PROGRAMLAMA TEMELLERİ

DERS NOTLARI

Dr. Öğr. Üyesi Mahmut ÜNVER

KIRIKKALE MESLEK YÜKSEKOKULU

PROGRAMLAMA DİLİ ÖĞRENMEK

Programlama dili öğrenmeyi yabancı dil öğrenemeye benzetebilirsiniz. Mesela her ikinde de öğrenilmesi gereken, o dile özgü bir grup kelime var. Anahtar kelime diye adlandırdığımız bu kelimelerin c# dilindeki sayısı 77 tanedir. C dilinin geleneğini sürdürdüğü için de bu kelimeler küçük harflerle yazılır.

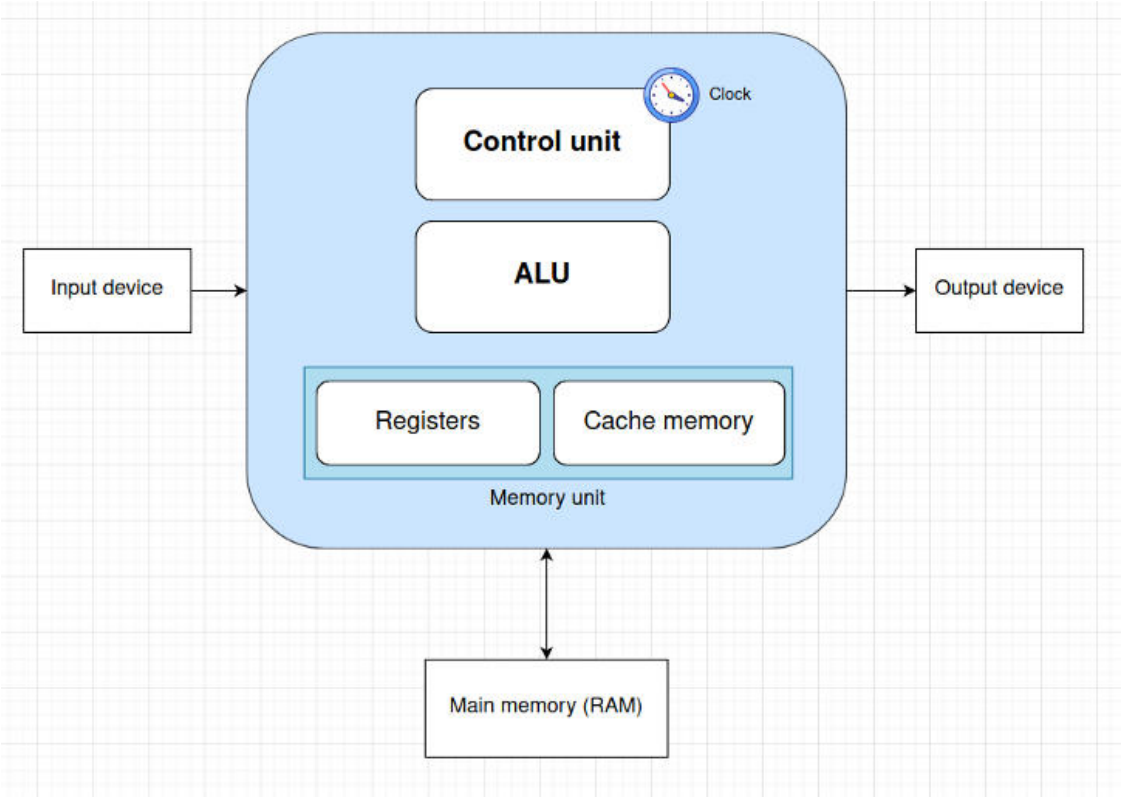
Yabancı dil öğrenirken bir takım yazım ve dil bilgisi kuralları öğrenmek zorundayız. Programlama dilinin yazım ve dil bilgisi kuralları konuşulan dillerin kurallarına göre daha katıdır.

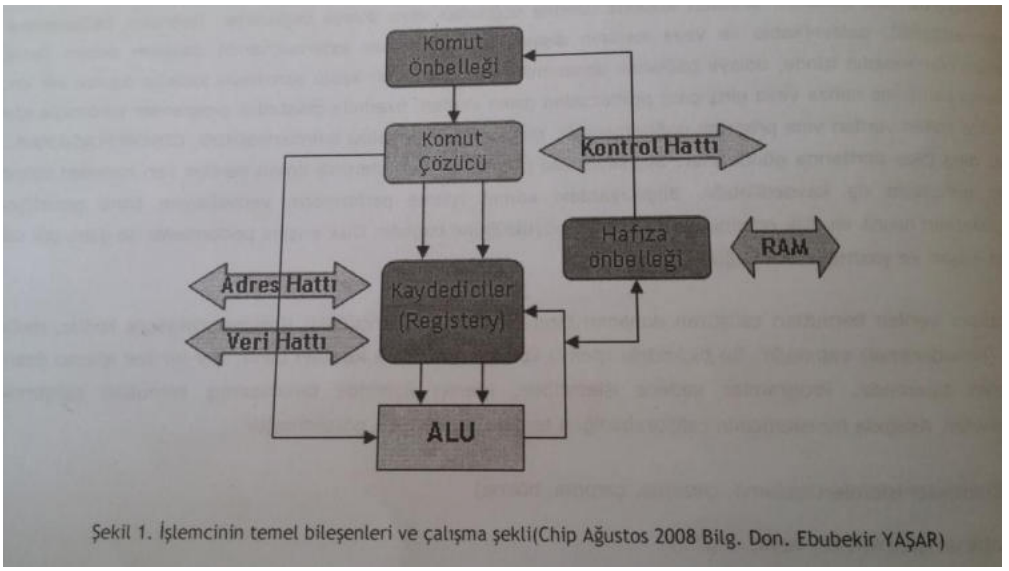
Programlama dili ile yabancı dil öğrenme arasındaki en büyük benzerlik bence her ikisinin de okuyarak öğrenilememesidir. Pratik yapmak zorundayız. Bir müzik aletini çalmayı öğrenirken olduğu kadar çok pratik yapmalıyız hem de. Bir programcı gibi düşünmeye ve kendi kodumuzu kendimiz yazmaya alışmalıyız.

Programlama bir problem çözme etkinliğidir. Programcının tek en önemli kalite göstergesi problemleri bileşenlerine ayırma ve daha küçük parçalara bölerek çözebilme yeteneğidir. Programcının sorumluluğu bu küçük parçaları bir araya getirip çözümü oluşturmaktır. Bu beceri de bence sadece pratik yaparak gelişir.

CPU nedir? Nasıl çalışır?

CPU-Central Processing Unit-Merkezi İşlemci Birimi'nin çalışma prensibi nasıldır?





Şekil 1. İşlemcinin temel bileşenleri ve çalışma şekli(Chip Ağustos 2008 Bilg. Don. Ebubekir YAŞAR)

Bilgisayarın en önemli bileşenidir. **Transistor** denilen yarıiletken elektronik malzemelerden oluşurlar. Kendisine sırasıyla verilen aritmetik ve mantıksal komutları yapar.

Tüm donanım birimlerini emrinde kullanabilir ve yönetebilir. İşlemcinin matematiksel ve mantıksal işlemleri yapan birimine **ALU(Arithmetic Logic Unit)** denir.
İşlemcinin **ALU** dan geriye kalan yapısını, hafıza birimleri, veriyoları, çeşitli kontrol ve denetleme bileşenleri oluşturur. Günümüzde Intel, AMD, Motorola, VIA gibi işlemci üreticileri vardır.

Mikrodenetleyici

Mikroişlemci, bir bilgisayar sisteminde programların işletilmesinden sorumlu olan ve tüm bileşenleri merkezi şekilde kontrol eden entegre devredir.

Mikro işlemcinin temel işlevi, işlenen ve kullanılan verileri birimler arasında iletmek, iletilen verileri işlemek, verileri bir durumdan diğerine çevirmek ve verileri saklamak. Mikroişlemci, işlemci ana işlem biriminin (CPU- Central Process Unit) fonksiyonlarını tek bir yarı iletken tümdevrede (IC-Integrated Circuit) birleştiren programlanabilir bir sayısal elektronik bileşendir. Kullanıcı ya da programcı tarafından yazılan programları meydana getiren komutları veya bilgileri yorumlamak ve yerine getirmek için gerekli olan tüm mantıksal devreleri kapsar. Bu devreler genelde transistörlerden meydana gelmektedir.

İşlemcinin Temel Bileşenleri

1. ALU(Aritmetik ve Mantıksal İşlem Birimi): Toplama çıkarma, çarpma, bölme, mantıksal ve, veya, değil komutları ve kaydırma komutları.

2. Komut Çözücü(Instruction Decoder): İşlemcinin yapması gereken kodların icrası için gerekli işlemleri başlatır ve komutun çalıştırılması için gerekli işlemleri belirler.

3. Kaydediciler(Registery): İşlemci içerisinde sayıları depolamak için kullanılan hafıza çeşididir. İşlemci veri uzunluğu kadar genişliğe(32, 64 bit) sahiptirler. Literatürde test, EBX, EAX, BX, ES, IP gibi isimler alan kaydedici hafıza gözleri vardır.

4. Bayraklar(Flags): İşlemlerin sonucuna göre 1 ya da 0 değerlerini alan 1 bit genişliğe sahip hafıza gözleridir. Sıfır, işaret, elde, eşlik, taşma gibi çeşitleri vardır. Örneğin bir çıkarma işleminde sonuç sıfır çıkarsa sıfır bayrağı 1 değerini alır.

5. Veriyolları(Buses): İşlemcinin diğer donanım birimleri ile bağlantısını sağlayan iletken elektriksel yollardır. Üç adet veriyolu bulunur. Bunlar veri(data), adres(address) ve kontrol(control) veriyollarıdır.

İşlemcinin Temel Bileşenleri(2)

İşlemciler üretilirken kendilerine yüklenen komutları istenildiğinde yapabilme kabiliyetine sahiptirler. Bu komutlar genel işlemlerin icrası için üç gruba ayrılırlar. Bir bilgisayar kendisinde tanımlı olmayan komutları icra edemez.

- **Matematiksel** ve **Mantıksal** İşlem komutları. Toplama çıkarma, çarpma, bölme, mantıksal ve, veya, değil komutları ve kaydırma komutları.
- Verilerin hafıza veya kaydediciler arasında transfer edilmesini sağlayan komutlar. **Hafıza** ve kaydediciler kendi aralarında veya karşılıklı veri transferi.
- Karar verme ve istenen komut satırına dallanma komutları. Sayıların karşılaştırılarak, **pozitif, sıfır, eşitlik, negatif...** durumlarının oluşumuna göre istenen komuta dallanabilme.

İşlemciler komutları yürütürken öncelikle işletilen komut sırasını üzerinde tutan program sayıcının(PC=program counter) gösterdiği adresteki komut **RAM** den alınır(Fetch). Alınan komut, komut çözücü tarafından, nasıl yürütüleceği ve ne anlama geldiği belirlenir(Decode). Sonunda ise çözülen komut doğrultusunda ALU ya verilen direktifler yardımıyla istenen işlemler yaptırılır(Execute). Elde edilen sonuçlar istenen

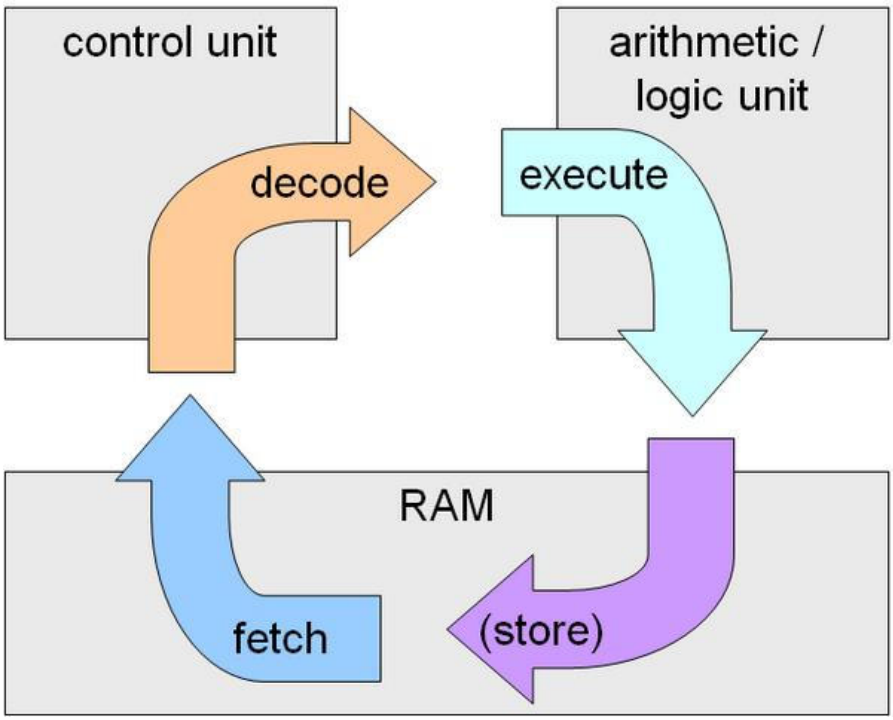
hafıza gözlerine yazılır(Write Back). Bu işlemler bir sonraki komut için benzer şekilde devam ederek işletilmesi gereken komutlar bitene kadar sürer.

Yukarıda bahsedildiği üzere komutların icra edilmesi 4 aşamada gerçekleşmektedir. **Fetch(F), Decode(D), Execute(E) ve WriteBack(WB)** aşamalarıdır. Eğer bir komutun icrası önceki komutun write back aşamasından sonra başlarsa burada zaman kayıpları oluşur. Çünkü işlemcinin örneğin fetch sırasında **ALU** çalışmaz boşta kalmış olur. Zaman kayıplarını önlemek için pipeline(kesintisiz iş akışı) denilen bir yapı kullanılır. Yani aynı anda işlemcinin boşta kalan tüm birimleri kullanılmış olur. Bu kayıp aşağıda gösterilmiştir. 12 saat frekansında sadece üç komut icra edilirken pipeline yapıda 4 komutun icrası 7 saat frekansında halledilir.

Mikroişlemcilerin komut işleme mantığı

Bir mikroişlemcinin komut işleme döngüsü aşağıdaki gibidir. İlk olarak bellekten veri fetch edilir. Kontrol birimi komutu decode eder ve aritmetik işlem birimine devreder. ALU'da komut

çalıştırılarak işletilir ve tekrar belleğe yazılır



Yarı İletken Bellekler

Yarı iletken bellekler 4 çeşittir

1. RAM (yaz-oku)
2. ROM (sadece-oku)
3. PROM (programlanabilen ROM)
4. EPROM (siline bilen PROM)

Mikroişlemcinin Görevleri

•Sistemdeki tüm elemanlar ve birimlere zamanlama ve kontrol sinyali sağlar.
•Bellekten komut alıp getirir.
•Komutun kodunu çözer.
•Komutun operandına göre, veriyi kendisine veya G/Ç birimine aktarır.
•Aritmetik ve mantık işlemlerini yürütür.
•Program işlenirken, diğer donanım birimlerinden gelen kesme taleplerine cevap verir.

İşlemci Parametreleri

1. Hız: İşlemcinin en önemli parametresi komutları işleme hızıdır. Birimi frekans olarak GHz katsayısı(10003) ile değerlendirilir. İşlemcinin hızlı olması işlemlerini daha kısa sürede tamamlaması anlamına gelmektedir. Günümüzde 1.8, 2.0, 2.2, 2.4, 2.6, 3.0, 3.2, 3.6, 3.8 GHz hızlarında olanları vardır.

2. Bit Genişliği: İşlem yapabilme boyutunu gösterir. Günümüzde 64 ve 32 bit işlemciler vardır. İşlemcinin sahip olduğu kaydediciler, veri hattı ve adres hattının genişliğini gösterir.

3. FSB Hızı: İşlemcinin, kuzey köprüsü ile iletişim hızını gösterir. Günümüzde 1333, 1066, 800, 533 Mhz değerlerine sahip işlemciler vardır.

4. Level2(L2) Cache: İşlemciye yakınlığından dolayı bu isim verilir. En yakın olana L1, diğerine L2, L3 gibi isimlendirmeler kullanılır. Yapısı SRAM hafıza tipindedir. SRAM hafızalar daha hızlı fakat maliyetleri yüksek hafıza çeşitleridir. Yavaş olan RAM erişimlerini azaltmak için işlemci içerisinde yer alan daha hızlı fakat küçük boyutlardaki hafızaya verilen isimdir. İşlemcinin hafıza kontrol devresinden(MCH) istekte bulunduğu her kod bu belleğe yazılır. İşlemci aynı kodu RAM yerine daha hızlı olan bu bellekten alır.

Eğer bu bellekte olmayan bir kod isteği gelirse MCH uzun süre kullanılmayan kod yerine yenisini yerleştirir. Böylece cache belekte sık kullanılan kodlar kalarak, ortalama bellek erişimi hızlanmış olur. Günümüzde **Athlon 64 X2 Dual** işlemcilerde 2x1MB, P4 lerde 2MB, Core2 Dualarda 4MB, Core2 Quad larda 8MB ve Celeron larda 1MB veya 512 KB L2 bellek miktarı vardır. L2 miktarı ne kadar fazla olursa o kadar çok kod saklanabilir. Fakat uygulama belirli kodları daha sık kullanıyorsa bu büyük **L2** kapasitesinin faydası olmayacaktır. RAM'den tipik olarak 4 kat daha hızlı çalışmaktadır.

5. Çekirdek(Core) Sayısı: İşlemci paketi içerisinde birbirinden bağımsız olarak komutları çalıştırabilen her yapıya çekirdek ismi verilmektedir. Gerçek zamanlı olarak kendisine verilen işa kışı görevlerini aynı anda yerine getirerek performansı artırır. Her

çekirdek birbirinden bağımsız FSB ye sahiptir. Tüm çekirdekler L2 yi ortak kullanırlar. Günümüzde 2 ve 4 çekirdekli işlemciler bulunmaktadır. Intel Pentium Dual Core, Intel Pentium Core 2 Duo, Intel Quad Core ve AMD Athlon X2 serisi çok çekirdekli yapıya sahiptir.

6. Soket Yapısı: Anakartlar konusunda da değinildiği gibi işlemci pin yapısı ile anakart soket yapısı birbirinin aynısı olması gerekir. Anakart konusuna bakabilirsiniz.

PROGRAMLAMA

Charles Petzold, bilgisayar devrimini iki evreye ayırır: Birinci evre bilgisayarların tasarlandığı ve yapıldığı zaman, ikinci evre ise onlarca yıl sonra bilgisayarların programcı olmayan kimseler tarafından da kullanılabilir olmasıdır.

İlk programlanabilir bilgisayar 1930'lu yıllarda yapıldı. Uzun bir süre programlama işlemi delikli kartlar gibi mekanizmalar kullanılarak sağlandı. Bilgisayarın kendisine donanım, üzerinde çalışan kodlara da yazılım adı verildi. Bir kaç dekad boyunca donanım ve yazılım birbirine bitişikti. Her makinanın kendine özgü bir komut seti vardı ve bu komutlar başka makinalarda çalışmazdı. Sayısal tümleşik devreler kullanıldığı dönemlerde de bu durum devam etti.

1950'lerin başında mimariden bağımsız bilgisayar dilleri geliştirilmeye başlandı. Bu diller Makine kodu ile karşılaştırıldığında yüksek seviyeli diller olarak aklandırıldı. COBOL, BASIC, FORTRAN gibi diller hala farklı sürümleri ile hayattadır.

Derlemek, Yorumlamak

Bilgisayar sadece Makine kodunu çalıştırabilir. Öyleyse yüksek seviyeli dillerle yazılan program kodları Makine konuda dönüştürülmelidir. Derleyici, bir programı tamamen Makine koduna çevirirken yorumlayıcı bu işlemi eş zamanlı olarak yapar.

Bilinen en eski programlama dillerinden birisi de ALGOL dilidir. 1950'li yıllarda uluslar arası bir komite tarafından geliştirildi ve uzun yıllar değişiklikler yapılarak kullanıldı. Hâlâ Pascal, PL/I ve C gibi dillerin temelini oluşturduğu için yaşamını sürdürmektedir.

C DİLİNİN SERÜVENİ

Bell Laboratuarlarının etkisi olmadan çağdaş dünyayı düşünmek imkansızdır. 1947 yılında transistor orada icat edildi. 1970'lerde UNIX işletim sistemi orada geliştirildi.Uzun yıllar, büyük bir kısmı Dennis Ritchie tarafından geliştirilen C dili, UNIX işletim sistemi ile ilişkilendirilir. C dili tek harfle temsil edildiğinden bu adın nereden geldiği merak edilebilir: İlk olarak CPL (Combined Programming Language) dili vardı. BCPL (Basic CPL) dili daha sonra CPL dilinden esinlenerek geliştirildi. Sonraları BCPL dilinin basitleştirilmiş bir hali olan B dili ortaya çıktı. C dili de bu B dilinden türetildi. C ismi oradan gelir.

ALGOL ve onun türevi olan dillerde, program parçaları BEGIN ve END kelimeleri arasına yazılır. C ise bu işi küme parantezlerini kullanak yapar. C'de yazılmış programların en belirgin farkı hızlı olmalarıdır. Çünkü C işaretçiler dediğimiz, bellek bölgelerine doğrudan erişmeye yarayan yapılara izin verir. Bazıları C'ye yüksek seviyeli assembly dili der. Çünkü C; bit, byte ve bellek düzeyinde çok iyi çalışır.

Aslında bu yaklaşımın tehlikeli tarafı da var. Yüksek seviyeli dillerin derleyicileri program kodları derlenirken programın çökmemesi ve veri kayıplarına neden olmaması için fazladan kodlar eklerler. C derleyicisi ise hızlı program ortaya koyma adına, fazladan kod eklemez. Bu yüzden, dikkatli davranılmadığında hata oluşturmaya daha elverişlidir. Hatasız program olmaz ise de, C'de bu durum daha yaygındır. Hataların çoğunluğu işaretçi kullanmanın yol açtığı hatalardır.

C, hala çok yaygındır ama bazı açılardan tarih olmuştur. Geleneksel prosedürel diller grubuna dahil edilmektedir. Program içerisinde belirli bir işi yapan ya da algoritmayı gerçekleştiren program parçasına prosedür denir. Bir C programı da, bir grup prosedür ya da fonksiyondan oluşur. Fonksiyonlar, devamlı veriler ile çalışır. Prosedürel yapıda bu veriler ile program kodu iç içe bulunur.

Nesne yönelimli programlama terimini çok sık duymaya başladık. Nesne yönelimli programlama dillerinin (OOP) ilki Palo Alto Araştırma Merkezinde (PARC) geliştirilen SmallTalk dilidir. PARC, Microsoft Windows ve Apple Machintosh'ta da kullanılan grafiksel kullanıcı arayüzü kavramlarını geliştiren Xerox tarafından kurulan araştırma laboratuarıdır.

Nesne yönelimli dilde, programcılar prosedürler yerine sınıflar oluştururlar, kodların ve verilerin birleşiminden oluşan nesneler de bu sınıflardan türetilir. Programlamadaki bu bakış açısı değişimi, çeşitli programlama işlerinde tekrar tekrar kullanılabilecek kodlar yazmaya olanağı sağladı.

C dilinin de nesne yönelimli sürümünü yazmak isteyenler oldu. 1980lerin başında Bjarne Stroustrup tarafından Bell laboratuarlarında C++ (Si plas plas okunur) geliştirildi. C++ dilinin adında geçen ++, C dilinde bir sayıya 1 eklemek için kullanılır, yani sayının değeri 1 artar.

C++ dili, C'nin tüm özelliklerine sahip ama nesne yönelimli olması için bir takım ek özellikler katılmış hali diye özetlenebilir. Ama bu yapı biraz hantal bir yapı oldu. Çünkü C'de olan her şeyi destekleme kaygısı ile garip bir notasyon ortaya çıktı.

Bu hantal yapının yanında, baş edilmesi gereken bir sorun daha vardı. Bilgisayar donanımı hızla gelişiyor. Gereğinden fazla hızlı işlemciler ve gereğinden fazla bellek var. Bir program yazılırken dikkat edilen temel kriterler değişti. Eskiden performans ve ekonomik kaynak kullanımı iken şimdilerde yerini daha hızlı ve hatadan arınmış kod yazmaya bıraktı. Bu da alçak seviyeli yapıdan hızla uzaklaşma ve işaretçi kullanımının azalması demekti. Artık programcılar işaretçi kullanmaktan kaçınıyorlar.

1990'larda Sun Microsystems, C++ dilinden daha esnek bir notasyona sahip olan, daha güvenli kod yazmayı olanaklı kılan bir yapı ortaya koydu, yani Java dilini geliştirdi.

C#'ın Doğuşu

Borland firmasından ayrılan Anders Hejlsberg, 20002li yılların başında Microsoft için C# (Si Şarp okunur) dilini geliştirdi. C#'taki diyez işareti (şarp), C++ taki ++ ın 2 tanesinin üst üste koyulmuş halidir.

2001 yılında ortaya çıkmış olup Microsoft tarafından geliştirilmiştir. Tüm metodlar sınıflar içeirisinde tanımlanır.

C# Java gibi, C# da C'nin tehlikeli özelliklerini almadı. Ama işaretçiler C#'tan tamamıyla çıkarılmadı. Java ile C# arasındaki diğer bir benzerlik, derleyicinin rolündedir. Geleneksel olarak, derleyici kaynak kodu (yüksek seviyeli dil metin dosyası) makina koduna çevirir. Makina kodu, bilgisayar tarafından çalıştırılabilir bir formdadır. Bu da Microsoft Windows ile Apple Machintosh üzerinde aynı programın çalıştırılamayacağı anlamına geliyor.

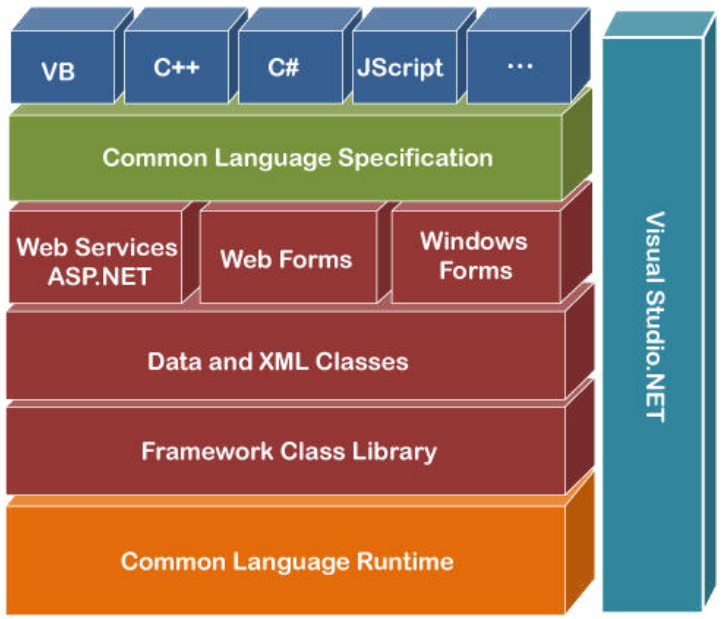
C# derleyicisi, kaynak kodu bir ara dile (IL) çevirir. Program çalıştırılacağında ise IL makina koduna dönüştürülür. Bunu kullanıcı fark etmez. Teorik olarak, bu iki adımlı işlem, aynı IL kodunun farklı makinalarda da çalışabilmesine olanak tanır. Ayrıca IL formundaki bir programda bulunan kötü ve zararlı kod işletim sistemi tarafından kolaylıkla tetkik edilebilir.

.net Framework

.NET FRAMEWORK

Bir çok uygulama geliştirme dili var. Bu dillerle yazılmış uygulamaları çalıştırmak için ise farklı platformlar var. .Net Framework, Windows tabanlı uygulamalar geliştirebilmek için

farklı programlama dillerinin birlikte çalışabileceği ortak bir geliştirme ortamıdır. Bu ortam hem yazılımları geliştirmek hem de çalıştırmak için gereklidir.



Şekil. .NET Framework genel diyagram

.NET, Microsoft tarafından geliştirilen, farklı türde uygulama oluşturmayı sağlayan, ücretsiz ve açık kaynaklı bir yazılım geliştirme platformudur.

Her ne kadar .NET'ten framework olarak bahsedilse de .NET aslında bir yazılım geliştirme platformudur. Bir programlama dili değildir, programlama dili ile çeşitli uygulamalar geliştirmeyi sağlayan bir ortamdır.

.NET sistemi birden fazla programlama dilini, kütüphaneyi ve kitaplığı içerir. İçerisinde yer alan kütüphaneler sayesinde web, mobil, masaüstü, IoT uygulamaları ve daha fazlası kolaylıkla geliştirilir. .NET kütüphanesinin ana dili C# olarak gösterilmektedir. Ancak sadece C# kullanılmaz, pek çok farklı programlama dilini de desteklenir.

Stack Overflow 2022 anketine göre .NET en popüler çerçevelerin başında gelmektedir. Aktif bir geliştirici topluluğunun olması onun önemli özelliklerinden bir tanesidir. Bu topluluk .NET platformunu destekler ve bakımını yapar.

.NET modüler bir mimariye sahiptir. .Net mimarisi 3 temel katmandan oluşur. Bunlar dil, kütüphane ve çalışma zamanıdır. Geliştiriciler, .NET uygulamalarını oluşturmak için .NET programlama dillerini ve uygulama modellerini kullanırlar. .NET çalışma zamanı daha sonra bunları çalıştırır.

.Net ile Neler Yapılabilir?

.NET ile farklı uygulamalar geliştirmek mümkündür. .NET ne işe yarar sorusuna verilebilecek pek çok cevap vardır. Bu uygulamaları .NET Framework'ü kullanarak geliştirmek isteyenler, Microsoft'un web sitesinden güncel sürümünü indirebilir ve kullanmaya başlayabilirler. Ancak Microsoft programları içinde .Net Framework hali hazırda bilgisayarda kurulur olarak yer alır. İstenildiği zaman aktif edilebilir. .NET kullanarak şu uygulamaları geliştirilebilir:

Windows Uygulamaları,

Windows Phone Uygulamaları,

Web Uygulamaları (ASP.Net),

Oyunlar

Mobil Uygulamalar

Windows Azure ile cloud uygulamaları,

MS Office için eklentiler,

Veri tabanı uygulamaları

.NET'in Avantajları Nelerdir?

Geliştirme kolaylığı: Geliştiriciler, .NET kullanarak daha hızlı ve verimli bir şekilde geliştirme sürecini yönetebilirler.

Üretkenlik ve zaman tasarrufu: .NET'in çerçevesi ve ortak kitaplığı geliştiricilerin üretkenliğini artırmaya yardımcı olur. .NET zamandan tasarruf etmeye yardımcı olur.

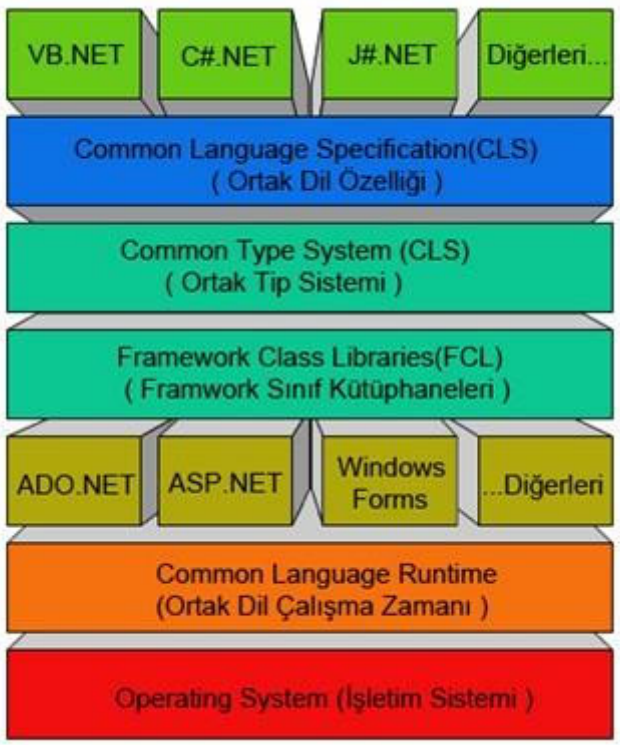
Yüksek performanslı uygulamalar oluşturma: .NET, daha hızlı yanıt süreleri sağlaması ve daha az bilgi işlem gücü gerektirmesi sonucunda yüksek performanslı uygulamalar geliştirmeye yardımcı olur.

Güvenilirlik: .NET kullanılarak tasarlanan uygulamalar güvenilir ve yüksek performanslıdır. Microsoft, resmi olarak .NET'i destekler ve bu da .NET'i güvenli kılar.

Zengin kaynaklar ve kitaplıklar: .NET, geliştiricilerin uygulamaları daha hızlı geliştirmesine yardımcı olmak için çok çeşitli kitaplıklar, araçlar ve yöntemler içerir. Bu .NET'in önemli avantajlarından bir tanesidir.

Kolay bakım: .NET içeren sayfaların yazılması ve bakımı son derece basittir.

Topluluk desteği: .NET'i herkes özgürce kullanabilir, okuyabilir ve değiştirebilir. Açık kaynaklıdır. Aktif ve çok geniş .NET Foundation adında bir geliştirici topluluğu vardır. Bu topluluk, .NET yazılımının gelişmesini sağlar. Amacı, .NET topluluğunu etkinlikler ve kaynaklarla desteklemektir.



Şekil: .Net Framework yapısı

. Net Framework aşağıdaki bileşenlerden oluşur

- **The Common Language Runtime (CLR)** *Ortak Dil Çalışma Zamanı*
  Uygulamaların çalışabilmesi ve yönetilmesine yardımcı olan, dilden bağımsız bir yazılım geliştirme ve çalışma ortamdır.

- **The Framework Class Libraries (FCL)** *Framework sınıf kütüphaneleri*
  .Net çatısı altındaki dillerin ortak kullanabilecekleri kütüphaneleri barındırır.

. Net ortamında yazılımlarınızı dilden bağımsız olarak geliştirebilirsiniz. Örneğin C# 'ta yazılmış kod derlendiğinde, derleyiciden çıkan kod önce **MSIL ya da IL'e**dönüştürülür. Bu MSIL ya da IL dosyası çalıştırılabilir bir kod değildir.

MSIL ya da IL: İçerisinde değişken tanımları, değişkenlerin nasıl saklanacağı, metotların nasıl çalıştırılacağı, aritmetik ve mantıksal işlemler, bellek kullanımı v.s. gibi işlerin nasıl yapılacağını içeren özel tipte kodlar barındıran bir dosyadır.

CLR: Program çalıştırıldığında aradile(MSIL ya da IL) dönüştürülen kodların, çalıştırılabilir koda dönüştürme işini CLR yapar. CLR kodu alıp bulunduğu makinanın işlemcisine ve işletim sisteminin anlayacağı şekle (makine diline) dönüştürür. IL kodu makine diline dönüştürülürken

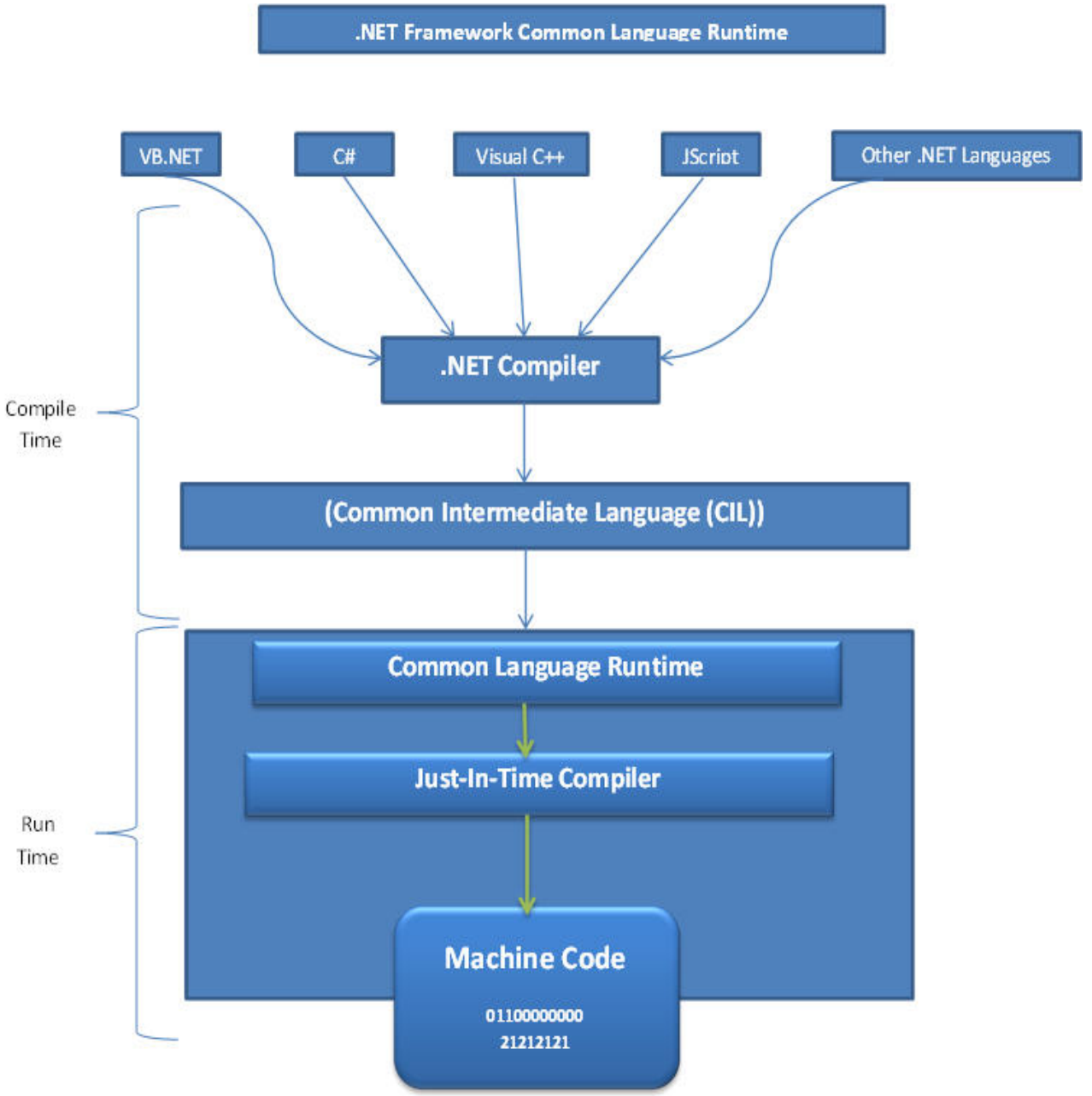
JIT (just in time) derleyiciler devreye girer. CLR makine diline çevrilmiş kodu önbellekte tutarak, daha hızlı çalışmasını da sağlar.

MSIL ya da IL'e dönüştürülen her kod CLR'nin uygulandığı her ortamda çalıştırılabilir. Yani bu da kodun .Net Framework'ün kurulabileceği her platformda çalıştırılabileceği anlamına gelir.

2.3. C# İle .Net Framework Arasındaki İlişki

.net framework ve geliştirme ortamları C# dili ile geliştirilmiştir. C#, .net ile birlikte doğmuştur, modern nesne yönelimli dillerin bütün özelliklerini barındırmakla birlikte nesne yönelimli geliştirmeye yeni yaklaşımlar da getirmektedir.

Bir .exe Uygulamasının Anatomisi (CLR, CIL, JIT, CTS Kavramları)



Bir exe uygulamasını Compile Time ve Run Time'da çalıştıran bileşenler

Bu kavramları daha iyi idrak edebilmek adına iki kavramın üstünden hızlıca geçmk faydalı olacaktır. Diyagramda yer alan her bir bileşen incelendiğinde,

Compile Time ve Run Time Nedir?

Compile Time (Derleme Zamanı)

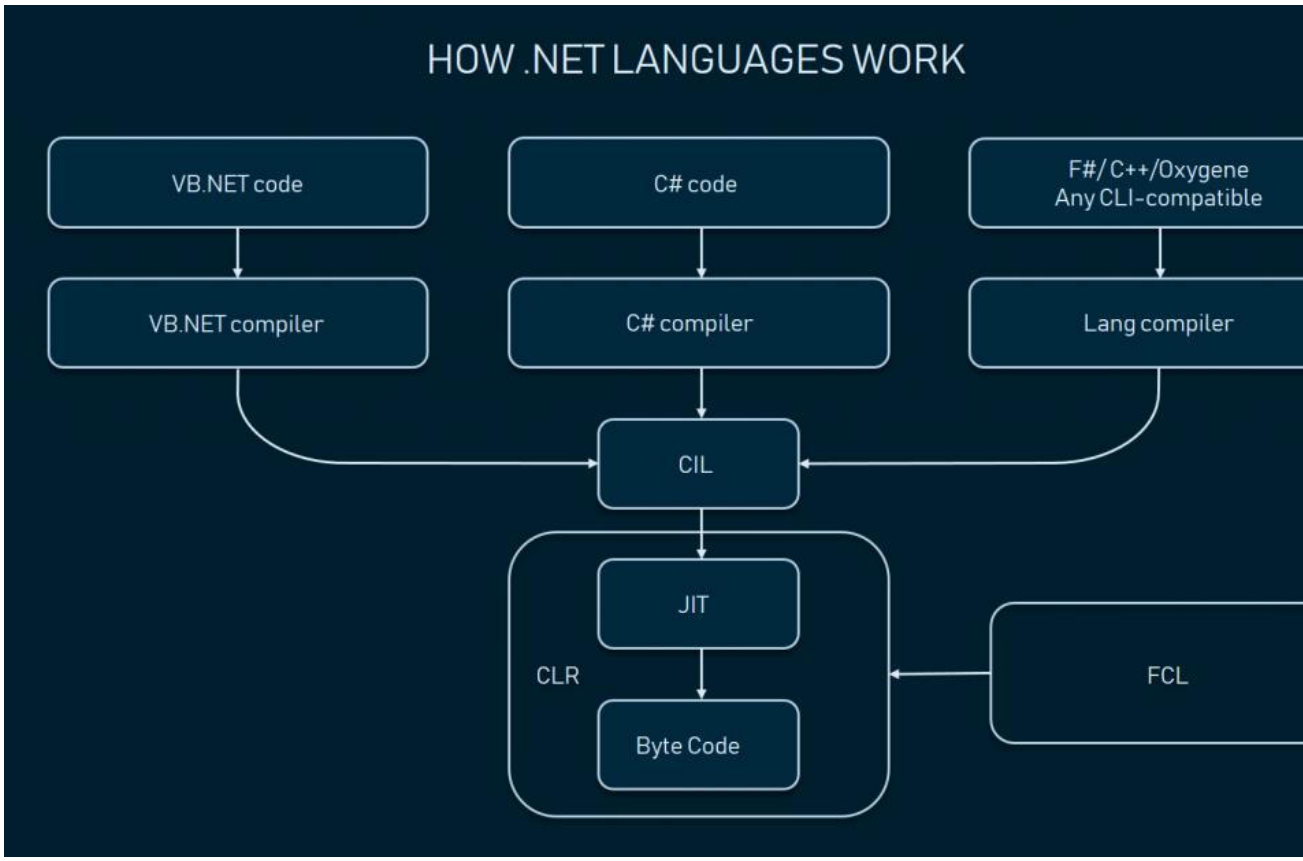
Basitçe compile işlemi yüksek seviyeli bir dilin daha düşük seviyeli bir dile dönüştürülmesi işlemidir

Compile işlemi; oldukça basit ve akılda kalıcı bir izahla "**yüksek seviyeli bir dilin daha düşük seviyeli bir dile dönüştürülmesi işlemidir**". Yani biz yazılım geliştiriciler tarafından yazılmış kod, compile işleminden sonra makinelerin seviyesine yaklaşmaktadır. İşte bu işlemi gerçekleştiren mekanizmaya **Compiler(Derleyici)** derken, bu süreçte yaşanan tüm işlemleri zaman açısından anlatmak için de **Compile Time (Derleme Zamanı)** kavramını kullanmaktayız. Yukarıdaki compiler diyagramında Compilation Error kısmı ise, yüksek seviyeli bir dilin compile edilme işlemi sırasında oluşan hataları anlatmaktadır. Compile Time hatalarının en güzel tarafı .exe dosyası daha oluşturulmadan derleme işlemi sırasında hata alınmasıdır. '*Syntax Error (Sözdizimi Hatası)'* hatasını Compilation Error'a örnek olarak verebiliriz.

Run Time (Çalışma Zamanı)

Programın çalıştırıldığı andan sonlandırılıncaya kadar geçen süreci anlatmak için kullanılan kavramdır. Compile Time'da olduğu gibi Run Time'da da hatalar almak pek mümkündür ki aslında kıyaslandığında en acı senaryo Run Time errorlardır. Bazen bir run time hatasını bulmak için çok fazla efor sarfetmek gerekebilir. Basitçe yine neleri Run Time Error'a örnek olarak verebiliriz diye düşündüğümüzde aşağıdaki örnek maddeler en azından bu kavramı daha iyi idrak etmeye yardımcı olacaktır;

- Bellek yetersiz hatası
- Programda bir dosya için işlem yapacağımızı düşünelim ve bu dosyanın yolunu kod içerisinden sabit bir şekilde verdiğimizi düşünelim. Kod burada compile işlemini başarıyla geçecektir ve program çalıştırıldığında eğer kodda belirttiğimiz dosya yolunda ilgili dosya bulunamazsa hata oluşacaktır. Bu hata örneği compilation error ve run time error arasındaki farkı güzel bir şekilde idrak etmeye yardımcı olacaktır.



Compile Time ve Run Time kavramlarına da neşter vurarak damarlarında gezindiğimize göre ana konumuza tekrar geri dönelim. Bir .exe uzantılı programın .NET Framework'te çalışması için yazdığımız kod öncelikle compile işlemiyle daha düşük seviyeli bir dile çevriliyor. Yukarıdaki görselde de görüleceği üzere VB.NET ile yazılmış bir kod .NET Framework içerisinde yer alan VB.NET compiler ile, C# ile yazılmış bir kod C# Compiler ile ve F#/C++ gibi dillerle yazılmış kodlar da yine .NET Framework içerisinde yer alan o dile özgü compilerlar ile compile edilmektedir. Bu kısımdan itibaren önemli olan şey tüm bu farklı dillerde yazılan kodlar compilation

sonrası **CIL (Common Intermediate Language)** denilen bir ara dile çevrilmektedir.

O halde ana diyagramımızdan nerede olduğumuza geri dönüp bir bakacak olursak şu anda yazdığımız kodu compile ettik ve CIL (Common Intermediate Language) oluştu. Derleme işlemi sırasında bir hata oluşmadıysa artık Compile Time'ı sorunsuz bir şekilde aşmış bulunuyoruz. Şimdi CIL (Common Intermediate Language) nedir, biraz da bu kavramı inceleyelim.

CIL (Common Intermediate Language) Nedir?

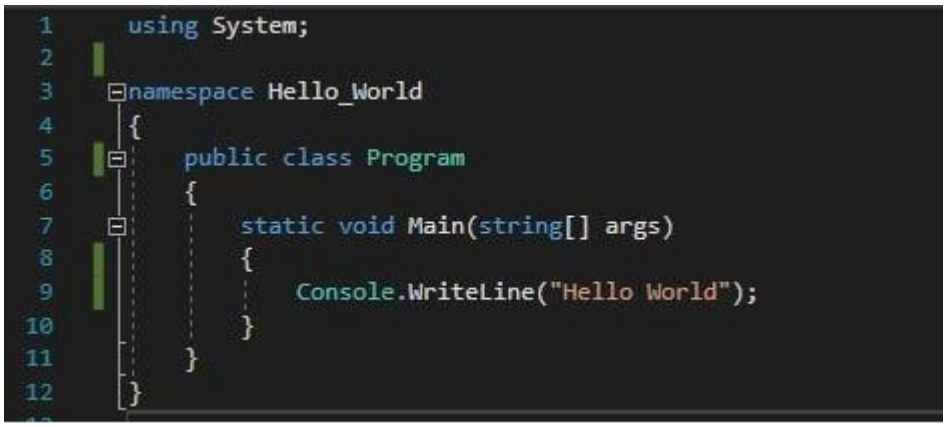
CIL veya .NET Framework için MSIL olarak adlandırılır.
Türkçe'ye **Ortak Ara Dil** olarak çevrilmektedir ve ara dil olarak nitelendirilmesinin sebebi; **C# kadar yüksek seviyeli yani bir insanın rahatça anlayabileceği kadar kolay bir dil olmamasına karşın, makine dili kadar da insandan uzak bir dil değildir**.

Compilation işlemi sonrası CIL'e dönüştürülen kodlarımız exe dosyası olarak kaydedilir. Bu aşamadan itibaren artık çalıştırılabilir bir executable file'ımız mevcuttur. Burada dikkat edersek .exe içerisinde direkt olarak makine kodları yer almamaktadır, yazdığımız koddan CIL'e dönüştürülmüş kodlar yer almaktadır. Bu sebeple exe dosyasının çalışması için makinede muhakkak .NET Framework'ün yer alması gerekmektedir. Çünkü çalıştırma işlemini .NET Framework içerisindeki CLR mekanizması gerçekleştirecektir.

Not: Bir exe dosyasını compile edildikten sonra direkt olarak başka birisine verdiğimizde aslında bir bakıma projenin kaynak kodlarını da vermiş oluyoruz. Decompiler'lar aracılığı ile eğer herhangi bir şifreleme algoritması veya güvenlik aracı ile exe'mizi şifrelemezsek kolayca kaynak koduna erişilebileceğini unutmayalım.

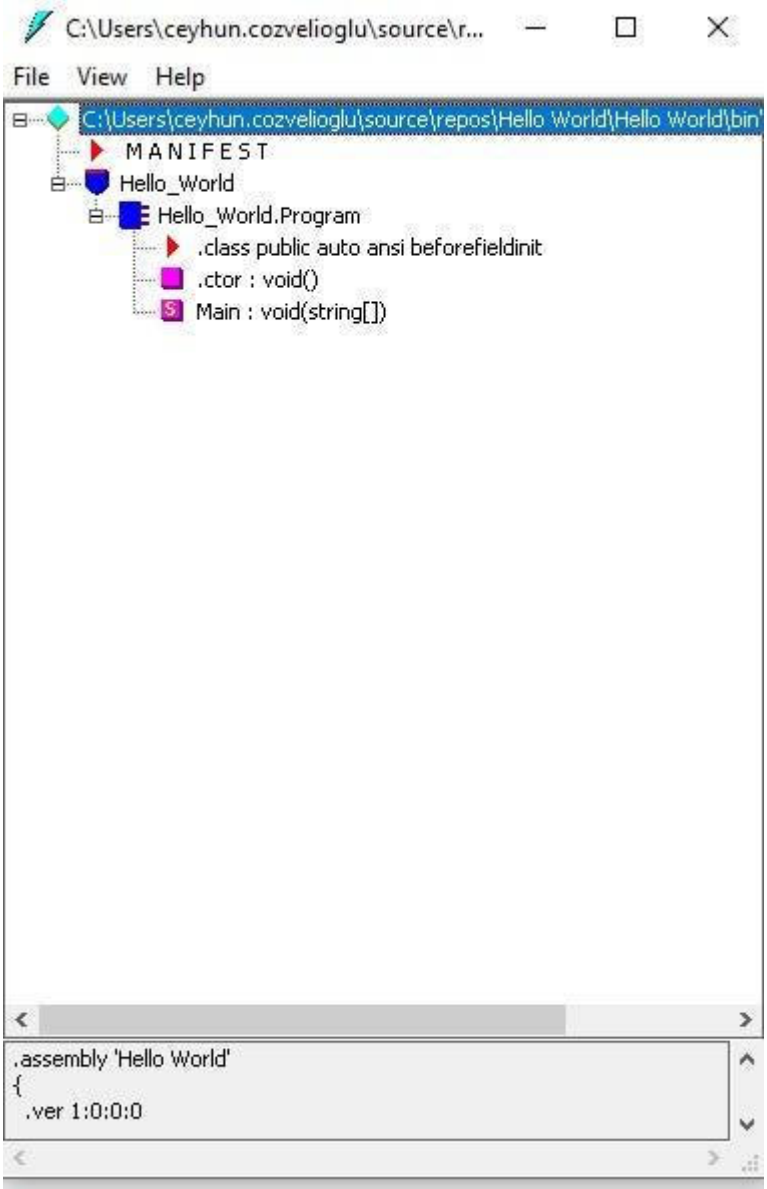
CIL içerisinde Metadata olarak adlandırılan bir yapı daha vardır. Metadata programda kullanılan veri tiplerini, oluşturduğumuz sınıfları, sınıfların methodlarını, özelliklerini ve diğer bilgilerini de içerir. Yani CIL'e biraz daha üstten genel bir bakış atarsak, CIL içerisinde değişken tanımları, değişkenlerin nasıl saklanacağı, methodların nasıl çalıştırılacağı, aritmetik ve lojik işlemler, bellek kullanımı gibi birçok işin nasıl yapılacağı açıklanır.

Sihir gibi değil mi? :) Bir "Hello World" uygulamasının CIL kodunu inceleyelim dilerseniz. Visual Studio ile basit bir Hello World console application projesi oluşturup, konsola "Hello World" yazdıran kodu yazalım.

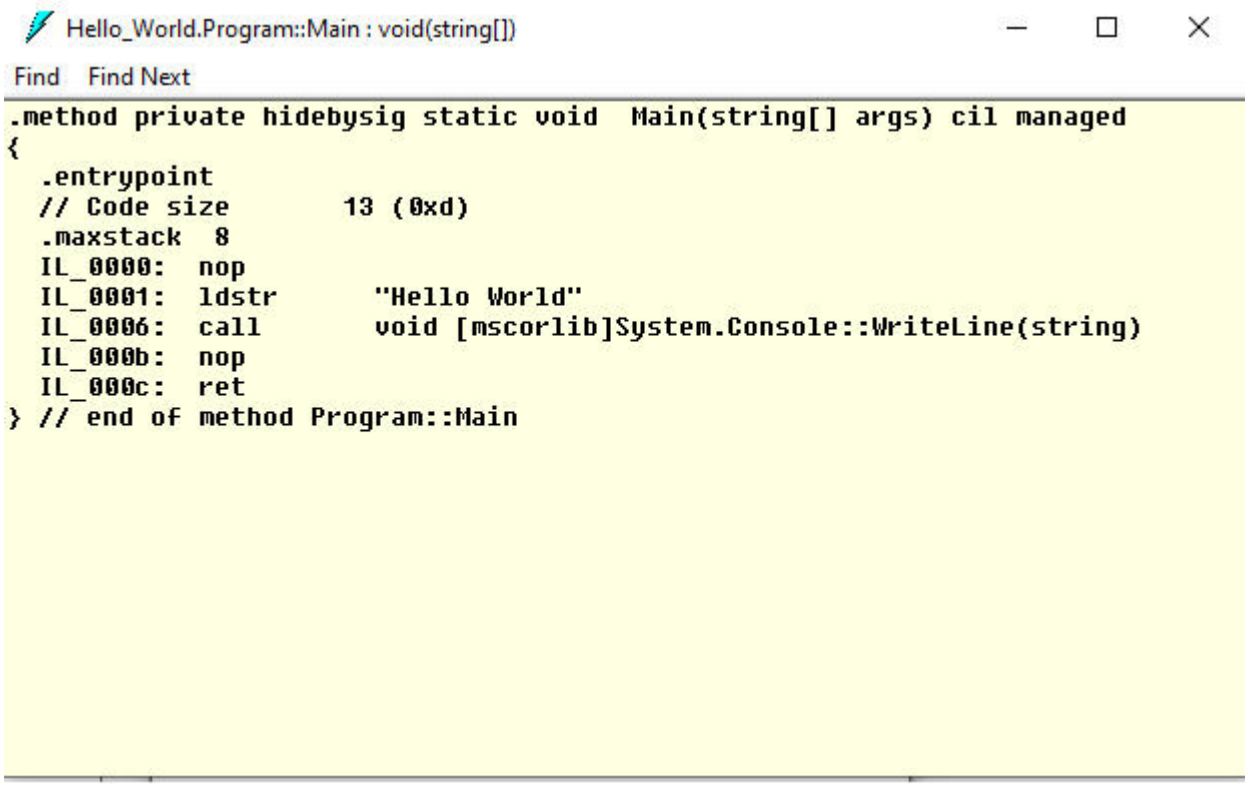


```
using System;

namespace Hello_World
{
    public class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
        }
    }
}
```

Kodu derledikten sonra, CIL kodunu görüntüleyebilmek için **IL DASM** tool'unu kullanabiliriz. Bu toola'a Developer Command Prompt'a **ildasm** yazarak kolay bir şekilde ulaşabiliyoruz. IL DASM ile exe dosyasını açtığımızda ilk olarak aşağıdaki görüntü bizi karşılamaktadır.



Burada programın manifesto bilgisine, methodlarına ve classlarının özelliklerine erişebiliyoruz. Gelin şimdi bir de Main methodunun CIL kodunun nasıl olduğunu görüntüleyelim.



İşte sihirli kısım. CIL kodunu anlamak aslında çok da zor değil. Örneğin; **ldstr "Hello World"** satırında 'Hello World' stringinin load edileceği bilgisi yer alıyor. Hemen bir altındaki call çağrısıyla hangi methodun çağrılacağı belirtilmiş. Görüldüğü gibi, CIL ne makine dili kadar insandan uzak ne de C# kadar insana yakın.

Artık kodumuz derlendi, exe dosyası oluşturuldu ve çalıştırılmaya hazır. Run Time'daki operasyonlara geçerek, exe dosyasının .NET Framework'te nasıl çalıştırıldığını inceleyelim.

CLR (Common Language Runtime) Nedir?

CLR (Common Language Runtime) .NET Framework'te programların çalışmasını kontrol eden ve işletim sistemi ile program arasında yer alan bir arabirimdir. CIL kodlarını çalıştıran ana mekanizmadır. CIL kodları daha önce de incelediğimiz gibi makine dili değil bir ara dildir. Bu sebeple programın çalıştırılabilmesi için bu ara dili alıp makine diline çevirecek ve makinede işletecek bir mekanizmaya ihtiyaç duymaktaydık. Bu ihtiyacı CLR karşılamaktadır.

CLR, içerisinde yer alan **JIT (Just in Time)** derleyicisi ile, CIL kodunu alır makine koduna çevirir ve programın çalışmasını sağlar. Akıllara hemen 'İyi de neden böyle bir şeye ihtiyaç duyuyoruz ki, direkt olarak neden yazdığımız kodlar makine diline çevrilmiyor da ara bir dile dönüştürülerek CLR'a ihtiyaç duyuyoruz?' gibi bir soru gelebilir. CLR'ın bize sağladığı avantaj programlarımızı platform bağımsız çalıştırabilmemizi sağlamasından ileri geliyor. .NET Framework yüklü herhangi bir makinede CIL kodunu CLR işleteceği için kaynak kullanımları, performans ile ilgili maksimum optimizasyonu da yine JIT derleyicisi ile beraber sağlamış oluyor.

Hazır JIT kavramı geçmişken, JIT derleyicisi nedir hemen inceleyelim.

JIT (Just in Time) Compiler Nedir?

C#'ta CIL'e compile ettiğimiz program çalıştırılırken CLR aracılığı ile JIT derleyiciler devreye girer. Bu derleyiciler CIL kodunu işleyerek programın çalıştırıldığı sistemin ve işlemcinin anlayabileceği makine kodunu oluştururlar. Burada bir miktar kafa karışıklığı yaşanmış olabilir. "Bir defa kod yazıyoruz ve iki kere mi derleme işlemi uygulanıyor?" gibi bir soru akıllara gelebilir. En başta yazdığımız bir kod iki kere derleniyor evet. Önce bizim yazdığımız C# kodu C# derleyicisi ile CIL'e dönüştürülüyor ve bu oluşan CIL, CLR mekanizması içerisindeki JIT derleyicileri ile makine koduna çevriliyor. Just in Time denilmesinin sebebi, Türkçe'ye '**O an**' veya '**Tam o anda**' şeklinde çevirerek biraz daha türlerini incelediğimizde daha net ortaya çıkacaktır.

3 çeşit JIT compiler bulunmaktadır. Bunlar

1. Normal-JIT
2. Econo-JIT
3. Pre-JIT

JIT Türleri Karşılaştırma Tablosu

CLR içerisinde yer alan başka bir yapı ise CTS (Common Type System) yapısıdır. Bu yapıyı da inceleyerek bir .NET programının çalışma anatomisini tamamlamış olalım.

Özet

- Yazılan kod, o dile özgü derleyici ile CIL (Common Intermediate Language)'e çevrilir.
- CIL içerisinde Metadata yapısını da barındırır.
- CLR mekanizması aracılığı ile CIL kodları JIT derleyicileri ile derlenerek program çalıştırılır.

C# İle Program Yazmak

C# ile program yazmak için bilgisayarınızda .NET Framework SDK kurulu olması yeterlidir. .NET Framework SDK Microsoft un web sitesinden indirebilirsiniz.

C# editörü olarak genellikle Microsoft a ait Visual Studio kullanılır. Visual Studio krulduğu zaman .NET Framework SDK da kurulmuş olur.

Problemden Algoritmaya

Bir problemin en küçük ayrıntılarına kadar bölünerek, çözüm yolunun belli bir mantık düzeni içerisinde ve sınırlı sayıda adımlarla belirlenmesine ***Algoritma*** *denir.* Algoritma tekniği ile çözülebilecek problemlere aşağıdaki örnekler verilebilir:

Bir kravatı bağlamak, iki tabi sayının çarpılması, otomatik bilet makinesinin kullanılması, bujilerin değiştirilmesi, v.d.

Algoritma, aşağıdaki iki kısmı içermektedir:

* Nesne (Obje) : Nesneye bir hareket yapılacaktır.
* İş (Olay) : İş, belirli bir sırada nesneye yapılan faaliyetler.

Örnek: Bir aracın lastiğinin değiştirilmesi aşağıdaki gibi tarif edilebilir:

1. Bijonların gevşetilmesi
2. Aracın kaldırılması,
3. Bijonların çıkarılması,
4. Lastiğin değiştirilmesi,
5. Bijonların takılması,
6. Aracın indirilmesi,
7. Bijonların sıkıştırılması.

Bu örnekte aşağıdakileri gözlemek mümkündür:

a) Bu algoritmayı uygulamak isteyen kişinin, her bir adımı anladığı ve uygulayabildiği beklenmektedir. Bu durumda sonuç kesinlikle bellidir. Aksi takdirde işlem adımları daha detaylı bir hale getirilmelidir. Bir algoritmanın kullanılabilmesi uygulayan kişinin kaabiliyetine bağlıdır.

b) Verilen Yol (usul- yöntem), belirli sırada uygulanması gereken bir dizi aksiyonları içermektedir. Aksiyonların işlem sırasına algoritmanın uygulandığı sırada da karar vermek mümkündür.

c) Algoritmadaki bazı aksiyonların işlem sırası önemsizdir. Yukarıdaki örnekte örneğin hangi bijonların ilk önce gevşetileceği önemsizdir.

d) Kimin algoritmayı uygulayacağı önemli değildir. Önemli olan uygulayacak kişinin ya da makinanın basit temel kaabiliyetlere sahip olmasıdır.

Her bir aksiyonun makine –bilgisayar- tarafından gerçekleştirilmesi beklenmektedir. Bu durumda yukarıdaki algoritmanın yeniden düzenlenmesini gerektirmektedir:

* Her bir aksiyonun bilgisayarın işleyebileceği forma dönüştürülmesi gerekecektir.
* Katılan nesnelerin de bilgisayarın işleyebileceği forma dönüştürülmesi gerekecektir. Bu iş ise veri tipleri ve veri yapıları kanalıyla gerçekleştirilmektedir.

1.1. Problem – Algoritma - Program

Bilgisayar bilimlerinin çözmesi gereken en temel ödevi; bir problemden hareket ederek problemin çözümüne ulaşmaktır. Tipik bir çözüm yolu aşağıda tanıtılmaktadır:

1. ***Problemin Analizi (Detaylı gösterimi)*** *ve dokümantasyon :* Bu aşamada problem tanınır, ayrıntılarıyla incelenir. İstenenler tesbit edilmeye çalışılır. Giriş, çıkış bilgileri ve aralarındaki ilişki tesbit edilir.

2. ***Çözüm Yolunun Bulunması (Algoritma Geliştirme***): Bu adımın yerine getirilebilmesi için problemin çok iyi anlaşılmış olması gerekmektedir. Problem en küçük parçalarına ayrılarak, her parçanın yapacağı işlevler tesbit edilir. Sonra tüm program adım adım şekillendirilerek planlı ve mantıksal bir taslak program yapılır. Algoritmanın geliştirilebilmesi için çeşitli prensip ve tekniklerden faydalanılır. Sonuçta formal olarak gösterilen bir algoritma ortaya çıkar.

   Aşağıdaki noktalar bu aşamada mutlaka yapılmalıdır:

   * Programın adının ve neticesinin tarifi
   * Verilerin organize ediliş şekli
   * Verilerin giriş, işleniş ve çıkışının tarifi
   * Diğer programlar için gerekli olan donanım ve ara birimlerin listesi
   * Programda kullanılan formüllerin kısa bir özeti
   - Akış Diyagramı

3. ***Algoritmanın Programa Dönüştürülmesi (Kodlama***): Algoritma, seçilen dilin kurallarına göre yazılır. Neticede seçilen dilde yazılmış bir program ortaya çıkar. Arayüz oluşturma.

4. ***Programın Bilgisayarda Çalıştırılması***: Bu adım, küçük adımlardan oluşan makro bir adımdır. Bu aşamada;
* Programın bir editörde yazılması,
* Programlama dilinin kurallarına göre yazılıp yazılmadığının denetlenmesi,
* Programın derlenmesi,
* Programın test edilmesi,
* Ve nihayet girilen veriler için gerçek çözümün bulunup bulunmadığının tesbit edilmesi.

Test ve hataların düzeltilmesi

5. Bakım ve güncelleme

1.2. Problemin Analizi

Belirtilen problem için bir algoritma geliştirmeden önce; problemin detaylı olarak analiz edilmesi gerekmektedir. Problemin analiz edilmesini bir örnekle açıklayalım.

Örnek: Personel bilgilerini içeren bir dizi kayıtların, personelin soyadına göre sıralanması.

Problemin tanımı çok belirsiz ve yanlış anlamalara çok müsait. Bu nedenle ilk önce biri takım soruların cevaplandırılması gerekmektedir:

* Kayıtlar hangi bilgileri içermelidir? (Ör: Adı, Soyadı, Yaşı, Mesleği, v.b.)
* Kayıtlar hangi formda düzenlenmeli? Hangi veri yapısı ile oluşturulmalı (Ör: Listeler ya da Veri kayıtları gibi)
* Hangi değerler kullanılabilir? (Ör: Soyadın büyük harfle başlayıp küçük harflerle devam etmesi gibi)
* Soyadlar nasıl karşılaştırılmalı? (Ör: Alfabedeki diziliş sırasına göre ...)
* Sıralama nasıl olmalı? (Azalan ya da artan sıraya göre)
* Aynı soyada sahip personel de sıralama nasıl yapılmalı? (Ör: Soyada ilave olarak yaş ya da başka bir bilgi de sıralama kiriteri olarak eklenebilir)
* Sonuçlar hangi formda ve hangi sıralamada çıkarılmalı?
* Algoritma da hangi temel operasyonlar kullanılabilir? (Kontrol yapıları, fonksiyonlar)

  Problemin analizinde işlevsel olarak ayrıştırmaya gidilmelidir. Bunlar:
* Giriş verilerinin doğru olarak tespit edilmesi (Verilerin türü, kapsamı, verinin şekli ve veriler arasındaki kısıtlamalar)
* Çıkış bilgilerinin doğru olarak tespit edilmesi
* Çerçeve koşullarının tespit edilmesi (Algoritmanın formüle edilmesinde hangi temel operasyonların ve akış yapılarının kullanılabileceğinin tespit edilmesi)

Örnek: İki kişi birbirilerinden uzak bir mesafeden karşılıklı olarak birbirlerine doğru karşlılaşıncaya kadar sabit hızda yürümeye başlıyorlar. Kişilerden daha hızlı koşan bir köpek birinci kişi ile birlikte ikinci kişiye doğru yine sabit hızda koşmaya başlıyor. İkinci kişinin yanına varınca dönüyor tekrar birinci kişiye doğru koşmaya devam ediyor. Böylece iki kişi yüz yüze gelene kadar köpek kişiler arasında gidip geliyor.

Problemi analiz ederek, köpeğin aldığı yolu bulan bir algoritma geliştiriniz.

İlk soru; başlangıçta hangi verilerin belli olduğunun tespitidir. Biz, iki kişi arasındaki mesafenin, kişilerin hızlarının ve köpeğin hızının önceden bilindiğini varsayıyoruz. Öyleyse aşağıdaki tespiti yapabiliriz:

Giriş Verileri:

- Uzaklık d (km), d >0;
- Kişilerin hızları V_1 ve V_2 (km/h) , V_1 >0 ve V_2>0
- Köpeğin hızı V_K, $V_K > V_1$ ve $V_K > V_2$

Çıkış Bilgileri: Köpeğin kat ettiği yol

Bu problemi çözebilmek için birçok çözüm yolu bulunabilir. Fakat en basiti; Karşılıklı yürümeye başlayan kişilerin buluşuncaya kadar geçen süreyi bulmak olacaktır. Bu ise $d / (V_1 + V_2)$ formülü ile kolayca hesaplanabilir. Artık köpeğin aynı süre içerisinde aldığı yolu bulmak çok kolay olacaktır. Yani;

$$D_k = V_k * d / (V_1 + V_2)$$

Çerçeve koşulları olarak da pascal programlama dili yapısı geçerli olacaktır.

1.3. Algoritmanın Gösterimi

Algoritmaların gösterimi için uygun bir dile ihtiyaç vardır. Konuştuğumuz dil bu iş için pek uygun görülmemektedir. Çünkü yanlış anlaşılmalara çık müsait olmaktadır. Bu nedenle algoritmaların gösterimi için yarı formal, grafiksel ve metinsel olarak değişik gösterim şekilleri önerilmiştir. Belirli anlamlarla donatılmış olan sembollerle gösterim şekli çok yaygın olarak kullanılmaktadır. Algoritmaların gösteriminde bazı adımlar çok kaba olarak gösterilebilmektedir.

Algoritmanın gösteriminde aşağıdaki hususlar belirtilir:

- Belirtilen durumu (izlenen nesnenin) tarif eden büyüklükler. Bu değişkenler vasıtasıyla tanımlanır,
- İzlenen büyüklüklere uygulanan temel operasyonlar (işlemler),
- Hangi koşullarda ve hangi sırada temel operasyonların yapılacağını tanıtan akış diyagramları ve akış koşulları.

Algoritma yazarken şunlara dikkat etmelisiniz:

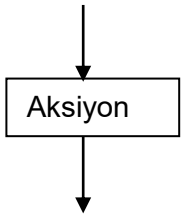
- Probleme uygun bir noktadan başlamalısınız,
- Problemi adım adım düşünmelisiniz,
- Her adımı açık ve net yazmalısınız,
- Her adımda mümkün olduğu kadar az olayı belirtmelisiniz,
- Mutlaka bir sonuca ulaşmalısınız.

1.3.1. Program Akış Şemaları

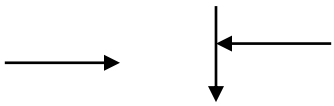
Algoritmaların akışının doğru olarak tarif edilmesi için DIN 66001 normlarına göre standartlaştırılmış, grafiksel olarak gösterilen bir metodudur.

Akış şemalarının gösteriminde kullanılan sembollerin önemlileri aşağıda gösterilmiştir.

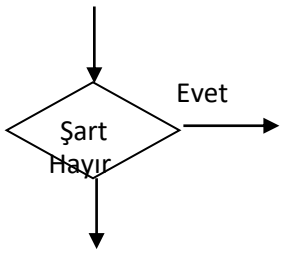
* ***Aksiyonlar*** (Operasyonlar, deyimler, giriş-çıkış bilgileri-aritmetiksel ve mantıksal işlemler.değer atama



* ***Akış / Birleştirme Sembolleri***

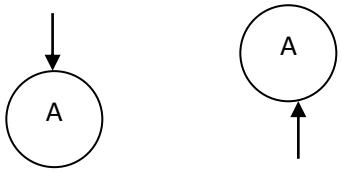


* ***Dallanma Sembolleri (Karar verme)***

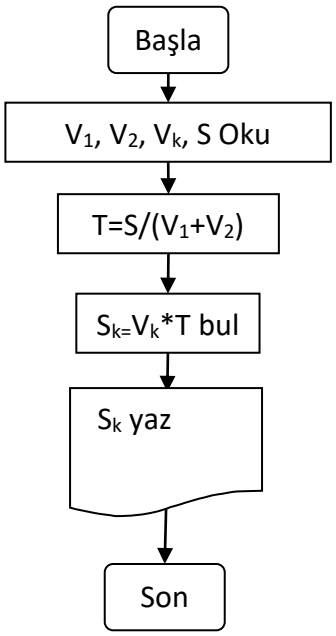


* ***Başlangıç / Bitiş Sembolleri***

* ***Bir diyagramdan başka bir diyagrama geçiş noktası sembolleri***



Örnek: Yukarıda verilen örnekteki köpeğin kat ettiği yolu bulan algoritmayı akış şeması yöntemiyle gösteriniz.
Girilen Veriler: V_1, V_2, V_k ve S
Çıkan Bilgiler: S_k



Açıklama: Birinci (V_1), ikinci kişinin (V_2) ve köpeğin hızını ve kişiler arasındaki mesafeyi (S) sisteme girelim. Daha sonra kişilerin buluşacakları zamana kadar geçen süreyi $T=S/(V_{1+}V_2)$ bulalım. Bu bulunan süre ile köpeğin hızını çarptığımızda ($S_k=V_k*T$) köpeğin aldığı yolu bulmuş oluruz.

Akış Şemalarının Avantajları

* İşlem akışları grafiksel olarak gösterilmektedir. Algoritmanın bu şekilde görselleşmesi yeni başlayanların daha çabuk anlamasına yardımcı olmaktadır.
* Çok karışık bir aksiyon başka bir şemayla (ya da birden çok) detaylandırılabilir.
* Programlama için önemli olan tüm akış yapılarını (sıra, dallanma, döngü) göstermek mümkündür.

Akış Şemalarının Dezavantajları

* Algoritmanın gösterimi çok büyük şemalarda çok çabuk karmaşık bir hale gelebilmektedir. Bu açıdan birçok algoritma için bu şekilde gösterim uygun değildir.
* Dallanma ve birleştirmeler istenildiği kadar birbirleriyle birleştirilebilirler. Buda şemanın yapısal olmaktan çıkmasına neden olmaktadır. Bir anlamda Goto programlamaya benzemektedir.

* Akış Şemasıyla İlgili Örnekler

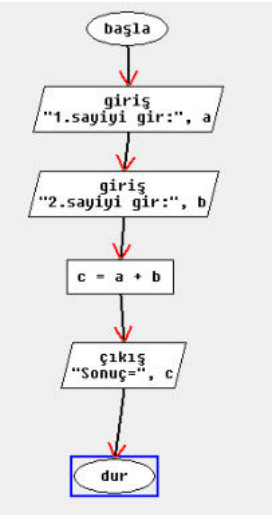
Örnek: Klavyeden 2 sayının toplamını ekrana yazdıran akış diyagramınıyapınız.

Algoritma:

1. Klavyeden 2 sayı al, (a,b)

2. Sayıları topla ve sonucu ata, (c=a+b)

3. sonucu ekrana yazdır.(c)



Örnek: Klavyeden girilen 2 sayının büyük olanını ekrana yazdıran akış diyagramını yapınız.

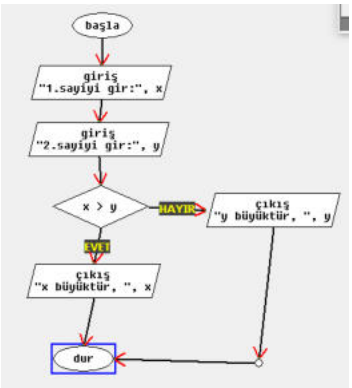
Algoritma:

1. Klavyeden 2 sayı al, (x,y)

2. Sayıları karşılaştır. (x>y)

3. Evetse ekrana x büyüktür yaz.

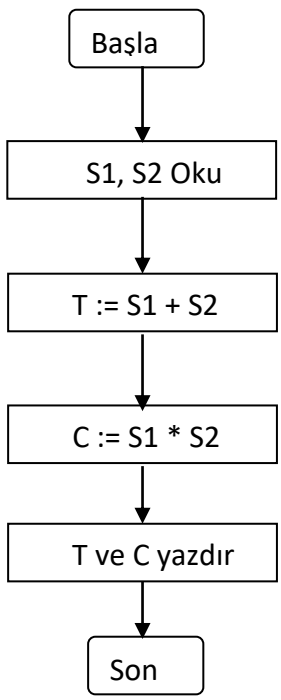
   Hayırsa ekrana y büyüktür yaz.



***Örnek 1*:** Klavyeden girilen iki sayının toplamı ile çarpımını bulan ve sonuçları ekrana yazdıran algoritmayı akış şeması yöntemiyle gösteriniz.

Girilen bilgiler: S1, S2

Çıkan bilgiler : Sayıların toplamı ve çarpımı .

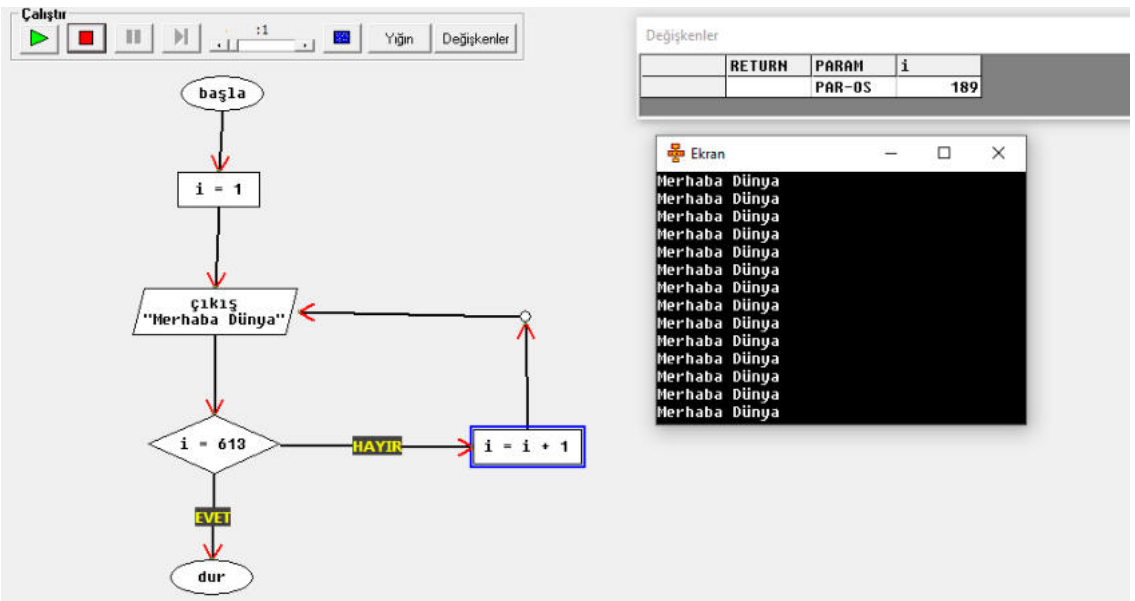


Açıklama: S1 ve S2 adında iki sayı giriyoruz. Bir sonraki adımda T:=S1+S2 komutu ile girilen sayıları topluyoruz. Ekrana yazdırıyoruz. Daha sonrada C:=S1*S2 komutu ile de çarpıyoruz. Ekrana yazdırıyoruz.

C# uygulaması aşağıdadır.

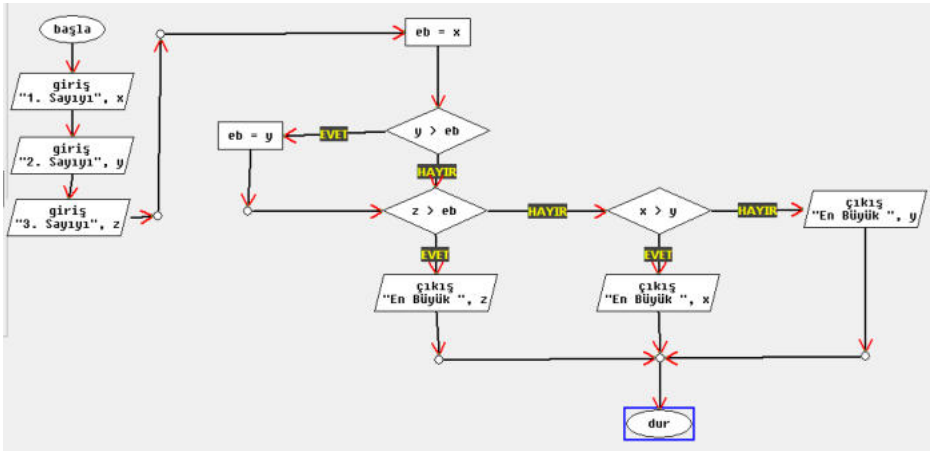
```
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Birinci Sayıyı Gir:");
            int S1 = Convert.ToInt32(Console.ReadLine());

            Console.Write("İkinci Sayıyı Gir:");
            int S2 = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine("TOPLAM:" + (S1 + S2));
            Console.WriteLine("ÇARPIM:" + (S1 * S2));

            Console.ReadLine();
        }
    }
}
```

UYGULAMA: Merhaba Dünya cümlesinin 613 kere ekrana yazdıran Akış diyagramını FC programında oluşturunuz.

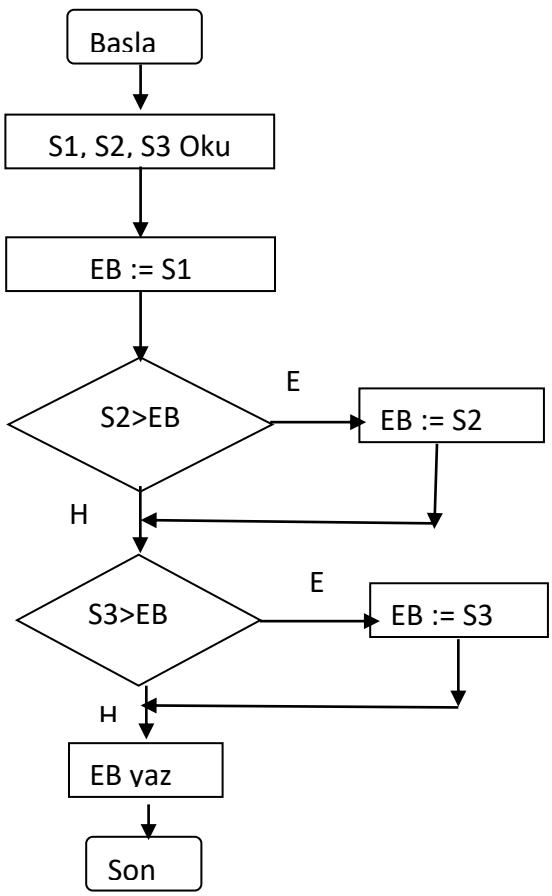


Örnek 2: Okutulan birbirinden farklı 3 sayı içerisinden en büyük sayıyı bulan ve yazan programın algoritmasını akış şeması yöntemiyle gösteriniz.

İ=1,2,3



Girilen veriler : S1, S2, S3
Çıkan bilgiler : En büyük sayı (EB)

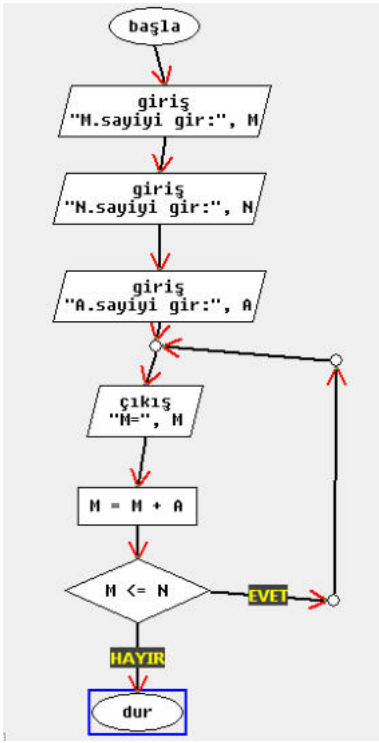


Açıklama: S1, S2 ve S3 adında üç adet sayı giriyoruz. S1 sayısını EB := S1 komutu ile en büyük kabul ediyoruz. Daha sonra S2>EB ise bu durumda EB := S2 komutu ile S2 sayısını en büyük sayı yapıyoruz. S2, EB'den (şu ana kadarki) büyük değil ise bir sonraki adıma geçerek S3>EB komutu ile S3'ün EB'den büyük olup olmadığını soruyoruz. Eğer S3 mevcut EB'den büyükse S3'ü en büyük yapıyoruz. Daha sonra EB'yi yazdırıyor ve programı sonlandırıyoruz.

Programın C# kodu aşağıdadır:

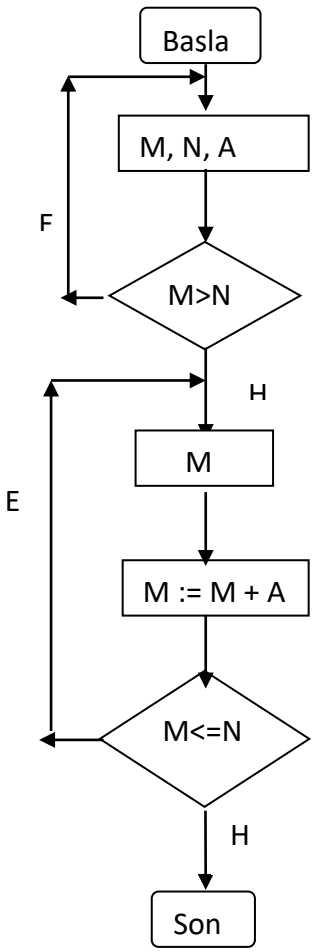
```
using System;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Birinci Sayıyı Gir:");
            int S1 =Convert.ToInt32(Console.ReadLine());

            Console.Write("İkinci Sayıyı Gir:");
            int S2 = Convert.ToInt32(Console.ReadLine());

            Console.Write("Üçüncü Sayıyı Gir:");
            int S3 = Convert.ToInt32(Console.ReadLine());

            int EB = S1;
            if (S2>EB)
                EB=S2;
            if (S3>EB)
                EB=S3;

            Console.WriteLine("En Büyük Sayı:"+EB);
            Console.ReadLine();
        }
    }
}
```

***Örnek** 3*: Bilgisayara girilen M sayısından N sayısına kadar, her seferinde A kadar artan sayıları ekrana yazdıran programın algoritmasını akış şeması tekniği ile gösteriniz.



N=20;
A=5;
N=N-1;
For(M=0;N<M;M=M+A)
{

Döngünün içi
}
Girilen Veriler: M, N, A
Çıkan Bilgiler: M'den N'ye kadar artış miktarı oranında artırılan sayı dizisi.



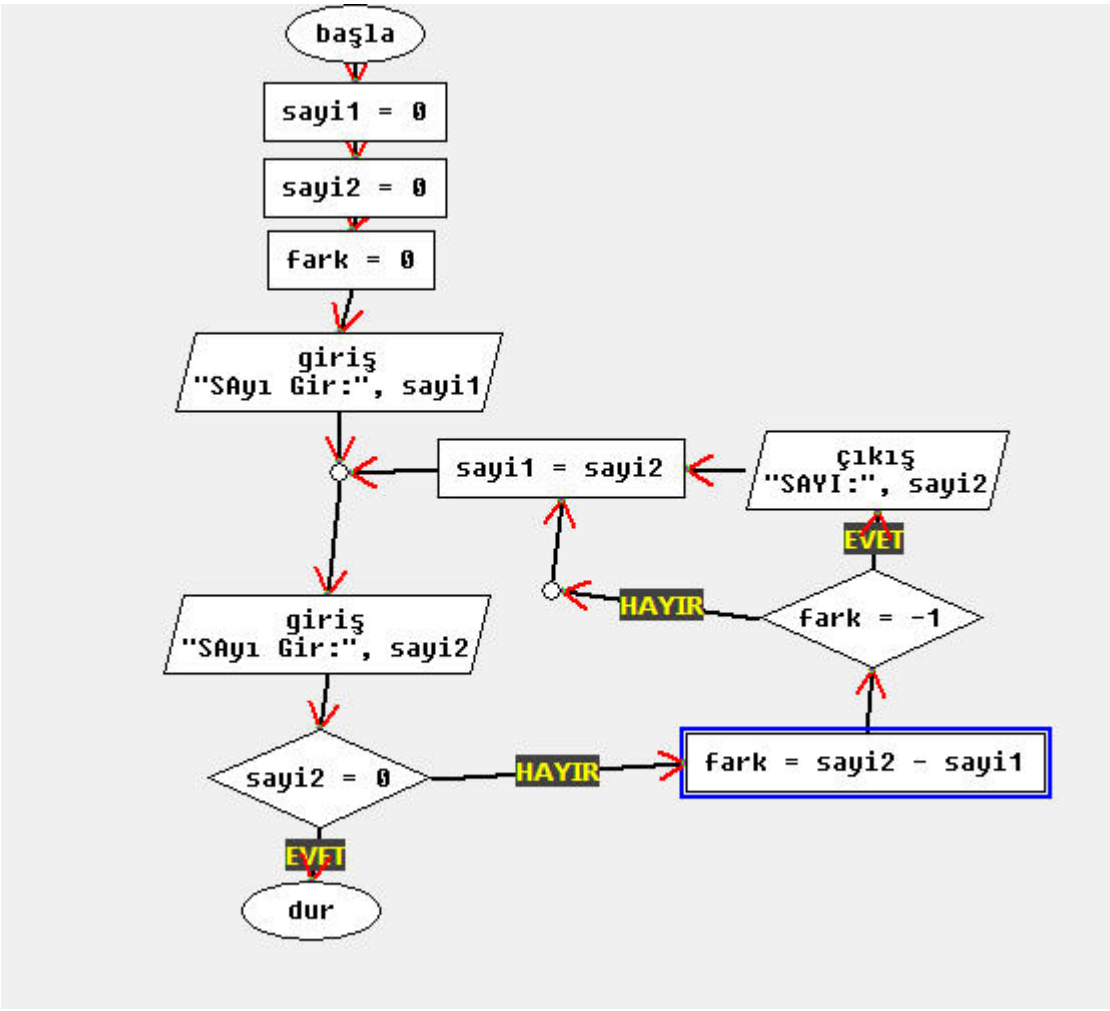
Açıklama: M, N ve A sayıları giriliyor. M>N ise tekrar başa dönülerek yeni sayıların girilmesi isteniyor. Bunun sebebi M sayısının N sayısından küçük olması gereğidir. Yani M sayısı N sayısından küçük ise başlangıç olarak M sayısını yazdırarak işe başlıyoruz. Daha sonra M sayısını A kadar artırıp ve soruyoruz, M hala N den küçük veya eşit ise tekrar yukarıya giderek M sayısını yazdırıyoruz. Bu şekilde döngü devam ediyor. M sayısı N'den büyük olunca program sonlanıyor.

C# kodu aşağıdadır: (goto kullanılması tavsiye edilmez ama başlangıçta kullanabiliriz)

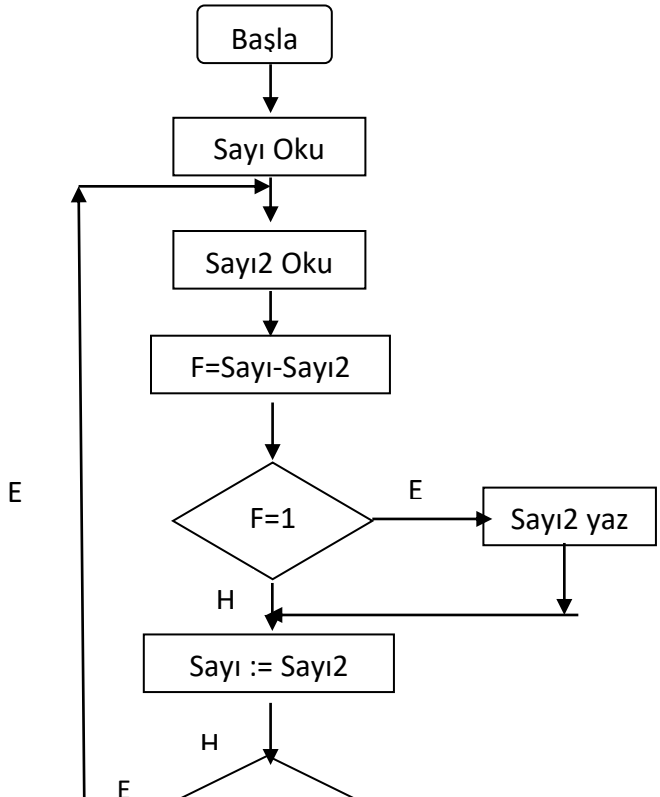
```
using System;

namespace ConsoleApplication3
{
    class Program
```

```
    {
        static void Main(string[] args)
        {
            a:
            Console.Write("Başlangıç Sayısını Gir:");
            int M = Convert.ToInt32(Console.ReadLine());

            Console.Write("Bitiş Sayısını Gir:");
            int N = Convert.ToInt32(Console.ReadLine());

            Console.Write("Artış Miktarını Gir:");
            int A = Convert.ToInt32(Console.ReadLine());

            if (M > N)
            {
                Console.WriteLine("Yanlış değer girdiniz");
                goto a;
            }
            while (M <= N)
            {
                Console.WriteLine("     " + M);
                M = M + A;
            }
            Console.ReadLine();
        }
    }
}
```

***Örnek 4*:** Bilinmeyen miktarda okutulan sayılardan, kendisinden önce gelen sayıdan 1 eksik olan sayıları bulan, sıfır sayısı girildiğinde durarak sonucu ekrana yazdıran programın algoritmasını akış şeması yöntemiyle gösteriniz.



Örnek Sayılar: 4 5 21 9 8 32 0



Açıklama: Birinci sayıyı (4) ve arkasından ikinci sayıyı (5) giriyoruz. Bir sonraki adımda F=Sayı-Sayı2 komutu ile farkı buluyoruz. Örneğe göre fark -1. Fark (F) eğer 1'e eşitse Sayı2'yi yazdırıyoruz değilse bir sonraki adımda Sayı := Sayı2 komutu ile Sayı2'yi Sayı1'e atıyoruz. Yani 2. Sayıyı 1. Sayı yapıyoruz. Daha sonra soruyoruz; girilen sayı 0 dan farkı ise yukarı çıkarak sıradaki sayıyı (sayı2) okuyarak işleme devam ediyoruz.

C# Kodu aşağıdadır:

```
using System;

namespace ConsoleApplication4
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("SayıyıGiriniz:");
            int sayi=Convert.ToInt32(Console.ReadLine());

            while (sayi != 0)
            {
                Console.Write("Sıradaki Sayıyı Gir:");
                int sayi2 = Convert.ToInt32(Console.ReadLine());
                int F = sayi - sayi2;
                if (F == 1)
                    Console.WriteLine(sayi2);
                sayi = sayi2;
            }
            Console.ReadLine();
        }
    }
}
```

Örnek 5: Bilgisayara 0 sayısı girilene kadar, girilen sayıların kendilerinin, kareköklerinin ve karelerinin toplamını ve sayıların toplamının ortalamasını bulup sonucu ekrana yazdıran programın algoritmasını geliştirelim.

Girilen Veriler : S : Sayı dizisi,

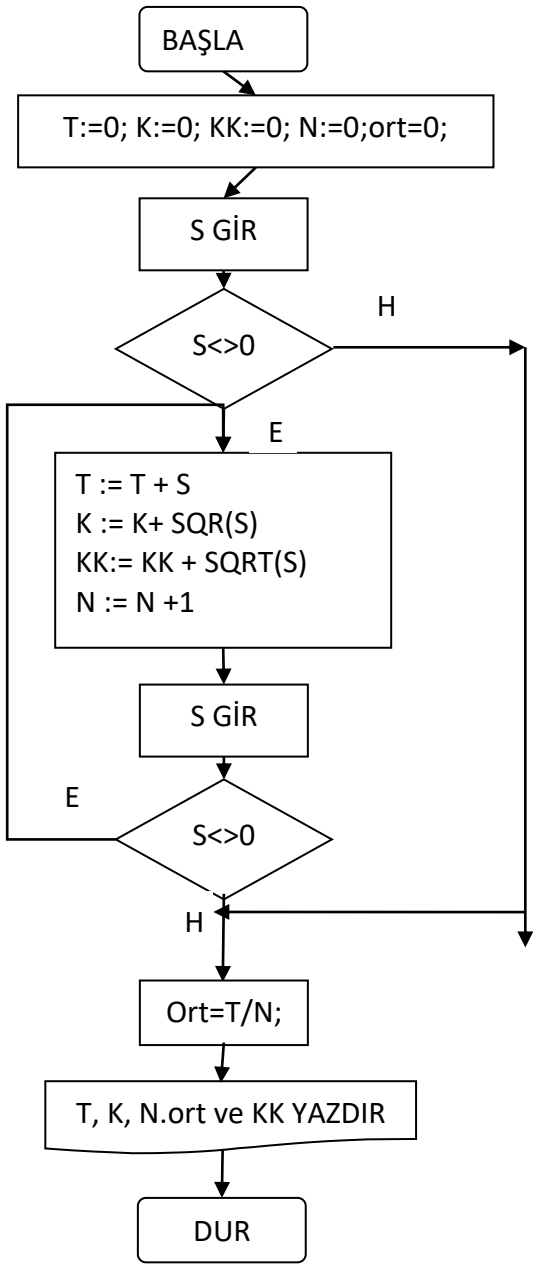
İstenen Bilgiler: T : Sayıların toplamı,

K : Kareler toplamı,

KK :Karekökler toplamı,

Ort : Sayıların ortalaması,

N : Sayıların adedi.



1. *T, K, KK ve N değişkenlerine sıfır değerini ata.*
2. *S gir // Sayıyı gir.*
3. *S <> 0 değilse (yani sayı 0 ise) 9. adıma git.*
4. *T (Toplam) değişkenini S (sayı) kadar artır.*
5. *Sayının (S) karesini al ve K değişkenine ekle.*

6. *Sayının (S) karekökünü al ve KK değişkenine ekle.*
7. *Sıradaki sayıyı (S) gir,*
8. *S<>0 (sayı sıfırdan farklı ise) 4. adıma git.*
9. *T, K, KK ve N değerlerini yazdır.*
10. *Durdur.*

Algoritmanın C# dilinde kodlanmış şekli aşağıdadır:

```
{
            double kk=0,ss;
            int s = 1, k = 0,n=0,t=0,ort=0;
            while (s > 0)
            {
                Console.Write("Sayı Giriniz:");
                s =Convert.ToInt32(Console.ReadLine());
                t = t + s;                                  //sayılar
toplamı
                k = k+s * s;                          //sayının karesi
toplamı
                ss =Math.Sqrt(Convert.ToDouble(s));     //sayının kare
kökü
                kk = kk + ss;                              //karekök ler
toplamı
                n = n + 1;
            }
            ort = t / (n - 1);
            Console.WriteLine("SAYILARIN TOPLAMI:"+t);
            Console.WriteLine("SAYILARIN KARELERİ TOPLAMI:" + k);
            Console.WriteLine("SAYILARIN KAREKÖKLERİ TOPLAMI:" + kk);
            Console.WriteLine("SAYILARIN ORTALAMASI:" + ort);
            Console.ReadLine();

              }
```

Örnek 6: Bilgisayara kişilerin boyu, yaşı, kilosu ve cinsiyeti girilmektedir. Aşağıdaki formüle göre kişilerin ideal kilolarını bularak;

IK=(Boy – 100 + Yas/10)*CK

Eğer ; İdeal kilo = gerçek kilo ‘İDEAL’

İdeal kilo < gerçek kilo ‘ZAYIFLA’

İdeal kilo > gerçek kilo ‘KİLO AL’

mesajı ekrana yazdırılsın. CK katsayısı erkekler için 0,9, kadınlar için ise 0,8 olsun. Yaş bilgisi olarak 0 girildiği zaman programın çalışması dursun. Bu problemi çözmek için bir algoritma geliştirelim.

Girilen Veriler : B : Boy

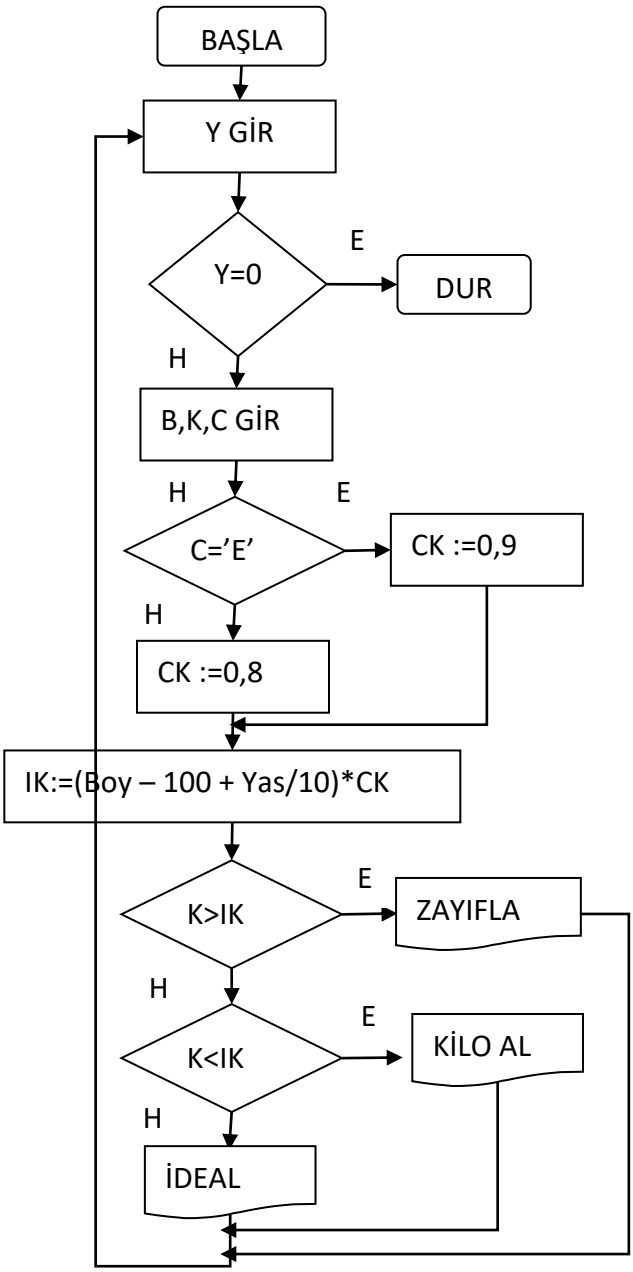
Y : Yaş

K : Kilo

C : Cinsiyet

Çıkan Bilgiler: IK: İdeal kilo

1. ***Y (yaş) gir.***
2. ***Y=0 ise programı durdur.***
3. ***B, K ve C gir. // Boy, kilo ve cinsiyet gir***
4. ***C (cinsiyet) erkek ise CK değişkenine 0.9 değerini ata ve 6. Adıma git.***
5. ***C (cinsiyet) erkek değilse CK değişkenine 0.8 değerini ata.***
6. ***IK=(Boy – 100 + Yas/10)*CK formülü ile IK hesapla.***
7. ***K>IK ise ‘ZAYIFLA’ yazdır ve 1. Adıma git.***
8. ***K<IK ise ‘KİLO AL’ yazdır ve 1. Adıma git.***
9. ***Diğer durumda yani K=IK ise ‘İDEAL’ yazdır ve 1. Adıma git.***



Algoritmanın C# dilinde kodlanmış şekli aşağıdadır:

```
static void Main(string[] args)
        {
            Byte y, k,b;
            double ik, ks;
            char c;
```

```
        Console.Write("Yaşı Giriniz:");
        y = Convert.ToByte(Console.ReadLine());
        while (y > 0)
        {
            Console.Write("Boyu Gir(cm.):");
            b = Convert.ToByte(Console.ReadLine());
            Console.Write("Kiloyu Gİr(kg.):");
            k = Convert.ToByte(Console.ReadLine());
            Console.Write("Cinsiyeti Gir(E/K):");
            c =Convert.ToChar(Console.ReadLine());
            if (c =='E' || c=='e')
                ks = 0.9;
            else
                ks = 0.8;
            ik = (b - 100 + y / 10) * ks;
            Console.WriteLine("İdeal Kilouz:" + ik);
            Console.Write("Yaşı Giriniz:");
            y = Convert.ToByte(Console.ReadLine());
         }
        Console.ReadLine();
    }
```

Örnek 7: Eleman sayısı belli olan bir dizide öğrencilerin boy bilgileri herhangi bir sıra gözetilmeksizin bilgisayara girilmektedir. Sınıftaki en uzun boylu ve en kısa boylu kişinin bilgilerini ekrana yazdıran programın algoritmasını gösteriniz.

1. N gir // Öğrenci sayısı

2. İ:=1 değerini ata // Sayacı 1'den başlatıyoruz.

3. (B(i)) gir. // i. Öğrencinin boyunu gir

4. EUB:=B(1) ; EKB:=B(1) // 1. Öğrencinin boyunu en uzun ve en kısa boy olarak ata

5. i := i + 1 komutu ile sayacı 1 artır.

6. B(i) gir. // i. Öğrencinin boyunu gir.

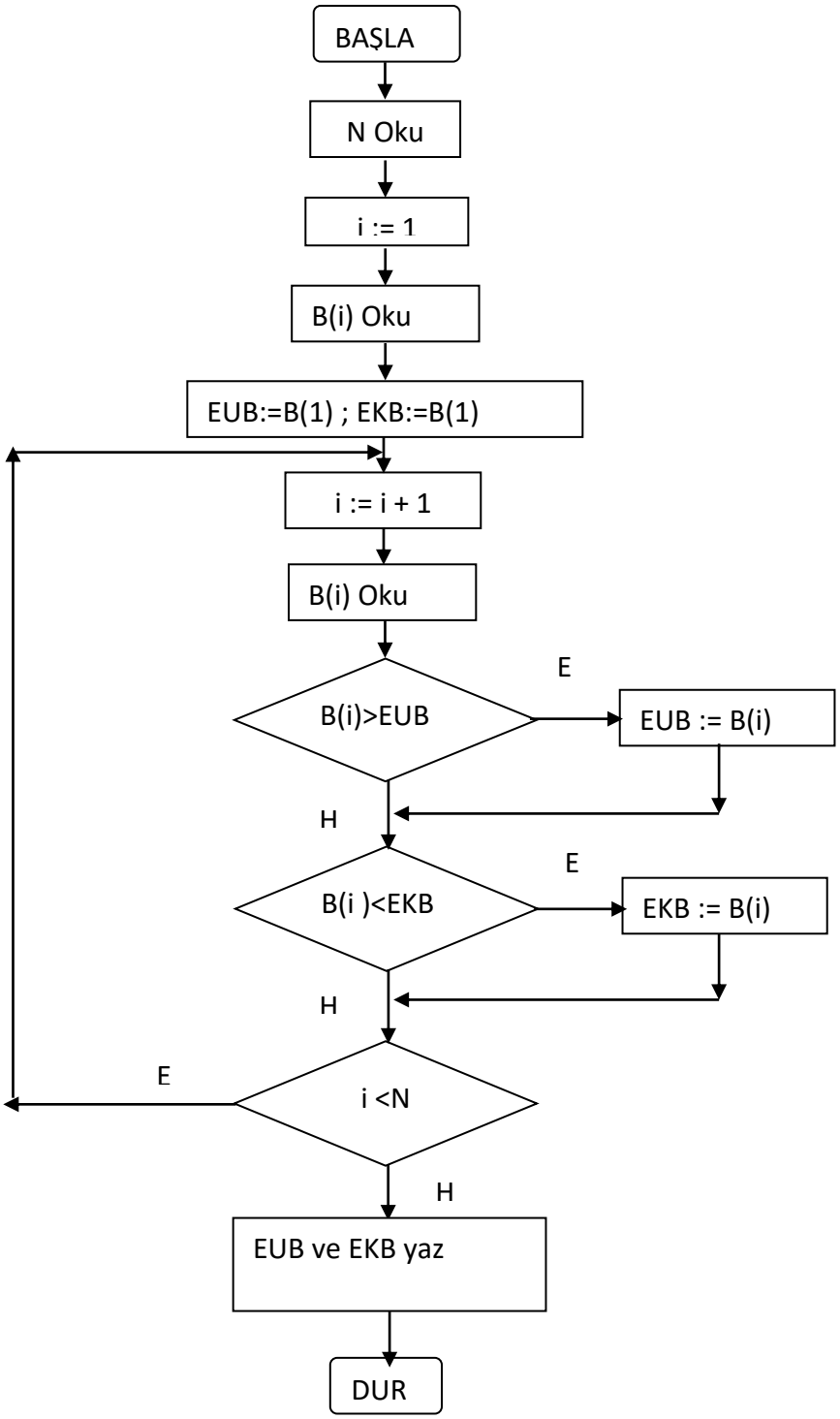
7. B(i), EUB'den büyükse EUB := B(i) komutu ile B(i) EUB yap.

8. B(i), EKB'den küçükse EKB := B(i) komutu ile B(i) EKB yap.

9. I<N ise (öğrencilerin tamamı değerlendirilmedi ise) 5. adıma git.

10. EUB ve EKB yaz.

11. Programı durdur.



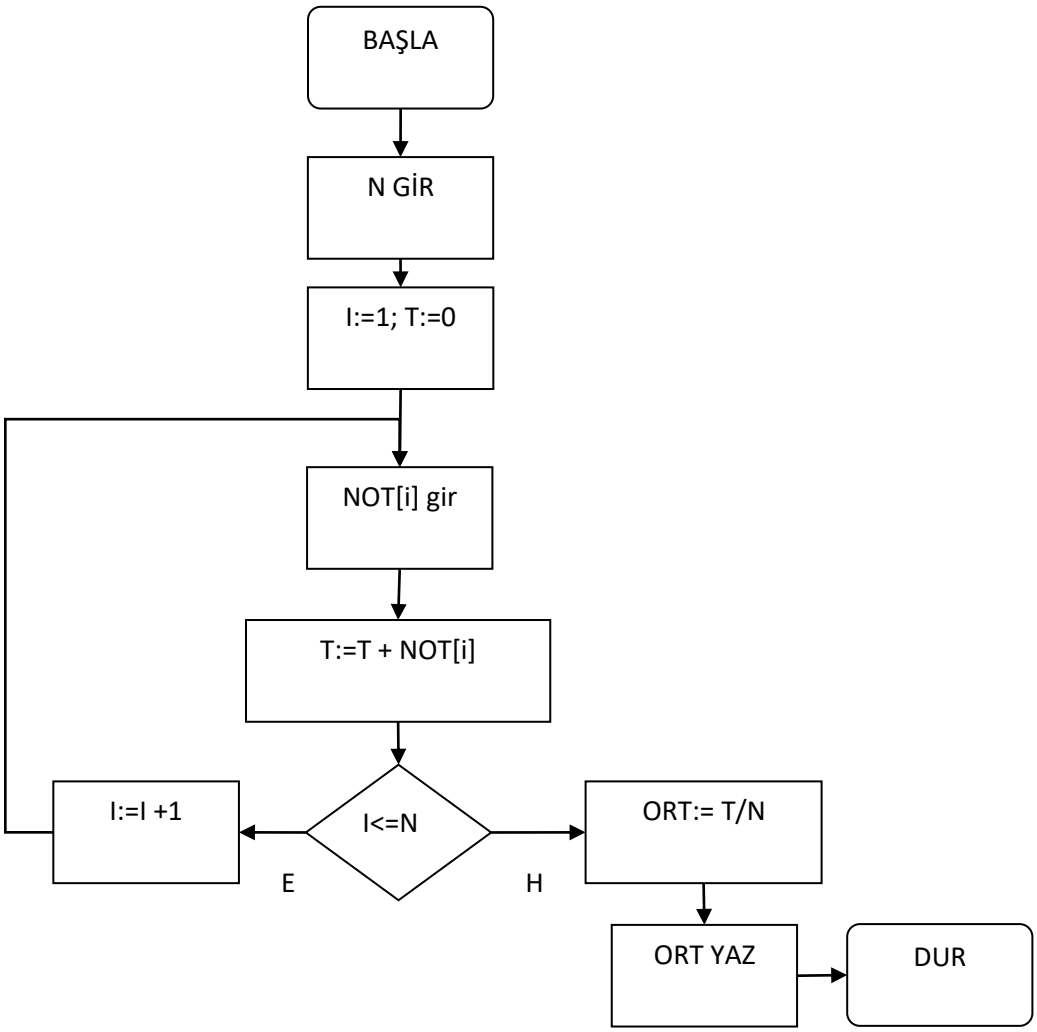
```
static void Main(string[] args)
        {
            int n = 0,i=0,not,EB=0,EK=100;

            Console.Write("Öğrenci adedini giriniz:");
            n=Convert.ToInt32(Console.ReadLine());
            //int[] D = new int[n];
            for (i = 1; i <= n; i++)
            {
                Console.Write(i+ "  . öğrencinin notunu gir:");
                not = Convert.ToInt32(Console.ReadLine());
                if (not>EB)
                  EB=not;
                if (not<EK)
                  EK=not;
            }
            Console.WriteLine("EN büyük Not:" + EB);
            Console.WriteLine("En küçük Not:"+EK);
            Console.ReadLine();
        }
```

Örnek 8: Bir sınıftaki N adet öğrencinin notları bilgisayara dışarıdan girilmektedir. Notların ortalamasını hesaplattırarak sonucu ekrana yazdıran programın algoritmasını geliştiriniz.

Girilen Veriler: N : Sınıftaki öğrencilerin toplam sayısı

Not[i] : i. Öğrencinin notu

Çıkış Bilgileri: Ort : Notların ortalaması



1. ***N gir. // Öğrencilerin sayısı***
2. ***I (sayaç) ve T (notların toplamına) başlangıçta 0 değerini ata.***
3. ***NOT[I] gir // i. Notu gir.***
4. ***T:=T + NOT[i] // i. Notu toplama ekle***
5. ***Eğer I<=N değilse (yani tüm öğrencilerin notu girildi ise) 7. Adıma git.***
6. ***Eğer I<=N ise I:=I+1 deyimi ile sayacı bir artır ve 3. Adıma git.***
7. ***ORT=T/N ile not ortalamasını bul.***

8. ***Ortalamayı (ORT) yazdır.***

9. ***Programı durdur.***

Algoritmanın C# diline dönüştürülmüş şekli ise aşağıdaki gibidir:

```
static void Main(string[] args)
        {
            double toplam, notu, ort;
            byte n, i;
            toplam = 0;
            Console.Write("Öğrenci Sayısını Gir:");
            n = Convert.ToByte(Console.ReadLine());
            for (i=1;i<=n;i++)
            {
                Console.Write(i + "  . öğrencinin notunu giriniz:");
                notu = Convert.ToDouble(Console.ReadLine());
                toplam=toplam+notu;
            }
            ort = toplam / n;
            Console.Write("Not ortalaması:" + ort);
            Console.ReadLine();
              }
```

Fibionacci Dizisinin Elemanlarını Bulma

Örnek*:* Bilgisayara girilen N sayısına kadar fibionacci dizisinin elemanlarını bulan ve diziyi ekrana yazdıran programın algoritmasını geliştirelim.

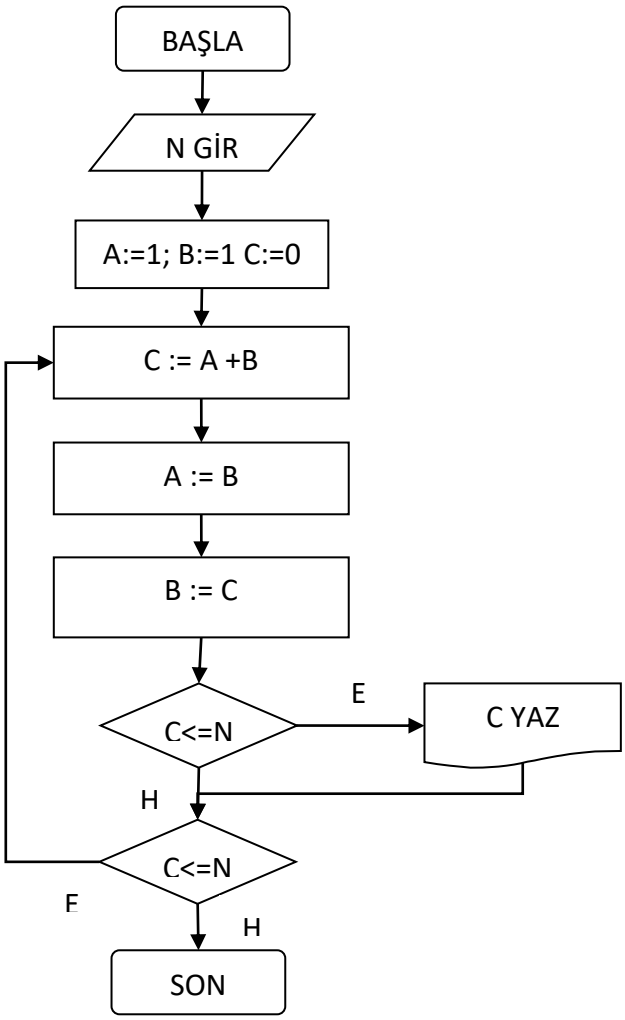
Fibionacci dizisi: 1 1 2 3 5 8 13 21...

Dizide 1. ve 2. elemanın değeri 1 dir. Üçüncü eleman kendisinden önce gelen son iki elemanın toplamına eşittir. Dördüncü eleman ve diğer elemanlar da aynı şekilde hesaplanmaktadır. Yani kendisinden önce gelen son iki elemanın toplamına eşittir.

A_1:=1;

A_2:=1;

$A_i := A_{i-1} + A_{i-2}, \qquad i > 2$



Fibionacci Dizisini Oluşturma

1. Başla
2. N gir // N'ye kadar fibionacci dizisi hesaplanacak
3. A:=1; B:=1; ve C:=0 değerlerini ata. // Başlangıçta A 1. B ise 2. Eleman olarak değerlendirildi.

4. C:=A+B komutu ile sıradaki elemanı hesapla
5. A:=B komutu ile bir önceki elemanı A'ya ata
6. B := C komutu ile yeni oluşturulan elemanı B'ye ata
7. Son hesaplanan eleman yani C, N'den küçükse C'yi yazdır.
8. Son hesaplanan eleman yani C, N'den küçükse 4. Adıma git
9. Son

Algoritmanın C# dilinde kodlanması aşağıdaki gibidir:

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

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int a=1,b=1,c=0,n;
            Console.Write("N sayısını giriniz:");
            n=Convert.ToInt32(Console.ReadLine());
            Console.Write("Fibonacci Dizisi:  " + a + "   " + b + "
");
            do
            {
                c = a + b;
                a = b;
                b = c;
                if (c <= n)
                    Console.Write(c + "    ");
            }
            while (c <= n);
            Console.ReadLine();
        }
    }
      }
```