir.

Boyutlu Dizi tanımlama: Bu kullanımda dizinin boyutu verilmek zorundadır.

double [,] sayi =new double[100,10]; //Tanımlaması
sayi[i,j]; //Kullanımı

ArrayList Dizisi: Bu tanımlamada dizinin boyutu belirlenmez. Normalde dizilerde tüm elemanlar tanımlanan tipde olmak zorundadır. Fakat bu dizi tanımlamasında farklı tipleri (string, int vs) aynı dizi içerisinde tutmak mümkündür. Kütüphanesi;

using System.Collections;

Tanımlama aşağıdaki gibi yapılır.

ArrayList Dizi1 =new ArrayList();

Kullanımı ile ilgili bazı komutlar şunlardır.

Dizi1.Clear(); //dizi içerisindeki tüm elemanları siler.

Dizi1.Count; //dizinin eleman sayısını verir.

Dizi1.RemoveAt(3); //indis numarası 3 olan elemanı Diziden siler. Not; indis numaraları 0 dan başlar.

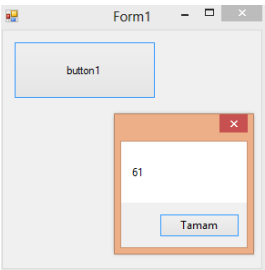
Dizi1.Add(TextBox2.Text); //Diziye eleman ekler

Dizi1[i].ToString() //Diziden okumayı sağlar

Komutlar çalışmazsa `using System.Collections;` Kütüphanesinin eklenmiş olmasına dikkat edin.

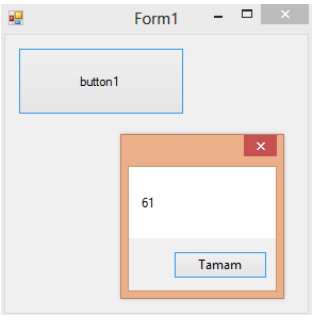
Örnek: Bir seçimde adayların almış oldukları oy sayılar bir dizi içinde tutulmuştur. En fazla oyu alan adayı gösteriniz.

```
private void button1_Click(object sender, EventArgs e)
{
   int[] Oylar = { 21, 4, 6, 7, 8, 9, 2, 61, 43, 5, 55 };
   int EnBuyukOy=0;
   foreach(int Oy in Oylar)
   {
      if(Oy>EnBuyukOy)
      {
         EnBuyukOy=Oy;
      }
   }
   MessageBox.Show(EnBuyukOy.ToString());
}
```



Örnek: Aynı örneği fonksiyon kullanarak yapın. Fonksiyona diziyi gönderin.

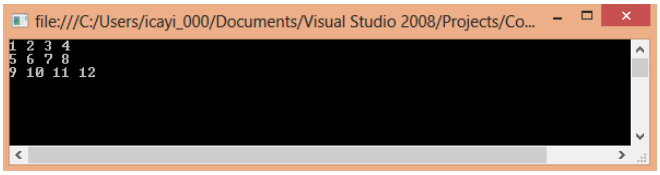
```
private void button1_Click(object sender, EventArgs e)
{
    int[] Oylar = { 21, 4, 6, 7, 8, 9, 2, 61, 43, 5, 55 };
    MessageBox.Show(EnFazlaOyAlaniBul(Oylar).ToString());
}
public int EnFazlaOyAlaniBul(int [] Oylar)
{
    int EnBuyukOy = 0;
    foreach (int Oy in Oylar)
    {
        if (Oy > EnBuyukOy)
        {
            EnBuyukOy = Oy;
        }
    }
    return EnBuyukOy;
}
```



Örnek: 1 den 12 kadar olan sayıları iki boyutlu dizi ve Console Application (dos ekranında) kullanarak ekrana 3 satır ve 4 sütun şeklinde yazdırın. Not: Console Application için Visual Studio da yeni bir proje oluşturulurken WindowsFormApplication yerine Console Application seçilmelidir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int[,] tablo = new int[3, 4];
            tablo[0, 0] = 1;
            tablo[0, 1] = 2;
            tablo[0, 2] = 3;
            tablo[0, 3] = 4;
            tablo[1, 0] = 5;
            tablo[1, 1] = 6;
            tablo[1, 2] = 7;
            tablo[1, 3] = 8;
            tablo[2, 0] = 9;
            tablo[2, 1] = 10;
            tablo[2, 2] = 11;
            tablo[2, 3] = 12;
            for (int x = 0; x < 3; x++)
            {
                for (int y = 0; y < 4; y++)
                {
                    Console.Write(tablo[x, y] + " ");
                }
                Console.WriteLine();
            }
```

```
            Console.ReadKey();
        }
    }
}
```

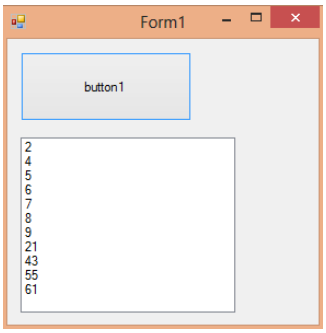


Örnek: Dizileri kullanarak bir sıralama algoritması hazırlayınız.

```
private void button1_Click(object sender, EventArgs e)
{
    int[] sayilar = { 21, 4, 6, 7, 8, 9, 2, 61, 43, 5, 55 };

    for (int i = 1; i < sayilar.Length; i++)
    {
        for (int j = i; j >= 1; j--)
        {
            if (sayilar[j] < sayilar[j - 1])
            {
                int tasiyiciSayi = sayilar[j];
                sayilar[j] = sayilar[j - 1];
                sayilar[j - 1] = tasiyiciSayi;
            }
        }
    }

    for (int i = 0; i < sayilar.Length; i++)
    {
        listBox1.Items.Add(sayilar[i]);
    }
}
```



Fonksiyon Ve Nesne Oluşturma

Gerçek hayatta program yazarken binlerce kod bir araya gelir. Bu kadar kodun düzgün bir şekilde yazılması ve yönetilmesi zor işlemdir. Bunun için yapılacak işlemler parçalara ayrılır ve bu şekilde yönetim sağlanır (divide and conquer-böl ve yönet). Bunun için kullanılacak yöntem, yapılacak işlemleri daha basit yönetilebilir parçalara ayırmaktır.

C# da modüler programlama için fonksiyonlar ve bunların nesne olarak programlanmasıyla gerçekleştirilir.

Fonksiyon Oluşturma

Fonksiyonlar dört farklı tipte oluşturulur.

A- Hiç bir dış değer almayan ve geri değer göndermeyen fonksiyonlar. Sadece işlem yapar	B- Dışarıdan değer almaz fakat geri değer gönderir
public Void FonksiyonAdi() *{* *Yerel degişkenler* *İşlemler* *}*	*public geridönüştipi FonkAdi()* *{* *Yerel degişkenler* *İşlemler* *Return geridönendeğer* *}*
private void button1_Click(object sender,	private void button1_Click(object sender,

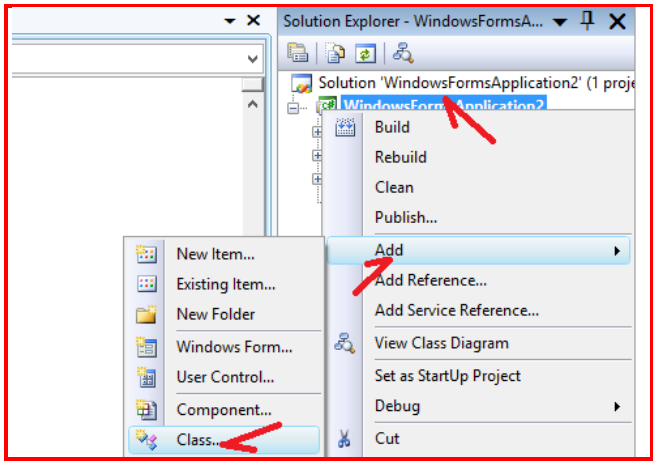
EventArgs e)
{
Hesapla();
}
 	
public void Hesapla()
{
int A=3, B=2, C;
C = A + B;
MessageBox.Show("Sonuc="+C);
}
	EventArgs e)
	{
	MessageBox.Show(Hesapla().ToString());
	}
	
	public int Hesapla()
	{
	int A=3, B=2, C;
	C = A + B;
	return C;
	}

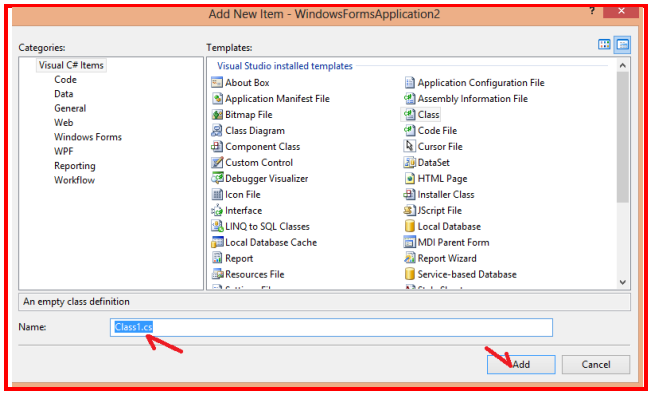
C- Dışarıdan değer alır fakat dışarı değer göndermez	D- Dışarıdan hem değer alır hemde değer göderir.
public Void FonksiyonAdi(tip Degişken1, tip Degişken2) *{* *Yerel degişkenler* *İşlemler* *}*	*public geridönüştipi FonkAdi(tip Degişken1, tip Degişken2)* *{* *Yerel degişkenler* *İşlemler* *Return geridönendeğer* *}*

private void button1_Click(object sender, EventArgs e)
{
int X = 3, Y = 2;
Hesapla(X,Y);
}
 	
public void Hesapla(int A, int B)
{
int C;
C = A + B;
MessageBox.Show(C.ToString());
}
	private void button1_Click(object sender, EventArgs e)
	{
	int X = 3, Y = 2;
	MessageBox.Show(Hesapla(X, Y).ToString());
	}
	
	public int Hesapla(int A, int B)
	{
	int C;
	C = A + B;
	return C;
	}

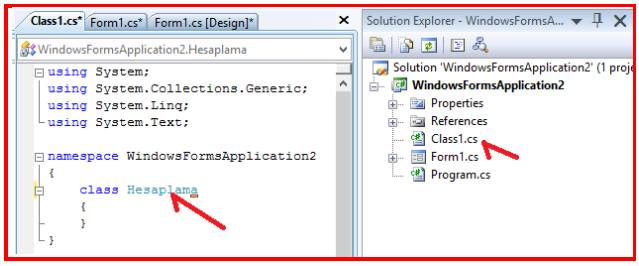
Sınıf (Nesne) Oluşturma

Projemizin içinde tekrar eden kodları bir nesne şeklinde (sınıf=class) şeklinde oluşturarak tekrar kullanabiliriz. Sınıf yapısını projenin her tarafından çağırıp kullanıma açabiliriz. Bunun için öncelikle solution penceresinden projemize sağ tuşa tıklarsak Add>Class yolunu takip edersek sınıf oluşturmak için pencere açılacaktır. Bu pencerede sınıf kodları için isim belirleyebiliriz.





Burada önemli olan Class1.cs isminden daha çok içerisindeki sınıf tanımlayan fonksiyonun adıdır. Bu ad bizim kodlarımızda kullanacağımız isimdir.



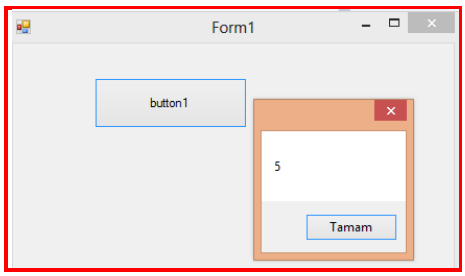
Şimdi bu hesaplama sınıfı içerisine iki tane fonksiyon (metod) yazalım.

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

namespace WindowsFormsApplication2
{
    class Hesaplama
    {
        public double Topla(double A, double B)
        {
            return A + B;
        }
        public double Cikar(double A, double B)
        {
            return A - B;
        }
    }
}
```

Sınıfı projemizde kullanırken şu şekilde tanımlarız.

```
private void button1_Click(object sender, EventArgs e)
{
Hesaplama Islem =new Hesaplama();
MessageBox.Show(Islem.Topla(2,3).ToString());
}
```



Recursive (Özyinelemeli) Fonksiyonlar

Tekrarlı işlemlerin yapımında iki yöntem kullanılabilir. Bunlardan birincisi klasik İterative (tekrarlı) yöntemlerdir. Bu işlemlerde bildiğimiz döngü blokları normal bir şekilde kullanılır. İkinci bir yöntem olarak ise Recursive (özyineleme-kendini çağırma) yöntemidir. Matemetiksel fonksiyon kullanılırken fonksiyon içerisinde yine fonksiyonun kendisi kullanılır. Yani fonksiyon kendi içinde kendisini çağırmaktadır.

Her iki yöntemlede bir üst alma işlemini nasıl yaparız açıklayalım.

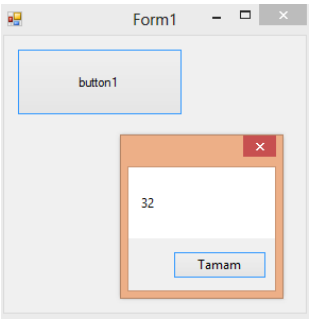
Örnek 1: 2 üzeri 5 kaç eder. Normal fonksiyon kullanarak yada çarpım ile yapınız.

$2^5 = 2 * 2 * 2 * 2 * 2$ işlemi döngü işlemi ile tekrarlı olarak yapılabilir.

Örnek 2: Aynı işlemi recursive olarak şu şekilde yazabiliriz.

```
private void button1_Click(object sender, EventArgs e)
{
   MessageBox.Show(UstAl(2,5).ToString());
}
```

```
public int UstAl(int taban, int ust)
{
   if (ust == 0)
   return 1;
   else
   return taban * UstAl(taban, ust - 1);
}
```



Burada fonksiyon özyinelemeli olarak tanımlanırken bir tane taban sayısının yanına yine özyinelemeli bir fonksiyon konulmuştur. Bu fonksiyon yine kendisidir. İşlemin doğru sonuç verebilmesi için üstlü ifadenin bir noktada sonlandırılması gerekir. Üstlü ifadenin üstü 1 iken sonuç tabana eşit olur, üst 0 iken sonuç 1 e eşit olur. Programı yazarken üstlü ifadenin üstü 1 yada 0 olduğunda işlem durdurulmalıdır. Özetle;

$a^n = a * a^{n-1}$ --> Özyineleme (recursive) işlemi

$a^1 = a$ veya $a^0 = 1$ --> Özyineleme işleminin sonlandırma şartı.

Burada ust=0 olana kadar fonksiyon kendisini çağırmaya devam eder. Bu esnada değişkenler ve fonksiyon yeni değişkenlerle hafızada saklanır. Sonlandırma işlemi doğru olduğunda buradaki return değeri hafızada tutulan bir önceki fonksiyona gönderilir. Ondan çıkan değer tekrar hafızada tutulan bir önceki fonksiyona gönderilir. Bu şekilde başa kadar varılır ve fonksiyon sonucu hesaplanır.

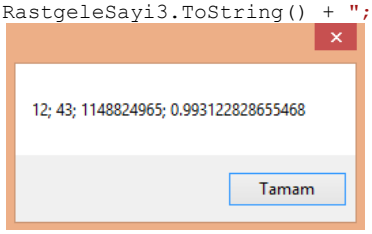
Yukarıdaki işlemde sonlandırmayı (ust==1) return taban şeklindede yapabilirdik.

Rastgele Sayı Üretimi

Rastgele sayı üretmek için öncelikle Random() sınıfından bir nesne üretmeliyiz.

```
   Random Rastgele = new Random();
   int RastgeleSayi1 = Rastgele.Next(10, 20);
   int RastgeleSayi2 = Rastgele.Next(50);
   int RastgeleSayi3 = Rastgele.Next();
   double RastgeleSayi4 = Rastgele.NextDouble();

   MessageBox.Show(RastgeleSayi1.ToString() + "; " + RastgeleSayi2.ToString() + "; " +
RastgeleSayi3.ToString() + "; " + RastgeleSayi4.ToString());
```



Birinci örnekte: 10 ile 20 arasında int türden rastgele bir sayı üretilir, 10 dâhil ancak 20 dâhil değildir.

İkinci örnekte: 0 ile 50 arasında int türden rastgele bir sayı üretilir, 0 dâhil ancak 50 dâhil değildir.

Üçüncü örnekte: int türden pozitif herhangi bir sayı üretilir.

Dördüncü örnekte: double türden 0.0 ile 1 arasında rastgele bir sayı üretilir.

SIRALAMA (SORTING) ALGORITMALARI

Seçerek Sıralama Algoritması (Selection Sort)

Sıralama algoritmaları içinde en basit olanıdır. Eleman sayısı çok olan diziler için etkili bir algoritma değildir. İşleyişi şu şekildedir.

Dizi baştan sona kadar taranır. İçerisindeki en küçük sayı bulunur. Daha sonra en küçük sayı ile dizinin ilk sırasındaki sayı yer değiştirilir böylece en küçük sayı dizinin en başına konmuş olur. Daha sonra ikinci sıradaki sayıdan sona kadar taranır. En küçük tesbit edilir. Tekrar en küçük tesbit edilen sayı dizinin ikinci sırasına konur. Çünkü ikinci en küçük sayıdır. Bu esnada ikinci sıradaki sayıda yer degiştiren sayının yerine atanır. Ardından üçüncü sıradaki sayıdan başlanır sona kadar taranır ve en küçük sayı bulunur. Sona vardıktan sonra bu sayıda dizinin üçüncü sırasına konur. Oradaki sayıda onun yerine geçer. Bu şekilde tüm dizi sıralanır.

5		3		3		3		3		3		3		3		3
9		9		9		4		4		4		4		4		4
3		5		5		5		5		5		5		5		5
4		4		4		9		9		9		9		6		6
6		6		6		6		6		6		6		9		9

Kabarcık Sıralama Algoritması (Bubble Sort)

Sıralama algoritmaları içinde iyi tanınan bir algoritmadır. Büyük dizilerde çok etkili değildir. Mantığı şu şekildedir.

Dizi en baştan başlayarak ilk eleman bir sonraki ile karşılaştırılır. İlk eleman büyükse bir sonraki ile yer degiştirir. Sonra ikinci eleman ele alınır. Bu eleman bir sonraki üçüncü elemandan büyükse yine yer degiştirilir. Her seferinde ele alınan büyük olursa bir sonraki ile yer değiştirerek sona kadar devam edilir. Sona varıldığında dizi içindeki en büyük eleman dizinin en sonuna atılmış olur.

Tekrar başa dönülüp ilk elemandan başlanır ve yine bir sonraki elemanla karşılaştırılır. Aynı yöntemle bu sefer sona kadar devam edilmez, sondan bir önceki elemana kadar devam edilir. Çünkü bir önceki işlemlerde zaten en sona en büyük elelman atanmıştı. Gereksiz yere fazla işlem olmaması için herseferinde sonda yerine konan sayılar ele alınmaz.

Bu şekilde dizi küçükten büyüğe doğru sıralanmış olur.

5		5		5		5		5		5		3		3		3		3		3		3		3		3		3	
9		9		3		3		3		3		5		4		4		4		4		4		4		4		4	
3		3		9		4		4		4		4		5		5		5		5		5		5		5		5	
4		4		4		9		6		6		6		6		6		6		6		6		6		6		6	
6		6		6		6		9		9		9		9		9		9		9		9		9		9		9	

Ekleyerek Sıralama Algoritması (Insertion Sort)

1.Versiyon

İlk versiyonda sıralama işlemi, dizi elemanları daha kullanıcıdan bilgileri alırken sıralama yaparak yerine yerleştirilir. Kullanıcıdan alınan ilk eleman dizinin ilk sırasına yerleştirilir. Yeni alınan her eleman mevcut elemanlarla karşılaştırılır. Hangi elemandan küçükse onun yerine konur ve diğer tüm büyük elemanlar bir adım kaydırılır.

5		9		3		4		6	
5		5		3		3		3		3
		9		5		4		4		4
				9		5		5		5
						9		6		6
								9		9

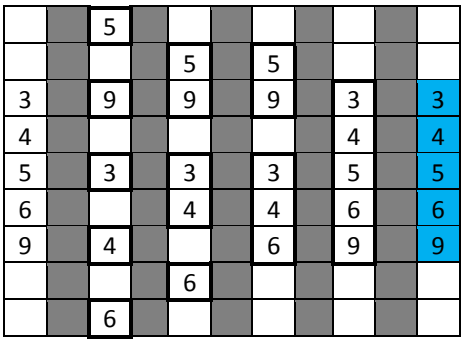
2.Versiyon

İkinci versiyonda ise daha önceden bilinen düzensiz bir dizi üzerinde işlem yapılır. Mevcut dizi sırasız, yeni oluşturulacak dizi ise sıralı dizi olarak kabul edilir. Sırasız diziden alınan her sayı sıralı dizi içerisindeki sayılarla sıra ile karşılaştırılır. Uygun yeri hangi ise oraya yerleştirilir. Böylece eldeki sırasız dizideki tüm elemanlar sırayla alınıp, sırasız diziye yerleştirilmiş olur. Başlangıçta sıralı dizi içinde dizinin ilk elemanı var kabul edilir.

5		5		5		5		3		3		3		3		3		3	
9		9		9		3		5		5		5		4		4		4	
3		3		3		9		9		9		4		5		5		5	
4		4		4		4		4		4		9		9		9		6	
6		6		6		6		6		6		6		6		6		9	

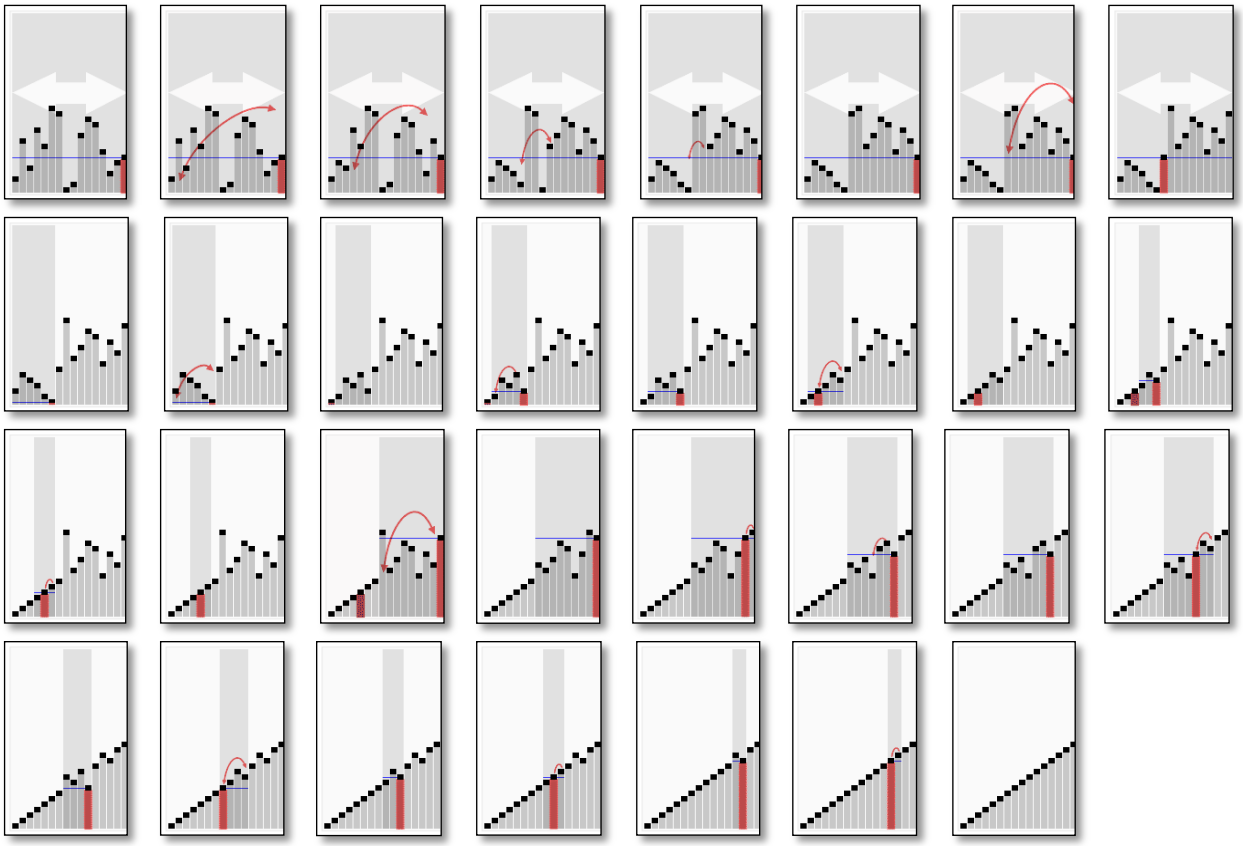
Birleştirerek Sıralama Algoritması (Merge Sort)

Bu algoritmada böl-yönet mantığı kullanılır. Dizi öncelikle elemanlarına bölünür. Daha sonra her iki eleman sıralı olarak yeni bir dizi elde edilir. Bu diziler ikişerli elemanlardan oluşur. Ardından ikişlerli diziler birleşleştirilerek dörder elemandan oluşan yeni diziler elde edilir. Bu birleştirme işlemleri esnasında oluşan her yeni dizi sıralı olarak oluşturulur. Bu şekilde tüm diziler her seferinde daha büyük dizi oluşturularak sıralı tam dizi oluşturulur.



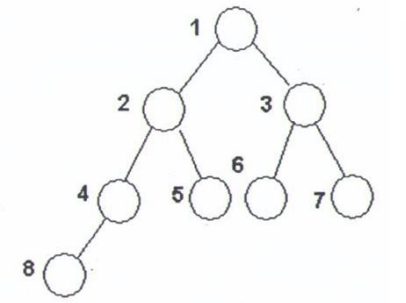
Hızlı Sıralama Algoritması (Quick Sort)

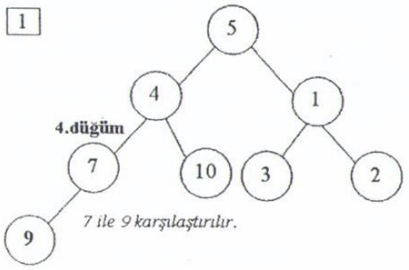
Eleman sayısı fazla olan dizileri sıralamak için uygun bir metotdur. Temel mantık bir elemanın sağındaki sayılar kendinden büyük, solundakiler ise kendinden küçük olmalıdır. Buna göre ilk baştan ele alınan bir sayı (pivot sayı) dizinin tüm elemanları ile sondan başlayarak karşılaştırılır. Kendinden küçük elemanlar solunda, kendinden büyük elemanlar sağında bulunmalıdır. Bu işlemin sonunda iki tane alt dizi ortaya çıkmış olur. Sonra her iki taraftaki alt diziler aynı mantıkla kendi içlerinde bir daha işleme tabi tutulur. Böylece diziler en küçük elemana kadar bölünerek sıralanmış olur.

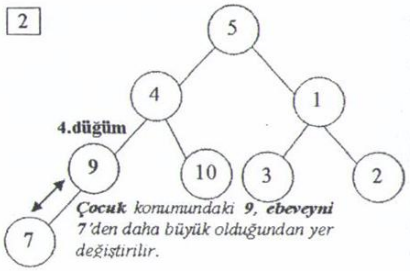


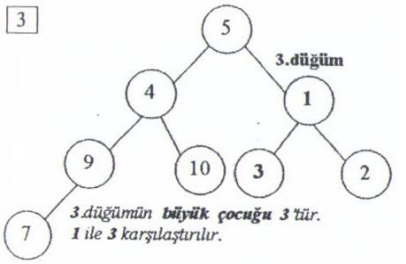
Yığarak Sıralama Algoritması (Heap Sort)

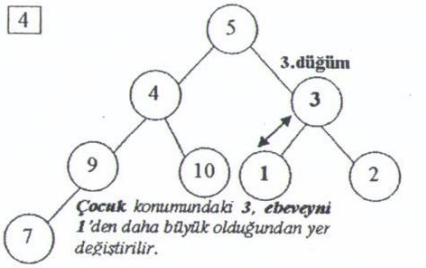
Bu algoritma Merge Sort ve Quick Sort algoritmalarına göre daha yavaş çalışmakta fakat Özyinelemeli (recursive) özelliği olmadığı için fazla sayıda elemanı bulunan dizilerde tercih edilmektedir. Algoritma ağaç yapısı mantığı kullanılarak geliştirilmiştir.

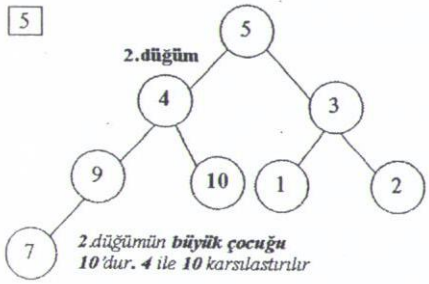

1
2
3
4
5
6
7
8

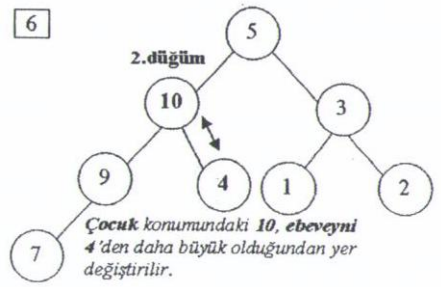

1
4.düğüm
7 ile 9 karşılaştırılır.

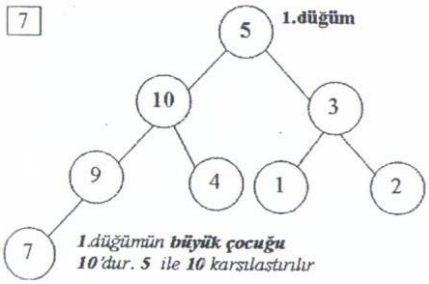

2
4.düğüm
Çocuk konumundaki 9, ebeveyni 7'den daha büyük olduğundan yer değiştirilir.

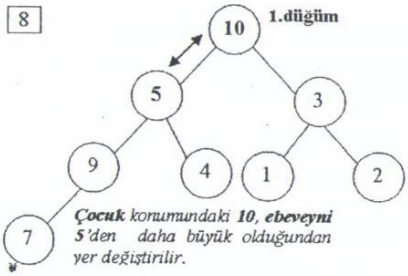

3
3.düğüm
3.düğümün büyük çocuğu 3'tür.
1 ile 3 karşılaştırılır.

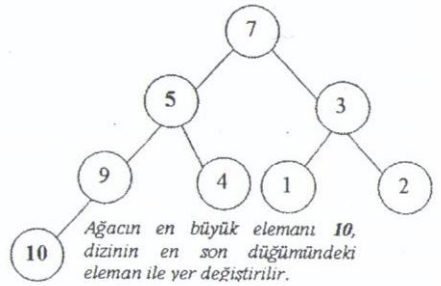

4
3.düğüm
Çocuk konumundaki 3, ebeveyni 1'den daha büyük olduğundan yer değiştirilir.


5
2.düğüm
2.düğümün büyük çocuğu 10'dur. 4 ile 10 karsılastırılır


6
2.düğüm
Çocuk konumundaki 10, ebeveyni 4'den daha büyük olduğundan yer değiştirilir.


7
1.düğüm
1.düğümün büyük çocuğu 10'dur. 5 ile 10 karsılastırılır


8
1.düğüm
Çocuk konumundaki 10, ebeveyni 5'den daha büyük olduğundan yer değiştirilir.


Ağacın en büyük elemanı 10, dizinin en son düğümündeki eleman ile yer değiştirilir.

ARAMA (SEARCHING) ALGORITMALARI

Çok sayıda veri içinden belli özellikteki verilerin bulunma işlemidir. Aranacak verilerin başlangıçta sıralı olup olmaması algoritmanın türünü ve hızını belirlemektedir.

Sırasal Arama (Lineer Search)

Arama algoritmaları içinde en basit forma sahip olan algoritmadır. Aranacak bilgi dizinin ilk elemanından başlayarak her elemanla karşılaştırılır. Eğer sayı dizi içinde bulunursa işleme son verilir. Bu aramanın bir özelliğide sırasız diziler içinden arama yapabilmesidir. Örnekleyelim;

3=?	5		5		5	
	9	3=?	9		9	
	3		3	3=?	3	Ok!
	4		4		4	
	6		6		6	

İkiye Bölerek Arama (Binary Search)

Sıralı diziler içinde arama yapan etkili bir algoritmadır. Mantığı şu şekildedir. Dizi ilk olarak ikiye bölünür. Dizinin ortadaki elemanı bulunup aranan eleman ile karşılaştırılır. Eğer aranan eleman ortadaki eleman eşitse işlem durdurulur. Şayet aranan eleman ortadaki elemandan küçükse dizinin sol tarafındadır. Aranan eleman dizinin orta elemanından büyükse sağ tarafta demektir. Dizinin sol tarafında ise sol taraftaki elemanlar tekrar aynı işlemlere tabi tutulur. Burada dizi elemanlarının küçükten büyüğe doğru sıralanmış olduğu varsayıldı. Örnek dizi üzerinde inceleyelim.

	15		15		15	
	19		19	19=?	19	Ok!
	33	19=?	33		33	
	45		45		45	
19=?	67		67		67	
	72		72		72	
	84		84		84	
	91		91		91	
	99		99		99	

Ödevler

1. Kullanıcıdan alınan N adet sayının içinde en büyük ve en küçük sayıları bulan programı yazınız.
2. Yıllık %8 zam artışı ile 1000 maaş alan bir kişinin maaşı 3000 liraya kaç yıl sonra ulaşır.
3. Kenar Uzunlukları verilen bir üçgenin hangi tip üçgen olduğunu veren programı yazınız.
4. Girilen bir metnin içindeki sesli harfleri çıkarıp, cümleyi sessiz harflerle yazan programı yazınız.
5. Bir otopark için arabaların kalış süresine göre ödemeleri gereken parayı hesaplayan bir program yazınız. İlk saat için 2 TL, daha sonraki her saat için 1 TL ücret talep edilmektedir. Araçların giriş ve çıkış saatleri programa girilmektedir.
6. 0-9999 arasındaki sayıları Roma rakamına dönüştüren programı yazın. (1000=M, 500=D, 100=C, 50=L, 10=X, 5=V, 1=I)
7. Programa girilen iki sayının OBEB (Ortak En Büyük Bölen Sayı) kuralını sağlayan sayıyı bulunuz. Bulunan en büyük sayı her iki sayıyıda tam olarak bölen en büyük sayı olmalıdır. Örnek, 12 ile 18 sayıları girildiğinde cevap 6 dır.
8. Aşağıdaki fonksiyonu hesaplayan bir program yazınız.

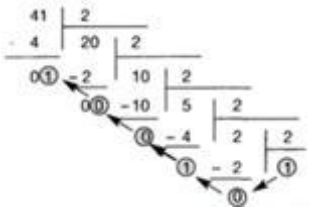
$$f(n) = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

9. 10x10 luk çarpım tablosunu oluşturan bir program yazınız. Matris görünümünde olsun. Tablonun gösterileceği nesneyi kendiniz belirleyiniz.
10. Girilen bir N değerine bağlı olarak **$1+X/1! + X^2/2! + X^3/3! +...+ X^{n-1}/(n-1)!$** Serisini hesaplayan programı yazınız.
11. Şekildeki gibi girilen N sayısına bağlı olarak Eşkenar dörtgeni oluşturan programı fonksiyon kullanarak yazınız. Gösterim için kullanacağınız hazır nesneyi ve tasarımı kendiniz belirleyiniz.

		1		
	2	2	2	

3	3	3	3	3
	2	2	2	
		1		

12. 0 ile 1000 sayıları arasında pisagor üçlülerini oluşturan üçlü grupları alt alta listbox'a yazan fonksiyonu yazınız. Örneğin 5,4,3 rakamları dik üçgenin kenarlarını oluşturan pisagor üçlüleridir.
13. Onluk tabandaki bir sayıyı ikilik tabanda bir sayıya dönüştüren fonksiyonu yazınız. Bilgi: Onluk tabandaki bir sayı ikilik tabanda bir sayıya dönüştürülürken sayı daima ikiye bölünür. Her bölümde kalan sayılar tersten başa doğru yazılarak ikilik tabandaki sayıyı oluştururlar. Örneğin 41 sayısının ikilik tabandaki karşılığı 101001 dir.



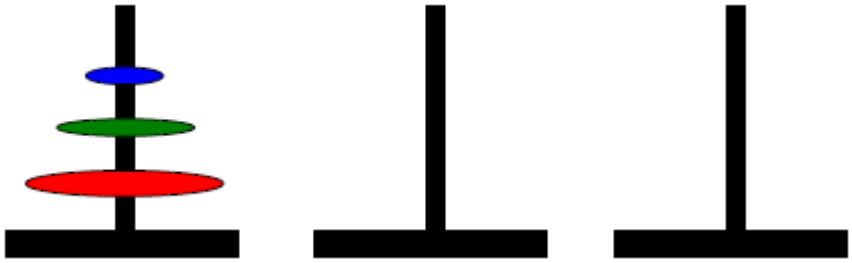
14. Pi sayısı matematikte önemli bir yere sahiptir. Bu sayının tam bir değeri yoktur. Virgülden sonraki hanesi sonsuza kadar devam etmektedir. Bu sayının hesaplaması ile ilgili aşağıda üç seri verilmiştir. Serileri bilgisayar hesaplarken bulmuş olduğu Pi sayısı değerlerini grafik halinde (yada listeleme) şeklinde gösteriniz. 1 milyon iterasyon sonucunda üç formülünde bulmuş olduğu pi sayısının sonucu nedir?. Virgülden sonra 25 haneye kadar görmek istersek değişkenleri nasıl tanımlarsınız. Virgülden sonra 25 haneye kadar Pi sayısını gösteriniz.

    Pi=4*(1-1/3+1/5-1/7+1/9...)

    Pi=sqrt(6+6/22+6/32+6/42...)

    Pi=4*2/3*4/3*4/5*6/5*6/7*8/7...)

15. Sayı Tutma Oyunu Programlayınız. Program iki şekildede çalışabilsin. Bilgisayar bir sayı tutsun kişi bu sayıyı bulmaya çalışsın. İkinci olarakta kişi bir sayı tutsun Bilgisayar bunu bulmaya çalışsın. Kaç adımda bulduğunu program söylesin. Programın nasıl çalışması gerektiğine kendiniz karar veriniz.
16. Verilen limitler arasında Kusursuz sayıları bulan programı yazınız. Kusursu sayı şu demektir. Sayının asal çarpanlarının toplamı kendisine eşit ise bu sayı kusursuz sayıdır. Örneğin 6 sayısını ele alalım. 6 sayısının asal çarpanları 3, 2, 1 dir. Bunların toplamı yine 6 sayısını verecektir. Asal sayı şu demektir. Bir sayıyı kalansız bölen sayılar asal sayılardır. Bu üç sayı 6 rakamını kalansız böler.
17. Şekildeki gibi 3 adet çubuk vardır. Birinci çubukta rastgele farklı çaplarda (hiç biri aynı çapta degil) N adet disk konulmuştur. Bu diskler en büyük çaptan küçüğe doğru ikinci çubuğa dizilmek isteniyor. Yardımcı olması için geçici olarakda üçüncü bir çubuk konulmuştur. Her bir diski çubuğa takmak 1 saniye alırsa tüm N adet diski 2 nolu çubuğa yerleştirmek toplam kaç saniye alır (kaç defa çubuğa takma işlemi gerçekleşir?) Problemi görsel olarak yada ListBox'lar içinde sayılarla göstererek yapınız.



18. Bir görevi vermek üzere bilgisayarda kura çekilecektir. Fakat kurada kişilerin tahmin ettikleri sayılar kullanılacaktır. Diyelim 5 kişi bu göreve talip olmakta. Herkese 1-100 arasında bir sayı atması söylenmekte. Bilgisayar da 1-100 arasında sayılar atarak bu kişilerin tahmin ettiği sayıları denk getirirse o kişiye +1 puan ekleyecek. Bu şekilde ilk 10 puana kim ulaşırsa o kişiye bu görev verilecek. (Burada mantık, tamamen bilgisayarın kişileri seçmesi yerine, onların atmış oldukları sayılarıda kullanarak işin içine kişileri de katmaktır). Örnekleyelim. 5 kişi şu sayıları atmış olsun. 23, 48, 73, 81, 99. Bilgisayar ilk sayı atmasında 11 atarsa bu işlem pas geçilir, daha sonrakinde 73 atarsa o kişinin puanı bir artar. Bu şekilde ilk 10 puana bir kişi ulaşana kadar işleme devam edilir.
19. Rakam formatında girilen sayıları yazı formatına dönüştüren algoritmayı geliştirip programlamasını yapınız. Tüm bilgiler dizilerde tutularak yapılacaktır, En az satır ile çözümü gerçekleştirmek esastır (Satır sayısını belirleme kriteri (;) işaretidir).

Projeler

1

Yıl sonu projesi olarak aşağıdaki konu üzerinde araştırma yapıp algoritma geliştiriniz ve makale formatında teslim ediniz.

Konu: Sıralama algoritmalarını programlayıp performansları üzerine bir araştırma yapınız. Performans ölçümü için Az veri, Orta seviye veri ve yüksek seviye verilerdeki performansını araştırınız. Performans ölçümleri için Ram ve CPU kaynaklarını kullanma, geçen zaman ve iterasyon sayısı açısından değerlendirerek performans ölçümlerini karşılaştırmalı olarak grafik şeklinde görüntüleyiniz. Ayrıca mevcut algoritmaları daha da iyileştirmek için yeni yöntemler geliştirmeye çalışınız yada sıfırdan yeni bir yöntem bulabiliyorsanız bunun peformansınıda diğerleri ile karşılaştırınız.

İpucu: Öncelikle mevcut algoritmalar programlanıp karşılaştırmalar yapılır. Zayıf algoritmaların neden daha zayıf olduğu konuları ve güçlü algoritmaların neden daha güçlü olduğu üzerinde araştırmalar yapılır. Bundan sonra daha da ileriye götümek için ne gibi algoritma fikirleri geliştirilir bunlar üzerine çalışılır. Aşağıda linkte algoritmaların performanlarının nasıl ölçüleceğine dair bilgiler mevcuttur.

http://www.algoritma.org/2011/05/algoritma-karmasklg-ve-buyuk-o.html

2

C# programlama yeteneğinizi geliştirmek için aşağıdaki üç programı yazınız.

1.Girilen bir metnin içindeki sesli harfleri çıkarıp, cümleyi sessiz harflerle yazan programı yazınız.

2.0-9999 arasındaki sayıları Roma rakamına dönüştüren programı yazın. (1000=M, 500=D, 100=C, 50=L, 10=X, 5=V, 1=I)

3.Bir otopark için arabaların kalış süresine ve tonajına göre ödemeleri gereken parayı hesaplayan bir program yazınız. Bu olayı simule eden bir program yazınız. Araçlar otoparka rastgele değişik tonajlarda gelsin. Farklı sürelerde rastgele kalsın. Her çıkan araçtan sonra ödenecek para miktarı kasa ekranında gözüksün. Bunun için 3 ekran olabilir. Birinci ekranda araçlar giriş yapar (Değişik araç resimleri gösterilir). İkinci ekranda çıkış yapan araçlar gösterilir. İçeride kalış süreleri süreyi kısa tutmak için saniye ile ölçülebilir (rasgele). Üçüncü ekrandada aracın Özellikleri ve fiyat gösterilir. Bunun girişte her araca bir ID numarası verilebilir. Çıkıştada bu ID numarasına göre özellikleri ve fiyatı gösterilirilebilir. Gerçek yaşamı düşünerek kendinize göre uygun kararlar alarak böyle bir programı yazın.

3

a) Üçüncü derece ve ikinci derece 2 tane fonksiyon belirleyin. Bu fonksiyonların mak ve min noktalarını hesapla anlatarak bulun. Excelde grafiğini çizdirip mak ve min ları gösterin. b) Notlardaki örneğe benzer şekilde Ankastra olarak bağlanmış bir I profilin optimum tasarımını yapın. Kirişin boyu 1 metre olacak. Uç kısmından F1=3000 N luk çekme kuvveti, F2=2000 N luk ta eğme kuvveti olacak. Kirişin akma dayanımı 250 MPa alınacak. Buna göre bu yükleri taşıyan en hafif I profil kirişin boyutlarını belirleyin. Profilin kesit alanının ölçüleri konusunda kısıtlama yoktur. Sadece şekli I şeklinde olmalıdır. I şeklindeki alanın atalet momenti nasıl hesaplanır öğrenin. Paralel eksenler teorimine bakın. Grafiği çizebilmek için program yazmanız gerekecektir. Elle yüzlerce noktanın hesaplanması zordur. Excelde yapmak da imkansızdır. Grafikteki eşdeğer eğrileri çizerken renk skalası kullanabilirsiniz.