Bilgisayar Kitabı

Mehmet Gençer

İçindekiler

Şekil Listesi

Bölüm 1

Giriş

1.1 Basit prensipler, karmaşık işler, farklı bir yaklaşım

Bilgisayar nasıl çalışır, ve bunu bilmek ne işinize yarar? Modern yaşam tarzı bize kendi işimize bakmamızı söylüyor. Neredeyse hepimiz işte veya özel hayatımızda bilgisayar kullanıyoruz ve bunu yaparken de işimizi ilgilendiren belirli birkaç yazılımla kısıtlıyız. Bu denli hayatımızın içine giren makinenin çevresinde ise bir gizem duvarı var. Bu kullandığımız başka makinelerde olmayan bir duvar ve bize bu makinenin çok karışık olduğunu, onu anlamak için boşuna çabalamamamız gerektiğini söylüyor. Kitapçıların bilgisayarlarla ilgili bölümündeki kitaplardan pekçoğunun başlığı bile bize birşey ifade etmiyor, ve böylelikle biz de bu kadar yoğun kullandığımız bir makinenin nasıl çalıştığını anlamak konusunda baştan pes etmiş oluyoruz. Modernizmin ve uzmanlaşmaya dayalı bir hayatın kendi alanımızın dışına çıkmamamızı buyuran görünmez sopasına boyun eğiyoruz.

Yıllardır uzmanlık alanı bilgisayarlar olmuş biri olarak pek çok insanın problemlerini dinlemiş ve çözmelerine yardım etmişliğim var. Bu insanların benim de kullandığım kelime işlemci, hesap tablosu programları gibi yazılımlar konusundaki kıvraklığı beni hep şaşırtmıştır. Programın yüzlerce menüsünden kayda değer bir kısmını bilirler, klavye kısayollarını ezberlemişlerdir, hem de benim yapabileceğimden çok daha fazlasını. Buna karşılık pekçok temel bilgiden habersizdirler. Çok kalabalık bir masaüstüne ve dosya miktarına sahiptirler, ancak disk bölümlemenin

ne olduğunu, yedeklemenin nasıl yapılacağını bilemezler. Sık sık bilgisayarla yazı yazarlar, ancak yazılarını yapısal olarak biçimlendirmek yerine birilerinin ona öğrettiği gibi her bölüm başlığını fare ile seçip koyultur ve teker teker yazı biçimini ayarlarlar; hassas bilgilerle uğraşmalarına rağmen bilgi güvenliği veya şifreleme gibi temel özelliklerden ve bunu sağlayan araçlardan habersizdirler. Çoğu insanın bu becerikli makineyi kullanma şekli kulaktan kulağa geçen bazı bilgilere dayanır, farenin iki düğmesine kısılıp kalır, ve bunun ötesini karıştırmanın tehlikeli olacağı korkusuyla biçimlenir. İnsan, yarattığı uygarlığın en güçlü hizmetkarlarından biri olan bu makine karşısında bir yanıyla oldukça itaatkar ve edilgen durumdadır.

Bu kitap bilgisayarın nasıl çalıştığına, nasıl programlandığına, teknolojisinin geçmişine ve geleceğine, kısacası konunun tamamına dair. Bu kitabı okumanın bilgisayar karşısındaki bu hissiyatınızı değiştireceğini, bu makineyle olay deneyiminizi başka bir yola sokacağını, ve kendi işlerinizin bilgisayarla ilgisi her neyse daha sağlam bir yere oturacağını umuyorum. Basit ilmiklerden örülerek oldukça karmaşık ve becerikli sistemlerin nasıl yapıldığına dair temel bir kavrayış edinebilir, ve böylece bilgisayarın başına oturduğunuzda onunla ürküntü yerine takdir ve artan bir merak temelinde bir ilişki kurabilirsiniz. Bu kitap yetersizse bile bunu denemekten vazgeçmeyin. Ben 30 yıldır bilgisayar programlıyorum, ve dönüp baktığımda bu işe dair aklıma gelen ilk sıfat 'zor' değil 'zevkli' oluyor, çünkü bu sanıldığının aksine rutin değil yaratıcı bir iştir. Programlama ile ilgili bölüme (bölüm 7) geldiğinizde siz de bu şekilde hissederseniz belki de bunu meslek olarak seçmeyi düşünmelisiniz. Ancak tersi olsa bile bu yolculuğu bilgisayarlarla ilişkininiz konusunda hafiflemiş ve makine karşısında hakimiyeti ele geçirmiş olma duygusuyla bitireceğinizi umarım.

Bilgisayarların geniş dünyasının belirli küçük bir alanına dair bir uzmanlık kitabı yazmak yerine yerine konunun bütününe dair bir kitap yazmaya girişmek riskli bir tercih. Üstelik konunun bütün yönlerine el atmak kaçınılmaz olarak bunların hepsinde ancak sığ ve kısıtlı bilgiler verebileceğim anlamına geliyor. Evet bilgisayarlar karışık makinelerdir ve bu yüzden Amazon.com'da bu konuda beşyüz bine yakın kitap var! Öte yandan bilgisayarın temel prensipleri şaşırtıcı derecede basittir. Ancak bu prensiplerle çok fazla sayıda ilmek atılarak karmaşık bir sistem ortaya çıkmıştır, ve sözkonusu beşyüzbin kitabın hemen hepsi bu ilmeklerin bir

tanesini anlatırken çok azı büyük resme dairdir. Oysa yolumuz bilgisayar teknolojisiyle neresinden kesişirse kesişsin büyük resme bakmaya ihtiyacımız vardır.

1.2 Binbir yüzlü makine, binbir aracı

Artık hemen hepimiz her gün değişik türden bilgisayarlarla, birçok farklı iş yapıyoruz. Cep telefonlarından parekende satış cihazlarına, dizüstü bilgisayarlardan bankamatiklere kadar birçok makine bilgisayar teknolojisine dayanıyor. Bu makinelerle iletişim, müzik dinlemek, dijital kütüphanelere ve arama motorlarına erişmekten tutun da metin hazırlamak, tablolarla hesaplamalar yapmaya kadar farklı işler görüyoruz. Çoğu meslek erbabı kendi alanlarına özgü bilgisayarlar kullanıyor: doktorların ultrason ve MR cihazları, oto teknisyenlerinin motor test cihazları gibi. Bunların ötesinde gündelik hayatta karşılaşmasak ta haberlerine sık sık rastladığımız bilgisayarlar var: Dünya satranç şampiyonunu yenen DeepBlue, Mars gezegeninde keşif yapan robotlar gibi.

Bunca farklı işimizi görmesine rağmen, daha doğrusu tam da bu sebepten, bilgisayarın becerilerinin tam olarak hangi temellere dayandığını anlamak işin uzmanı olmayanlar için oldukça zordur. Bir bilgisayarın becerisi 2.60 GHz işlemci, 2 GB RAM, 250 GB sabit disk gibi bir takım ölçütlerin arkasında gizlenmiştir. Bu ölçütlerle artık çok sık karşılaşmamıza rağmen yine de çoğumuz ihtiyacımıza uygun makine seçmek için bu ölçüleri yorumlamakta çok zorlanırız.

Bu yeni makinenin doğasına dair bir türlü net bir fikir edinemiyoruz. Oysa evimizdeki saat ya da çamaşır makinesi arızalansa, otomobilimizden olağandışı sesler gelmeye başlasa sorunun kaynağıyla ilgili çoğunlukla iyi bir tahminimiz olur. Hele ki benzer türden bir makineyi açıp inceleme şansımız olduysa tahminlerimiz daha da isabetlidir. Çalışması belirli bir fiziksel harekete dayanan makineleri doğal olarak daha kolay anlıyoruz, özellikle de bu tür makineler -farklı modelleri olsa da- tek bir işi hep aynı şekilde yaptıkları için. Oysa bilgisayar işini yaparken çok az fiziksel harekete dayanıyor, soğutucu pervanenin dönmesi gibi, ki bu da işin oldukça önemsiz bir yanı. Bilgisayarın içini açıp baksanız bile hareket eden hiçbir parça görmüyorsunuz. Birşeylerin olup bittiği, hem de çok hızlı olup bittiği belli ama makine bize bu olup bitenin *doğasına* dair hiçbir ipucu vermiyor.

Bilgisayarın doğasını anlamamızı zorlaştıran unsurlardan biri

de onu hep bir aracı vasıtasıyla kullanmamız. Müzik çalma veya pdf görüntüleme programı gibi kullanımı görece daha basit aracıların yanısıra veri tabloları hazırlama, grafik çizme gibi daha karmaşık işlevleri olan aracı programlar var. Bunların hepsi belirli bir işte uzmanlaşmış programlar, yada yaygın adıyla yazılımlar. Bilgisayarı bu aracı programlar varsıtasıyla kullanmak esas olarak bir şöförün kullandığı bir araçla seyahat etmeye benziyor. Bu şöförler hızlı ve dürüst, ama bir o kadar da ketumdurlar. Bzi onların dilini konuşmaya ve kurallarını benimsemeye zorlarlar, derdimizi anlamak için çaba sarfetmezler. Aracın kendisi bize çok karmaşık göründüğünden kendimiz direksiyona geçmektense kimi hallerini beğenmesek te bu şöförlere razı oluruz.

Üstelik bu program şöförler, ya da yaygın ismiyle paket programlar, bizim ancak belirli bir mahallede işimize yararlar. O mahallenin bilmediği sokakları varsa, bilgileri eksik ya da yanlışsa onlara doğrusunu öğretme şansımız yoktur. Patronlarını aramak ta pek işimize yaramaz çünkü oldukça zengin ve meşguldürler (ileride patronu olmayan, kamuya ait yazılımların, eğer becerebiliyorsanız sizin tarafınızdan eğitilebileceğini göreceğiz). Bu tür durumlar çoğu zaman bizim isteğimizden vazgeçmemizle sonuçlanır! Eğer vazgeçilmesi zor bir ihtiyacınız, ciddi miktarda paranız varsa, ve iyi bir bilgisayar programcısı bulacak kadar şanslıysanız (onlardan pek fazla yoktur) size yeni bir program şöför yaratmasını ve iş görmek istediğiniz muhiti öğretmesini isteyebilirsiniz.

Bu yaygın durumun önemli bir sonucu bizim bilgisayar denilen makineyi hiçbir zaman doğrudan pilot koltuğuna geçip kullanmamamızdır. Nasıl çalıştığına dair kavrayış eksikliğimizin temelinde de bu yatar: bu makineyle onca zaman geçirmemize rağmen onunla olan deneyimimiz hep dolaylı, ketum pilotlar aracılığıyla yaşanır.

Burada kullandığım iğneleyici dil ile paket programları topyekün kötülemek niyetinde değilim. Binlerce, hatta onbinlerce hazır program var ve bunlar bizim bilgisayarı etkili ve pratik bir şekilde kullanmamız için vazgeçilmezdir. Herbiri çok sayıda yetenekli insanın emeğiyle oluşmuştur ve içlerinde birçok uzmanlık alanına dair bilgi birikimini barındırırlar. Bir yazılımın inşası sırasında programcı yazılımın konusuyla ilgili uzmanların nasıl çalıştığını inceler ve onların bazı işleri nasıl yaptığını bir bilgisayar programı olarak ifade eder. Uzman insanları kopyalayamadığı-

mız halde bu bilgisayar programını kolayca çoğaltabiliyoruz. Bu sayede doktorlardan muhasebecilere, mimarlardan bankacılara kadar birçok insan sıkıcı ve rutin işleri bilgisayara yaptırarak işin insan müdahalesine ve muhakemesine en çok ihtiyaç duyulan taraflarına odaklanabiliyor. Bu yönüyle paket programlar yararlıdırlar. İleriki bölümlerde kullanıcılarının bu program yardımcılara hükmedebilme ve onları eğitebilmesinin koşullarını tartışacağız.

Eğer son 60 yılda olduğu gibi gitgide artan sıklık ve hallerde bu makineleri kullanacaksak onlara daha hakim olmamız gerekeceği açıktır. Bu hakimiyetin koşulu genellikle daha fazla para verip ihtiyaca özel yeni paket programlar satın almak olmuştur. Bu paket programlar bilgisayarın kendisinde kat kat daha pahalı olabiliyor, üstelik sık sık yenilenmeleri gerekiyor ki yaratıcılarının öğrettiği yeni özelliklerden yararlanabilelim. Daha sıkıntılı olan şu ki gerçekten özgün bir iş yapmak istediğinizde bu paket programların hiçbiri ihtiyacınızı tam olarak karşılamaz. Hazır giyimin işinizi görmediği, iyi bir terziye ihtiyacınız olduğu noktadasınızdır. Oysa terzilerin çoğu artık hazır giyim şirketleri için çalışıyor! Örneğin e-ticaret sitelerine baktığınızda bu web sitelerinin sıkıcı derecede birbirine benzediğini görürsünüz. Size özel bir program yaratacak bir programcı bulmak zordur; hele iyi olanların çoğu da büyük yazılım şirketleri için çalışıyor çünkü bu işte bir programı bir kez yaptırıp binlerce müşteriye satmak işin asıl karlı tarafı. Bunlar işe yaradığı sürece bu işte bir sorun görmüyorum. Ancak bu kitabın bir amacı da paket programların çizdiği sınırların ötesini merak eden veya oraya geçmek isteyen okurlara yol göstermek.

Sınırın ötesindeki bu alan pek çok açıdan ilginçtir. Herşeyden önce bilimsel olarak ilginçtir. Bilgisayar tek bir problem çözmez; problemin kendisi ve çözüm yöntemi açıkça tarif edilebildiği takdirde çok farklı problemleri çözebilir. Ayrıca mesleki ve ticari olarak ilginçtir. Mesleki projelerinizde istediğiniz yöntemleri uygulamak, bilimsel araştırmalarda verileri işlemek, ses veya görüntüler yaratmak gibi işlerden tutun da katma değeri olan yeni bir ticari girişimi gerçekleştirmeye kadar pek çok amaç için bu sınırı geçme ihtiyacı hissedebilirsiniz [1]. İhtiyaçlarınız piyasada varolan paket programların ötesine geçtiğinizde –ki işinizde bilgisayardan yararlanıyorsanız er yada geç bu sınırı geçersiniz– kendinizi yeni bir alanda bulursunuz. İhtimal o ki okurların büyük bir bölümünün bilgisayar programcısı olmak gibi bir niyeti yok. Bu tür ihtiyaçlarınız için bir programcıdan

yardım alarak ilerleyecekseniz bile bilgisayarlar konusunda temel bir kavrayış çok daha rahat ve ne yaptığınızı bilerek ilerlemenize yardım edebilir. Kısacası program yazmayı mutlaka denemelisiniz!

Yakın zamanda ABD'den gelen verilere göre[2] gelecek yıllarda oluşacak mühendislik istihdamınınyarıdan çoğu bilgisayar alanında olacak. Diğer gelişmiş ve gelişmekte olan ülkelerde de böyle bir trend izleyeceğini düşünürsek genç okurlarımızın önemli bir kısmı bilgisayarı anlama, kullanma, ve programlama işini ciddiye alıyor, bunu bir meslek olarak edinmeyi düşünüyor demektir. Bu okurlar için de kitabın derin olmasa da kapsamlı bir giriş olacağını umuyorum.

1.3 Makinenin gerçek yüzü

Bizim binbir çeşit ihtiyaca özel program aracılığıyla deneyimlediğimiz bilgisayar bunların arkasında temel ve basit birkaç işleve dayanır. Bilgisayarın *donanım* da denilen bu çıplak makine hali, üzerinde çalışan programa, diğer adıyla *yazılım*a göre bir kelime işlemciye, bir hesap makinesine, ya da bir müzikçalara dönüşür. Aynı bir insanın aldığı eğitimle müzisyen, marangoz, veya inşaat mühendisi olması gibi. Kaldı ki bilgisayar donanımı istendiği zaman başka bir yazılım çalıştırılarak istenilen role bürünebilen tek bir bedene benzer. Bu yüzden bilgisayarın doğasını anlamak için bilgisayar programı (yazılım) ve bilgisayarın fiziki varlığı, henüz harekete geçmemiş becerileri (donanım) arasında bir ayrım yapmak gerekiyor.

Önce donanımdan biraz bahsedelim. Bilgisayarın oldukça karmaşık çeşitli türden işlerin altından kalkabilmesi için temel bazı becerilere ihtiyacı vardır. Bunlar kısmen biz insanların temel zihinsel ve algısal becerilerine benzetilebilir: ayırdetme, karşılaştırma-birleştirme, ve hatırlama. Bizden farklı olarak bilgisayar donanımının bu temel becerileri olabildiğince sadedir. Bilgisayarın bedenini oluşturan elektronik devreler sadece elektrik voltajının varlığını ve yokluğunu (ya da 1 ve 0'ı) birbirinden ayırdedebilir. Başka bir değişle bilgisayarın anladığı dilde bir veri sadece iki farklı değer alabilir. Bu voltaj değerlerine 'doğru'-'yanlış', veya 1-0 gibi etiketler koyabiliriz. Böyle bir ayırtetme düşünülebilecek en basit ayırtetmedir. Ancak birden fazla veri değeri kullanılırsa oldukça karmaşık bilgiler bu temel etiketlerle yazılabilir. Bizim gündelik hayatta kullandığımız ondalık tabanda bir sayıyı ele alalım. Örneğin 42 sayısı; bu iki hanelik sayıyı ikilik tabanda

101010 diye, ancak altı hanede ifade edebiliyoruz, ki benim gibi bu işle uğraşan biri için bile kalem oynatmak gerektiren hiç doğal olmayan bir ifade şekli. Ama bilgisayarın kalabalık hanelerle bir problemi yoktur. Ayırdetmenin bu en basit biçimi bile, yeterince hane kullanırsak, büyük sayıları ifade etmek için yeterlidir. Örnekteki sayıyı bilgisayara *anlatmak* için altı tane elektrik kablosuna doğru sırada yüksek-düşük-yüksek-düşük-yüksek-düşük voltaj bağlamak yeterli olurdu. İkilik tabanın nasıl kullanıldığını bölüm 2'de, ve basit sayılar dışındaki bilgilerin (yazılar, sesler, görüntüler) bu basit prensipler temelinde nasıl temsil edildiğini bölüm 3'te inceleyeceğiz.

Donanımın ikinci temel işlevi verileri karşılaştırma ve birlikte değerlendirmedir. Bilgisayarın dünyasında gerçeklik sadece 1 ve 0'larla ifade edildiğinden bu karşılaştırma ve birleştirme işlemlerinin sonucu da aynı şekilde ifade edilir. Bölüm 2'de bu işeri yapan elektronik devrelerden bahsederken 'mantık köprüleri' terimini kullanacağız. Bu kafa karıştırıcı olabilir. Çünkü bu terimi kullandığımızda 1 ve 0 yerine 'doğru' ve 'yanlış' etiketlerini kullanmış oluyoruz. Elektronik devrelerde bizim mantık yürütürken kullandığımız 've', 'veya' gibi karşılaştırmalara karşılık gelen köprülerden bahsedeceğiz. Burada karşılaştırma konusu olan değerler iki sayının haneleri olduğunda aslında ikilik sistemde 1 ve 0 değerlerini karşılaştırıyoruzdur. Böyle bir karşılaştırmanın (örneğin a ve b ile temsil edilen iki verinin değerlerinin 'a ve b', ya da 'a veya b' karşılaştırmalarına tabi tutulması gibi) sonuçları mantıksal 'doğru' veya 'yanlış' değerleri olarak etiketlenir. Bu sonuçların da başka elektronik köprüler kullanılarak karşılaştırılması mümkündür. Köprüler açısından verilerin değerinin 'doğru'-'yanlış', veya 1-0 şeklinde etiketlenmesi bir önem taşımaz. Mantıksal doğruluk değeri ve ikilik tabanda sayı değeri arasındaki bu belirsizlik kafa karıştırıcı olsa da bölüm 2'deki açıklamaların bu karışıklığı gidereceğini sanıyorum. Karşılaştırma dışında iki değerin birlikte değerlendirilmesinden sözettik. Sayılar sözkonusu olduğunda (ve herşeyi sayılarla ifade edeceğimiz için) bu işlemler toplama ve çarpma gibi temel aritmetik işlemlerden ve büyüklük karşılaştırması gibi işlemlerden oluşuyor. Bunun temel bir sorun teşkil etmediğini göreceğiz. Örneğin ikilik tabanda iki sayının karşılaştırması ondalık tabanda sayıların karşılaştırılmasına benzer. Bizim çok doğallıkla yapıverdiğimiz bu işlem en soldaki, en büyük haneden başlayarak ve sağa doğru ilerleyerek rakamların karşı-

laştırılmasından ibarettir. Eşitliğin bozulduğu ilk hanede hangi sayının büyük olduğuna karar veririz (ya da ikisi eşit çıkabilir). Örneğin 268573 ve 268578 sayılarında ilk iki hane aynıyken ilk sayının üçüncü hanesi diğerinden büyüktür ve hem sayılar biraz uzunca olduğu hem de birbirine benzediği için kendinizi yukarıda tarif ettiğim adımları uygularken bulabilirsiniz (oysa 68 ve 41 sayıları sözkonusu olsaydı bir hamlede cevap verirdik). Bu sayıları ikilik tabanda yazarak gözlerinizi zorlamayacağım, ancak prensip aynıdır. İkilik tabanda yazım bizim yaptığımız işi elektronik mantık köprüleri ile yapmamıza imkan sağlar. Toplama ve çarpma gibi işlemlerin nasıl yapıldığından bölüm 2'de bahsedeceğiz.

Son olarak hatırlama konusu var. Eğer hatırlama mümkün olmasaydı duruma göre hareket edemezdik. Bilgisayar için de bu böyledir. Bir işlem adım adım gerçekleştirilirken bir önceki adımın sonuçlarının hatırlanması ve ileriki adımlarda veri olarak kullanılması gerekecektir. Örneğin 2+3+4 işlemini tek bir adımda yapaca devrelerimiz yok. Bunun yerine önce ilk toplama işlemi yapılır ve sonuçları(2+3=5) bellekte saklanır, daha sonra ikinci bir toplama işlemi (5+4) ile sonuç bulunur. Böylece ne kadar karışık olursa olsun bir aritmetik ifade sırasıyla toplama-çarpma gibi temel bir işlemin uygulanması, sonucun belleğe yazılması, ve ikinci bir işlemde kullanılması, vs., şeklinde herbiri tek bir aritmetik işlemin yapıldığı adımlara indirgenebilir. Bunu sağlayan bilgisayar belleğinin mantık köprüleri kullanılarak nasıl inşa edildiğine yine bölüm 2'te göz atacağız. Günümüzde bilgisayarların bellek kapasitesinin çok arttığının, ve bilgi işlemek kadar depolamak için de kullanıldığının altını çizelim. Belleğin nasıl organize edildiği ve depolama ile ilgili konulara bölüm 3 ve bölüm 4'te değineceğiz.

Bilgisayar donanımı bu minimalist işlevlerin tekrar tekrar ve farklı kombinasyonlarda kullanılmasıyla inşa edilir. Bol miktarda direnç, transistör, ve kablo alıp önce herbiri bunlardan üç-beş tane içeren birsürü mantık köprüsü yapabilir, sonra bunları sayıları toplayıp karşılaştıracak ve sonuçları hatırlayacak kombinasyonlarda birleştirebilirsiniz. Gerçekten de 1940'larda yapılan ilk bilgisayarlar böyleydi, ve çok büyük bir odaya ancak sığıyorlardı. Üstelik bugünkülerden çok daha yavaş ve bellek kapasiteleri düşüktü. Buna rağmen işlemleri hatasızca ve –herşeye rağmen– insanlardan çok daha hızlı bir şekilde yapmaları tüm dünyada bu teknolojinin üzerine muazzam bir yatırım yapılması ve hızla iyileşmesiyle sonuçlandı. Günümüzde bilgisayar donanımları

birkaç santimetrekarelik çipler üzerine iki milyar transistörün sıkıştırılmasına dayanan bir teknolojiyle yapılıyor, ve yukarıda bahsedilen karşılaştırma-birleştirme, kaydetme-hatırlama işlemlerini çok hızlı yapıyorlar (saniyede 2 milyar işlem ve üzeri). Bu tür bilgisayar sistemlerinin inşa edilmesi Bilgisayar Mühendisliği disiplininin konusudur.

Bu kadar hızlı giden bir aracın direksiyonuna hükmedemezsiniz. Bu yüzden bilgisayara hangi işlem sırasını takip edeceğini bizim canlı olarak vermemiz mümkün değildir. Üstelik bu hem hataya açık olurdu, hem de işi bizim yapmamızdan bile zahmetli olurdu. Bu yüzden aynı bir çamaşır makinesinin farklı yıkama-sıkma ve sıcaklık kombinasyonlarından oluşan 'beyaz', 'renkli', 'yünlü' programı olması gibi önceden hazırlanmış işlem dizileri kullanıyoruz. Bu işlem dizilerine program veya yazılım diyoruz. Bir bilgisayar programının satırları 'Belleğin 1 ve 2nci noktalarının içinde yazan sayıları topla, sonucu 3e koy', veya 'Belleğin 1inci noktasındaki sayı 2nci noktasındaki sayıdan büyükse iki adım öteye atla' gibi komutlardan oluşur, üstelik bütün bunları makinenin anlaması için ikilik tabanda ifade edilmeleri gerekir! Bir birgisayar donanımı sınırlı sayıda ve belirli komutları anlar. Örnekteki 1, 2, 3 bellek noktalarının değerleri, ve dolayısıyla komutların sonuçları değişse de bilgisayar donanımının anlayacağı komut seti değişmezdir, günümüzde tipik bir bilgisayar bu türden 100 civarı komutu anlayabiliyor. Ancak bu programlama konusunda gözünüzü korkutmasın. Bilgisayar programcılarının ilk yaptıkları iş kendi hayatlarını kolaylaştıracak programlar yapmak olmuştur. Günümüzde bilgisayar programlama yapılacak işin konuşma diline oldukça yakın ve zengin bir dilde ifade edilmesi, ve bu kolaylaştırıcı programlar sayesinde makinenin anlayacağı şekle dönüştürülmesi ile yapılıyor. Günümüzde böyle birçok farklı programlama dili kullanılıyor, ancak bunların hepsi nihai olarak aynı makine diline dayandığından olsa gerek birbirine oldukça benzerler. Programlamanın temel pratiklerine bölüm 7'da değineceğiz ve bu dillerden birini kullanarak bazı programlar yazacağız.

Bir matematik kitabında kullanılan dilin aksine bilgisayar programları çok daha az sayıda kelime ile ifade edilir ve hiçbir belirsizlik içermezler. Bir matematiksel yöntemin temel işlemlerden ibaret basit adımlar olarak ifade edilmesi, yani bilgisayar programlama, Bilgisayar Bilimleri disiplininin konusudur. Esa-

sen bu çalışma alanı 'nelerin otomatikleştirilebileceği' sorusuyla uğraşır (Knuth, 1996). Buna karşılık Bilgisayar Mühendisliği ise "otomatiği gerçekleştirecek makinelerin yapımı" ile uğraşıyor. Bu makinaların metalden (mekanik prensiplere dayanarak), silikondan (elektrik prensiplere dayanarak), genlerden (biyokimyasal prensiplere dayanarak), veya atomlardan (quantum computing-kuantum mekanik prensipler) yapılmasının bilgisayar bilimleri açısından esasen bir önemi yoktur, ki hepsi de yapılmış veya denenmektedir (ancak elektrik-silikon teknolojisinin yaygınlığı doğal olarak hem bilgisayar bilimleri hem de bilgisayar mühendisliği alanındaki uğraşın çoğunu bu teknolojiye yönlendirmiştir).

Bilgisayarın donanım ve yazılımına karşılık gelen bu iki temel disiplin arasında belirgin bir uğraş alanı farkı bulunmasına rağmen bu sınırın belirsizleşmesine neden olan bazı dinamikler sözkonusudur. Bir tarafta program geliştirenlerin gittikçe karmaşıklaşan işi nihai olarak bilgisayarın temel becerilerine ve komut setine bağlı olduğundan bu insanlar zaman zaman donanıma belirli yeni beceriler eklenmesini isterler, yanı donanım tasarımına bulaşırlar. Öteki tarafta donanımın gittikçe karmaşıklaşması bilgisayarı oluşturan birçok parçanın birbirinden bağımsız, hepsi birer donanım-yazılım kombinasyonu olan alt-sistemler olarak gelişmesine yolaçmıştır (bu ayrışmanın genel özelliklerine ve alt-sistemlerden bazılarına bölüm 4'de bakacağız), yani donanımcılar da yazılım tasarımına bulaşırlar. Bir yandan bu iki disiplin sınır çizgisinde kalan bazı alt-disiplinlerin sahipliği üzerine kavgaya tutuşurken, diğer yandan bilgisayar alanı tüm disiplinlerin evriminde olduğu gibi, ama çok daha hızlı bir şekilde, yeni alt-disiplinlere bölünüyor çünkü hayatın her alanına hızla yayılıyor. Bu durum bir yanıyla bu alana yönelmek isteyen gençlere çok renkli kariyer seçenekleri sunuyor, bir yandan da bu seçenekleri ayırt etmeyi güçleştiriyor. Bilgisayarın farklı amaçlarla kullanımı konusunu bölüm 6'te açacağız. Bilgisayarın genel olarak toplum ile etkileşimini, ve özel olarak bilgisayar teknolojisinin ekonomik açıdan değerlendirmesini ise bölüm 8'de yapacağız.

1.4 Başka türlü bir zeka

Okurların bir kısmı 'yapay zeka' terimini duymuş olmalı. Bir kısım bilgisayar programına atfedilen bu nitelik bilgisayarın belirli bir alanda insan zekasıyla yarışabilen başarı göstermesiyle özdeşleştiriliyor. IBM tarafından inşa edilen özel bir donanım ve yazılım

sistemi olan DeepBlue[3] bunun popüler örneklerinden biri çünkü 1997'de Dünya şampiyonu Garry Kasparov ile yaptığı satranç karşılaşmasını kazanmayı başarmıştı. Yani önemli bir zeka oyunu kabul edilen satranç konusunda bu bilgisayr bütün insanlardan daha başarılı!

Yıllardır üniversitede yapay zeka dersleri vermeme rağmen burada eleştirel bazı yorumlar sunacağım. Bunun temel sebebi bilgisayara atfedilen zeka ile bizim zekamız arasındaki bazı önemli farklılıkları vurgulamak ve bilgisayardan neleri başarmasını bekleyip neleri beklememek gerektiği konusuna biraz açıklık getirmek.

Kökenleri Descartes'a uzanan modernizm felsefesi esas itibarıyle rasyonelliği yani akıl yürütmeyi mekanik bir işlev olarak algılar. Aklın açıklanamaz, bedensel varlıktan ayrı, insana özgü ve tanrı vergisi bir özellik olduğu düşüncesinine karşı onun beynin işleyişinden kaynaklanan bir özellik olduğunu vurgular. Bu duruşun doğal bir sonucu beynin işleyişinin bir makine ile taklit edilmesiyle insanınkine benzer bir aklın da bu makine tarafından sergileneceği inancıdır. Bilgisayar teknolojisinin önde gelen mimarı saydığımız Alan M. Turing'in ilk çalışmalarından biri de bilgisayarın zekiliğinin nasıl tespit edileceği üzerine kurguladığı Turing testi idi (Turing, 1950). Turing testi bir hakemin görmeksizin, bir klavye ve monitör aracılığıyla sorguya çektiği iki varlıktan hangisinin insan hangisinin bilgisayar olduğu hakkında hüküm vermesine, ve bilgisayarın hakemi insan olduğu konusunda kandırıp kandıramayacağının ölçülmesine dayanır. Günümüzde temeli bu teste dayanan yarışmalar her yıl yapılıyor ve bu alanda oldukça prestijli etkinlikler.

Mühendislik eğitimi almış biri olarak yapay zeka kavramıyla temel bir problemim yok. Bana ukalalık etmedikleri sürece bu tür makinelerden yararlanmak ta hoşuma giderdi. Ancak yapay zeka araştırmalarının derinde yatan bir problemi var: taklit etmeye çalıştığı insan beyninin işleyişi yeni çözümleniyor, ve ortaya çıktığı kadarıyla yapay zekacıların düşündüğünden oldukça farklı (Crick, 1990; Newell, 1982). Beynimiz makine metaforunun altıda yatan mekanik ve hiyerarşik bir organizasyondan ziyade organik ve hiyerarşik olmayan bir organizasyona sahip. Yapay zeka şemsiyesi altında yapılan çalışmaların ancak bir bölümü organizma metaforuna sadık, ve oldukça da başarılı çalışmalar. Daha da ciddi bir sorun ise makinenin değişimi tolere edememesi. Bizler alışık olduğumuz yöntemler sonuç vermeyince yeni çözümler buluyoruz,

oysa bilgisayar programlamanın geneline ve bunun devamı olarak yapay zeka programcılığı pratiğine sirayet eden yaklaşımlar bunu sağlayamıyor. Buna karşılık örneğin son yıllarda yaygınlaşan dosya paylaşım ağları gibi sistemler hataları göğüsleyebilen, basit bileşenlerin etkileşimiyle bütüne özgü becerilerin ortaya çıktığı ilginç deneyimler sağladı, ve bunun sonuçlarından etkilenen bazı yeni programlama yaklaşımları son derece umut vadediyor. Yapay zeka konusuna ve bu yeni açılımlara özet olarak bölüm 8'de değineceğiz. Bu ve başka konuların altyapısında yeralan bilgisayar ağları ve İnternet iletişimine ise bölüm 4 değiniliyor.

1.5 Okuma notları

Farklı alan ve uğraşı olan okurlar için bu kitaptaki malzemeyi incelemenin çok farklı yolları olacağından eminim. Ayrıca bölümlerin bir kısmı konuya aşina olmayan okurları farklı derecelerde zorlayacaktır. Görece zor ve teknik detayı bol bölümleri ise bu detaylar akılda kalmasa da okurda konuya aşinalık hissi bırakacak şekilde yazmaya çalıştım. Genel olarak kitabın bölümlerini olabildiğince bağımsız ve tek başına okunabilir bölümler olarak kurgulamaya gayret ettim. Bunun sonucu kimi tekrarlar var, buna karşılık ilginizi çeken bölüme atlayıp okumaya imkan veriyor.

Bölüm 2 ve 4 bilgisayar donanımının temel işleyişini ve cihazın iç düzenini anlatıyor ve görece daha teknik bölümler. Bu konuyu merak etmeyen okurların yine de teknolojinin tarihsel fonundan ve modernizmle içiçe gelişiminden bahseden bölüm 2.1'ye bakmalarını öneririm. Buna karşılık dijital içerikten bahseden bölüm 3 sık karşılaşılan veri türlerinin içyüzünü anlatıyor ve görece daha az teknik. Bölüm 6 ise çoğumuzun bilgisayarla sıkça yaptığı metin yazımı veya tablo kullanımı gibi işlerle ilgili, ve yaygın yanlışlara dair biraz temelden bir sorgulama ve açıklama getirmeye çalışıyor.

Programlamayla ilgili bölüm 7 bağımsız olarak okunabilir.

Bilgisayarın birey ve toplumla etkileşim içerisinde nasıl evrimleştiğini, bunun geleceğine dair bazı öngörüleri, ve bu teknolojinin ekonomik yönlerini ele alan bölüm 8 ise diğerlerinden en ayrı, ve tek başına okunabilir bölüm oldu.

Bölüm 2

Dijital Bilgisayarlar: Matematikle Elektriğin Buluşması

2.1 Otomatikleştirmenin kısa tarihçesi

Mısır, Sümer gibi gelişmiş uygarlıkların ortaya çıkışından beri insanlık karmaşık matematik hesaplamalara ihtiyaç duydu. MÖ 3. binlere dayanan tarihsel kayıtlar, Ahmes papirüsü gibi arkeolojik buluntular (Cajori, 1991) bu ihtiyaçlrı karşılamaya yönelik matematiğin gelişimini ortaya koyar. Tarım arazilerinin bölünmesi ve mahsul miktarının önceden kestirilebilmesi için üçgen, yamuk tarım alanlarının yüzölçümü hesaplaması amacıyla geometrik formüller, çarpma bölme tabloları, mimari yapılar için hacim ağırlık formülleri vb. bunlar arasındadır. Ancak antik Yunan dönemindeki bilim insanları bu tür problemlere pratik uygulamasının ötesinde, ezberci değil keşfe yönelik bir ilgi göstermişler, ve özellikle sistematik bir yaklaşım geliştirmişlerdir. Bu sistematikliğin sonucu olarak Pisagor ve Arşimed gibi matematikçilerin açıklamaları çözüm yollarını ve gerekçelerini adım adım tarif eder (analiz), ki böylece belirli bir problemi çözen formül yerine benzer problemlerin de çözümü çıkarsanabilsin (sentez). Örneğin Aristo'nun insan zihninin nasıl çalıştığına, sorunları nasıl çözdüğüne dair çözümlemesi çağdaş bilim dilini oldukça

andırır, ve günümüzde yapay zeka uygulamalarında kullanılan bazı yöntemlerle bire bir örtüşür. Eski Mısır papirüslerindeki belirsiz açıklama diline karşın, Euclid ve Archimedes gibi Yunan döneminde yetişen Akdenizli bilim insanlarının yazdıkları, çoğu temel kavramdan ve gelişmiş bir sembolizmden, hatta sayıları temsil edecek doğru dürüst bir sayı sisteminden bile yoksun olmalarına rağmen (Ifrah, 1995; Cajori, 1991) son derece açık ve sistematiktir. Bu yaklaşım hem Arap aydınlanması döneminde (örn. El-Harezmi-8.yy Bağdat, ibn-Haldun-13.yy Kuzey Afrika) hem de onu takip eden Avrupa aydınlanması döneminde (örn. Fibonacci-12.yy İtalya) bilimin canlanmasına ve ilerlemesine katkıda bulunan insanlarca yadedilmiş ve benimsenmiştir.

Analitik düşünce matematik ve makineleşme için bu kadar önemliyken, bilgisayarın ismini aldığı kavramın –İngilizce Compute veya Latince Compotos– çıkışı matematik konusunda oldukça cahil sayılabilecek Romalılar dönemine rastlar. Daha öncesinden beri Latin dillerinde 'Calculus' sözcüğü yaygın olarak kullanılıyordu, ki çakıl taşlarıyla saymak anlamına gelir. Compotos sözcüğü ise 5. yüzyıl Roma-Bizans döneminden başlayarak bir kısım anlam kaymalarıyla ortaya çıkmıştır. Aloisio'ya göre (2004) bu sözcük Hristiyanlıkta İsa'nın dirildiği gün olduğuna inanılan Paskalya tarihinin hesaplanması problemine atfen kullanılmaktaydı. Meşhur İznik konsülünde (4.yy) Paskalya'nın Yahudi'lerin hamursuz bayramını takip eden ilk dolunaydan sonraki ilk Pazar günü olması kararlaştırılmıştı. Böylelikle Paskalya gününün bilinmesi Yahudiler güneş takvimini kullandığı, ve işin içinde bir de dolunay olduğu için birbiriyle uyumsuz güneş ve ay takvimlerini dikkate alarak karmaşık hesaplamalar gerektiren bir iş oluverdi. Temel aritmetik bir formüle indirgenemeyen, *aşamalı* işlemler gerektiren bu hesaplama için compotos sözcüğü kullanıldı. Paskalya probleminin çözümüne önemli katkıda bulunan Suriye'li matematikçi Diopanthes (6.yy) aynı zamanda matematik problemlerin ve çözümlerinin tarifinde değişken sembollerini ilk kullananlardan oldu (Cajori, 1991). 19.yy sonlarına gelindiğinde Computer sözcüğü karmaşık fizik, balistik, vb., hesap işlemlerini kalem ve kağıt ile yapan (çoğu kadın) insanlar için kullanılıyordu. Bilgisayarın ortaya çıkışıyla bu sözcük makineler için kullanımaya başlandı. Eski calculus sözcüğü ise halen hesap makinesi için (İngilizce calculator), tek aşamalı temel aritmetik işlemlere atfen kullanılıyor.

Bilgisayarın kelime kökenleri Roma dönemine dayanıyorsa da, kuramsal altyapısında yeralan önemli bir kavram, algoritma kavramı, Arap matematiğiyle ilişkilendirilir. Bu sözcük İran kökenli Bağdatlı bilim insanı El-Harezmi'nin isminden gelir. El-Harezmi'nin, başlığı aynı zamanda Cebir (İngilizce Algebra, Arapça el-cebr: indirgeme) sözcüğünün de kökeni olan, bir eseri denklemlerin nasıl birleştirilip sonra da çözüleceğine dair sistematik yöntemler sunar. Yüzyıllar boyunda kullanılan bu yöntemler halen ilköğretim düzeyi matematik derslerinde neredeyse hiç değişmeden öğretiliyor, örneğin $2x + 1 = y$ ve $x = 2y$ denklem sisteminin çözümü gibi. Elbette Harezmi'den öncesinden kalma algoritmalar vardır (halen kullandığımız Ğklid'in en büyük ortak bölen bulma yöntemi gibi), ancak Harezmi'nin hem matematik becerisi, hem de sistematik adımlara ve genelleştirmeye dayalı yaklaşımındaki berraklık ve kullanışlılık çok aşamalı işlemler içeren matematik yöntemlerin onun ismine atfen algoritma olarak adlandırılmasına yol açmıştır. Bugün bir bilgisayar programı içinde anlam belirsizliği olmayan bir algoritmadan, daha doğrusu içiçe geçmiş algoritmalardan ibarettir diyebiliriz.

Matematiksel çözümlerin hem tarif edilme yöntemi hem de bu yöntemlerin kesinliği bir yandan böyle kıtadan kıtaya, uygarlıktan uygarlığa aktarılıp üstüste birikerek gelişirken Avrupda'da herşey farklı bir ivme kazanır. Arap ve Bizans arşivlerine büyük bir ilgiyle sarılan ortaçağ Avrupası matematikçilerinin yanısıra tüm toplum bir felsefi devrim yaşamaktadır. Descartes'ın manifestosuyla ivme kazanan bu felsefe ilahi ve manevi olanı bir tarafa koyup maddi dünyaya ve insanın fiziki varlığına odaklanır. Bir yandan Guy de Mouppassant'ın hikayelerinde abartılı ifadesini bulan makinelerin gelişimi ve onlara duyulan tutku yükselirken, modernist düşünce insan aklının da kuralları belirli bir makine gibi çalıştığını, ulvi bir yanı olmadığını, ve bu kurallar ortaya konulursa bir makine aracılığıyla taklit edilebileceğini işaret eder. Fransa'da Pascal'in, ve Almanya'da Leibniz'in yaptığı ilk hesap makinelerinin (17.yy) başarısı da bu hedefin ulaşılabilirliği inancını besler.

Ancak hem matematiğin hem makinelerin belirli bir düzeye ulaşması ve çarpıcı bir birleşme yaşaması çok zaman ve emek gerektirecektir. 19.yy'da Charles Babbage'ın (ölümü 18 Ekim 1871) gerçekleştirmesini tam olarak başaramadığı 'analitik makine' her türlü makinenin yapılmaya başladığı bir dönemde matematiksel bir becerinin makinelere kazandırılması yolunda ilk denemeler-

dendir. Babbage'ın o günün metal işleme teknolojisinin yetersiz hassasiyeti ile başarısız olan makine tasarımlarından bazıları bir yüzyılı aşkın zaman sonra yapılmış ve başarıyla çalışmıştır. Babbage o dönemde varolan ve toplama-çıkarma yapabilen makinelerden çok üstün bir makine öngörmüştü. Tasarımında kısmen o dönemde tekstilde kullanılan ve desenlerin delikli kartlara işlenerek otomatik olarak örüldüğü Jacquard dokuma tezgahı teknolojisini kullanmıştı (Campbell-Kelly and Aspray, 2004). Böylece Babbage'ın planlarına göre hesaplamayı yönlendiren 'programlar' makineye bu delikli kartlar ile yüklenecekti, aynı yüzyıl sonra yapılan ilk bilgisayarlarda gerçekten olacağı gibi. Her ne kadar büyük projesi sonunda başarısız olsa da Babbage'ın projesine tutkuyla inanan ve o dönemde matematikle uğraştığı bilinen nadir kadınlardan olan Ada Lovelace bu hiç gerçekleşmeyen projeye önemli destek verir. Projenin hikayesini, ve Babbage'ın tasarımını detaylı olarak kaleme alır. Sadece 37 yaşında ölen ve gelecekte bilgisayarlarla müzik bile yapılacağını öngörecek kadar parlak bir dehaya sahip olan bu zeki kadını birçokları tarihte ilk bilgisayar programcısı olarak gösteriyor. Tarihsel bilgiler Ada Lovelace'in kaleme aldığı 'programlar'ın hemen hepsinin Babbage'ın orijinallerine dayandığına işaret ediyor (Campbell-Kelly and Aspray, 2004). Yine de Lovelace'ın bu makineyi anlatımındaki pırıltı gelecek nesillere ilham verecekti: " 'analitik makine aynı Jacquard dokuma tezgahının çiçek desenlerini ördüğü gibi cebirsel desenleri örecek"(Campbell-Kelly and Aspray, 2004, sf48).

Bir yandan bu tür denemelerle mekanik ilerlerken bir yandan da matematik mekanikleşmektedir. Bir süredir devam eden, bölük pörçük gelişen matematiği disipline etme ve sağlam temellere oturtma çabaları arasında bizi ilgilendiren kayda değer bir tanesi Whitehead ve Russell'ın Principia Mathematica (Matematiğin Prensipleri) adı altında 1910-1913 arasında üç cilt olarak yayınladıkları kitap. Bu kitap ele alınıp okunacak bir kitap olmaktan ziyade kesin tanımlanmış çıkarım kuralları ve temel mantık aksiyomlarından yola çıkarak matematiği –neredeyse bir makine tasarımı gibi– günlük kullandığımız belirsizlik içerebilen konuşma dili yerine sadece başta tanımlanan sembol ve çıkarım kurallarını kullanarak ifade ediyordu. Öyle ki bunu bir makine bile okuyabilir. Makine ile insan aklı arasında belirsizleşen sınıra atfen Russell'ın ortaya attığı bir paradoks ilginçtir: bir kutunun içindeki insan veya makineye Çince semboller ve bu sembollere nasıl cevap

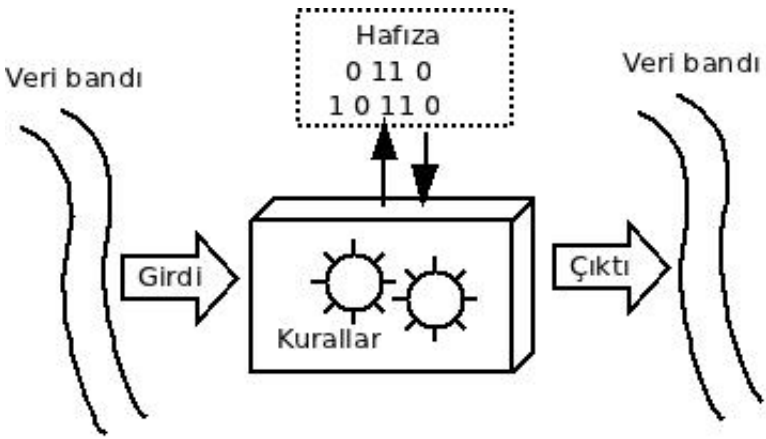
verileceğini anlatan bir kurallar kitabı versek, bu makinenin Çince sorulara vereceği makul cevaplara istinaden onun Çince anladığını söyleyebilir miyiz? Esasen bu soru insanın çevresini kavrayışına atfen de sorulabilir ve cevaplaması kolay değil. Ancak kitbın yazılışından yaklaşık elli yıl sonra bir bilgisayar programının Russell ve Whitehead'ın kitapta verdikleri bir matematiksel ispatı kendi kendine bulduğunu (ve Russell'ın pek memnun olduğunu) söylemekle yetinelim.

2.1.1 Turing'in katkısı ve iki farklı yaklaşım

Russell ve onun çağdaşı Alonso Church'un bu dönemdeki (1930lar) önemli bir katkısı fonksiyon kavramının matematiksel açıdan farklı bir zemine oturtulması olmuştur[4]. Gelişigüzel tanımlanan fonksiyon kavramını Russell'ın çalışmasını tamamlarcasına sağlam bir zeminde tanımlar. Öyle ki fonksiyonların aritmetik ifadelerde kullanımı 'mekanik' bir netlik kazanır. Gerçekten de yıllar sonra ilk programlama dillerini tasarlayanlar bu dillerin gramer kurallarını, bilgisayara bir fonksiyonun nasıl anlatılacağına dair sözdizimi kurallarını Church'un çalışmalarına dayandıracaktır. Church gibi bilim insanlarının katkısıyla oluşan matematiksel kesinliğin bir sonucu neyin hesaplanabilir olduğu ve bu hesabın ne kadar zor olduğunun nesnel olarak tartışılabilir hale gelmesiydi. Bu kesinlik sayesindedir ki yakın bir gelecekte kendisine verilen matematik yöntemi hatasızca uygulayan ve çalışma süresi öngörülebilir makineler yapılabilecekti.

Makinelerin rafineleştiği ve matematiğin mekanikleştiği bu dönemde Alonso Church'ün öğrencisi olan Alan M. Turing bu iki yolu birbirine sağlamca bağlayan kişi olarak ortaya çıkar (Resim 2.1. Russell ve Church'un formel mantığa dayalı kuramlarını ve yaklaşımlarını gayet iyi kavrayan Turing bunu makine metaforuyla başarılı bir şekilde sentezler. Daha sonra başkaları tarafından Turing makinesi ismi verilen tasarımı oldukça basit ilkelere dayanıyordu Turing (1936). Makine (şekil 2.2 bir kez çalıştırıldığında bir banttan bilgileri okumaya başlar, önceden belirlenmiş *kurallara* ve o anki *durumuna* (yani belleğin içeriğine) göre bir çıktı yazar ve aynı zamanda durumunu değiştirir. Böylece 'bitti' durumuna ulaşana kadar girdi okuma-durum değiştirme-çıktı yazma döngüsü devam eder. Girdi ve durum'un olası her kombinasyonuna karşılık gelen kuralları bir tablo halinde ifade ederseniz makineyi de tanımlamış olursunuz.

Şekil 2.1: Bilgisayar'ın gelişmesine önemli katkı yapan Alan M. Turing aynı zamanda başarılı bir maraton sporcusuydu.



Şekil 2.2: Turing Makinesi

Turing'in tasarımı o zamanın koşullarında gerçekleştirilebilir bir makine izlenimi bırakması açısından ilginçtir. Ancak daha da önemlisi bu tasarımın Church'un formel matematik kuramı ile eşdeğer olduğunun ispatlanmasıydı. Yani Church'un kurduğu matematiksel altyapı ile ifade edilen her çözüm Turing makinesine de uygulanabilirdi.

Turing makinesini bir yandan sağlam matematik temellere dayandırırken bir yandan da basit bir makine metaforu kullanarak günün teknolojisi ile gerçekleştirilebilecek bir senteze imza atmış olur. Örneğin girdi ev çıktı verileri kağıt kartlar üzerinde delik olma olmama durumu ile ifade edilebilirdi. Yine o günkü teknoloji ile hafızanın ve kuralların elektromekanik cihazlarla yapılması mümkün gözükmüştür. Teorisiyle dikkatleri çeken Turing'in ikinci Dünya savaşı sırasında Alman ordularının şifrelerini kırmak için yürütülen bir proje ekibinde yeralması bu tasarımın gerçekleşmesinde ve gelişiminde etkili olur. Almanların şifresi gerçekten de çözülür. Her ne kadar bilgisayar bir teknoloji olarak ikinci Dünya savaşı sonrası ABD'de gelişmeye devam etmiş ve Amerikan ekolü teknoloji tarihi kitaplarında Turing'in katkısı gölgelenmiş te olsa bugün pekçok bilgisayar bilimci disiplinin dönüm noktası olarak Alan Turing'i kabul etmektedir. Amerikada basılan bilgisayar tarihi eserlerinde konunun sunum şekli, neredeyse fizik ve astronominin gelişiminde Einstein ve Galileo'dan çok Hubble teleskobunun katkısı olduğunu söylemeye benziyor.

Turing 1954'te eşcinselliği yüzünden zorla tedaviye sokulduğu için intihar ettiğinde henüz genç yaştaydı ve canlıların gelişmesinin matematik temelleri üzerine çalışıyordu. Zamansız ölümü 20.yy bilimi için ciddi bir kayıp olmuştur.

Hem Turing hem de ondan sonra gelen bilgisayar bilimciler Turing makinesi ile Church'un daha geleneksel matematiğe bağlanan kuramı arasındaki eşdeğerliği net olarak gösterdiler. Bu sayededir ki günümüzde bilgisayar programlarken matematik yöntemleri alıştığımız sembollerle, kendi düşünce akışımıza tekabül eden algoritmalarla ve rahat anlaşılabilir doğruğulu kontrol edilebilir bir tarzda ifade edebiliyoruz. Bilgisayar programlama dilleri bir programcının soyutlamaya dayalı ifade dilini makinenin mekanik çalışmaya dayalı diline dönüştürüyor. Ancak Turing bu iki yolu başarıyla birleştirmiş te olsa programcıların önemli bir kısmının makineyi algılayışı ve program yazma tarzı Church'un tarzından çok Turing makinesi metaforunun etkisi altında kalmıştır. Birinci

tarz programlama biri diğerinden yararlanan birsürü işlemin tarif edilmesi, ve sonra seçilen bir tanesinin çalıştırılmasıyla bir silsile işlemin gerçekleşmesine dayanır ve işlevsel (functional) veya bildirimsel (declarative) programlama diye anılır. Bu tarzda yazılan programlarda bir matematik yöntemin programı yöntemin matematik kitaplarındaki yazımına son derece benzer; bilgisayarın hafızasının ve yapacağı işlemlerin planlanması yerine yöntemin küçük parçalara bölünerek ifade edilmesi, ve istenen ifadenin bu parçalar cinsinden yazılmasına dayanır. İkinci tarzda yazılan programlar ise matematiksel yöntemin bilgisayarın hangi hesabı yapıp bellekte ne tutacağını detaylıca tarif eden, adımlarını sıkı sıkıya kontrol etmeye çalışan, buyurgan türdendir. Bize göre bu bu ikinci tarz ancak özel durumlarda, bilgisayarın altyapısını oluşturan ve dolayısıyla makine metaforuna daha yakın düşünmeyi gerektiren durumlarda gereklidir ve temkinli olarak kullanılmalıdır. Ancak her ne kadar programların yazımı ve doğrulanmasını güçleştirse de, matematiksel soyutlama becerisinden çok mekanik tariflere dayandığı için bu tarz çok yaygın. Bilgisayar programcılığı alanında makine metaforunun bu yersiz popülerliğinin temelinde Amerikan tarzı pragmatik yaklaşım yatıyor olsa da temel bilgisayar eğitiminde bu sorunun çözülmesine dair önemli katkılar da yine orada ortaya çıkmaya başladı. Bugün MIT gibi önde gelen Amerikan üniversitelerinde programlama eğitiminde birinci tarzı kullanmaya dayalı müfredatlar geliştirilmekte ve son derece başarılı olmaktadır.

2.1.2 Hesaplanabilirliğin sınırları, yapay zekanın imkansızlığı, ve Evrensel Turing makinesi üzerine

Matematiğin gelişimi içerisinde kurallı ve mekanik tanımlama tekniklerinin ortaya çıkışını ve makine teknolojisi ile buluşmasının ana hatlarını sığ bir şekilde de olsa vermeye çalıştık. Yukarıdaki tarihsel çerçeve bu bölümün kalan kısmını okumak için yeterli bir fon oluşturuyor, ancak modernist felsefeye de dokunan bir konuya daha değinelim. Russell ve Church'un, Einstein ve fiziğin hızlı gelişme dönemine de denk düşen çalışmaları modernizmin maddi dünyaya odaklı felsefesi için bir zafer gibi görünüyordu. Madem fizik bilimi maddenin nasıl işlediğinin esaslarını ortaya çıkarmıştı, ve madem hesap bilimi ve bu hesapları yapacak makineler yapılıyordu, o zaman herşey hesaplanabilir ve öngörülebilirdi. Ancak Gödel isimli bir genç bir matematikçi bunu

temelden yanlışlar, hem de Russell'ın çizdiği çerçevede kalarak. Bu konu analitik felsefe açısından uzun yıllar sürecek bir dizi tartışmaya sebep olmuştur. Bunlardan önemli bir tanesi insan zekasının da matematiğin tarif ettiği türden bir hesaplamayla taklit edilip edilemeyeceği konusudur. Buradaki tartışma esinleme ve sezgi gibi insan aklına özgü işlevlerin (bunlar mutluluk veya üzüntü gibi duygusal durumlarla karşılaştırılmamalıdır, buradaki bahsi örneğin fiziki dünyaya dair bir problemi çözerken gösterdiğimiz yaratıcılıkla sınırlıyoruz) bilgisayarla taklit edilmesinin çok zor olduğu değil, imkansız olduğu konusundadır. Örneğin Roger Penrose (1989) konuyu fizikteki ilerlemeler ışığında inceleyerek insan aklının algoritmalara indirgenemeyeceğini savunur. Henüz bu konuda somut bir alternatif bulamamış olsak ta yapay zeka alanı bu esastan başarısızlığını kabul etmiş, yavaş yavaş ta olsa beyin gibi organik sistemlerin çalışmasını daha fazla andıran yöntemlere yönelmeye başlamıştır.

Felsefi açıdan yarattığı bu hayalkırıklığına rağmen Gödel'in başlattığı tartışmanın matematik açısından katkısı hesaplanabilirliğin sınırlarının, ve dolayısıyla hesaplamanın tanımındaki son belirsizliklerin de ortadan kalkması olmuştur. Gödel'in katkısıyla beraber Turing makinesi ile Church'un kuramı arasında net bir 'eşdeğerlik'ten sözedilebilmiştir. Bundan sonra ortaya çıkacak olan bilgisayar tasarımları da bu temel modellerle eşdeğer oldukları ölçüde uygulanabilir olmuşlardır.

Bu dönemde dönen tartışmaların önemli bir sonucu daha var. Bir şeyin hesaplanabilip hesaplanamayacağı, yani o hesabı yapmak için tasarlanan Turing makinesinin eninde sonunda 'bitti' durumuna ulaşıp ulaşmayacağı tartışılırken Turing ilginç bir kavram ortaya atar: Evrensel Turing Makinesi. Dikkat edilirse Turing makinesi belli bir kurallar kümesi ile çalılıyordu, yani belirlenmiş bir programı vardı ve tek bir işi yapıyordu. Turing'in Evrensel Makine adıyla sözettiği farazi makinesinin programı ise verilen herhangi bir Turing makinesini taklit etmekti! Turing'in esasen bir hesaplamanın sonlanıp sonlanmayacağını tartışmak için ortaya attığı bu farazi makine günümüzde istenilen programı yükleyi çalıştıran bilgisayar tasarımının ilham kaynağı olacaktı.

2.2 Gerçekliğin basit ifadesi ve basit elektronik bileşenlerle işlenmesi

2.2.1 İkilik sistemin gücü

Bilgisayarın yapımında kullanılan temel malzemeler ve bileşenler esas itibarıyle basittir. Çoğumuzun lise fizik derslerinden hatırlayacağı dirençler, elektrik akımı, ve bir de transistör adı verilen bir elektrik malzemesi kullanılır. Ancak bu basit bileşenlerden milyonlarcasının, temel bazı birleştirme yöntemlerini yineleyerek kullanılmasıyla karmaşık ve becerikli bir cihaz ortaya çıkar. Öyle ki bu cihaz sayıları toplama, çarpma, karşılaştırma gibi komutları yerine getirebilir ve yapılan karşılaştırmaların neticesine göre bizim belirlediğimiz şu yada bu komuta atlayarak istediğimiz işlemleri yerine getirir.

Bu karmaşık cihazın esasını oluşturan elektronik bileşenler sadece iki şeyi ayırtedebilir: 1 ve 0, daha doğrusu elektrik voltajının yüksek veya düşük olması durumu. Günümüz bilgisayarları ayırtetmenin bu en basit haline teferruatlı bir derinlik katarak inşa edilmiştir. Her türlü büyüklük (tamsayı, gerçek ve rasyonel sayılar), renk (renk tonu ve parlaklığının sayıya dökülmüş hali), ses (ses dalgalarının yüksekliği), vb., yani gerçek dünyanın bilimum temsili sayılara dökülebildiği takdirde bu gerçeklik sadece 1 ve 0 kullanılarak ifade edilebilir. Bizim alışkın olduğumuz ondalık sayılar kullanmaktan çok az farkı olan bu ifade şekline ikilik sayı sistemi, ya da kısaca ikilik sistem diyeceğiz.

Sayıların ikilik sisteme çevriminin nasıl yapıldığına ve bu sistemin başka özelliklerine biraz ileride tekrar döneceğiz. İkilik sistemin kullanımı bilgisayar yapımındaki gerekli elektronik için çok kolaylaştırıcı olmuştur. Örneğin ondalık sistem kullanılsaydı bu elektronik devrelerin on farklı voltaj seviyesini ayırt etmesi gerekecekti. Böyle bir durum hem son derece hataya açık olurdu (örneğin sinyallerdeki parazit) hem de yukarıda bahsettiğimiz üzere esasa bir katkısı olmazdı.

2.2.2 Transistör ve mantık köprüleri

Bu bölümde öncelikle makinenin temel taşları olan bazı bileşenlere bakacağız. Bu elektronik bileşenlerin özelliği herbiri sadece 1 veya 0 (yüksek veya düşük voltaj) değerini alabilen bir veya birden fazla girdiyi alıp bunların karşılıklı durumuna göre şu veya bu sonucu vermeleridir, ki burada sonucun kendisi de 1 veya 0

Şekil 2.3: Otopark bariyeri Kontrolü

değerlerinden biridir. Bu ayırtetme ve karşılaştırmanın sonucuna göre alternatif sonuçlardan birine varma becerisi makinenin temel taşlarını oluşturur. Bu bölümde basit değerleri işleyen devrelerden bahsedeceğiz. Daha sonra olarak bu devrelerin çok haneli sayıları nasıl işlediğine bakacağız. Son olarak ta tek bir işi yapan değil istenilen programı yükleyip çalıştıran bir makinenin nasıl yapılabildiğinden kabaca bahsedeceğiz. Böyle hızlı bir bakışı verebilmek için tamsayılar dışında bilgisayar ortamında kullandığımız faklı bilgilerin (kesirli sayılar, yazılar, ses, görüntü, vb.) nasıl olup ta 1 ve 0'lara indirgendiğinin detaylarını bir sonraki bölüme bıraktık. Ayrıca gerçek matematik problemlerinin çözülmesini sağlayan programlama dillerinin nasıl çalıştığını, bu bölümde tarif ettiğimiz makineyle nasıl uyumlu olduğu konusunu bölüm 7'da ele alacağız.

Bilgisayarın temel karşılaştırma ve karar verme işlemlerini nasıl yaptığını incelemek için bir örnekten yola çıkalım: bir otopark bariyerinin kontrolü. Şekil 2.3. Tek yünlü geçiş için tasarlanan bu bariyein önünde ve arkasında araç geldiğinde algılayan metal dedektörleri var, ve bunlar bir araç algılağından yüksek voltaj, aksi takdirde düşük voltaj veriyor. Bariyer kontrol motoru da yüksek voltaj verilince açılıyor, düşük verilince kapanıyor. Kontrol sistemi önden araç geldiğinde bariyeri açar, araç arkaya geçtiğinde ise artık bariyeri kapatabilir. Ancak aynı anda yeni bir araç gelmesi durumunda bariyer onun üstüne kapanmamalı.

Bu sistemin girdileri *ön* ve *arka* adını vereceğimiz dedektörler, çıktısı ise *bariyer* adını vereceğimiz elektrik sinyali olacak, ki bunların hepsi yüksek veya düşük voltaj, yani 1 veya 0 değerlerinden birini alacak. Problemi bu şekilde tanımladığımızda sistemin takip edeceği program aşağı yukarı ortaya çıkmıştır. Bunu basitleştirilmiş biçimde yazarsak:

```
1  eğer ön=1 ise
2     bariyer --> 1
3  aksi takdirde
4     eğer arka=1 ise
5        bariyer --> 0
6     aksi takdirde
7        bariyer --> bariyer
8
```

Bu gerçek bir bilgisayar programı değil, ama günlük dilin biraz sistematik kullanıldığı, sözde (İng. pseudo) bir program. Bu program bize sistemle ilgili bir unsuru da gösteriyor: son satırda görüldüğü gibi bariyerin açık mı kapalı mı olduğunu hatırlamak gerekiyor, yani sistemin bir hafızası olmalı. Çünkü araç ortadayken, ne ön ne de arka dedektörden sinyal geliyorken bariyeri olduğu gibi tutmamız gerekiyor. Sırf sistemi daha detaylı düşünmek adına bütün verilerin olası değerlerini bir tabloya dökelim. Bu döküm tablo 2.1'de gösterilmiştir. Böyle basit bir sistem için bile insanın gözünü yoran kalabalık bir tablo çıktı ortaya. Sistemin toplam üç girdisi (ön, arka, ve bariyerin şu anki durumu) var ve bunların herbiri 1 veya 0 şeklinde iki olası değer alabileceğinden toplam 2x2x2=8 farklı olası durum var. Her ne kadar tablonun son dört satırında, önde araç olması durumunda, hep aynı sonuç (bariyeri açma) ortaya çıkıyorsa da olasılıkları topyekün değerlendirmek gerekiyordu.

Matematiğin mantık dalı, ve özellikle bu türden iki değerli önermeleri içeren Boole mantığı denilen dalı, bu tür tabloların oluşturulması veya indirgenmesi için bize araçlar sağlar. Burada bu araçların neler olduğuna girmeyeceğiz, ancak bizim örneğimizde ilk bakışta işaret edebileceğim bariz bir durum var; bariyer sadece iki durumda açılıyor: ya önde araç varken, yada ön ve arka boş ama bariyer zaten açık iken. Bu gözlemi kullanarak sistemimizin oldukça basitleştirilmiş bir halini ortaya koyabiliriz:

(ön VEYA ((değil(arka) VE değil(ön) VE bariyer))) $\Rightarrow$ bariyeri aç

Burada 1 ve 0 (doğru ve yanlış, yüksek ve düşük voltaj) değerlerini alan üç veriyi kullanan bir mantıksal ifade yazdık, ancak yazarken verinin sıfır olmasını istediğimiz durumlarda değil(veri) simgesini kullandık. İfadenin sonucunu bariyere bağlarsak bu mantıksal

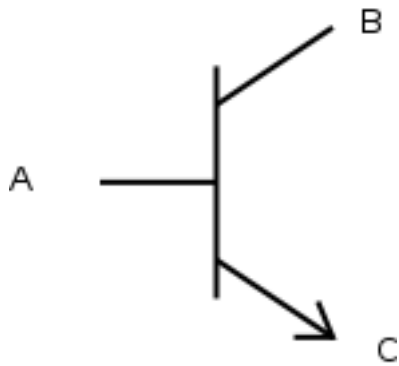
ÖN	ARKA	BAR. Şu an	BAR. **çıktı**	Açıklama
0	0	0	**0**	hiç bir hareket yok
0	0	1	**1**	algılama yok, bariyer açık, ve öyle kalmalı
0	1	0	**0**	ters yönden araç gelişi, bariyeri açma
0	1	1	**0**	bariyer açıkken, arkada araç, bariyeri kapat
1	0	0	**1**	önde araç, bariyeri aç
1	0	1	**1**	önde araç, bariyeri aç
1	1	0	**1**	önde araç, bariyeri aç
1	1	1	**1**	önde araç, bariyeri aç

Tablo 2.1: Bariyer kontrolünde olası durumlar

ifadenin doğru olduğu hallerde bariyer açılacak. Bu örnekteki indirgemeyi ister sözde programa ister tabloya bakarak yapabiliriz, ve elektronik sistemlerde bunların her ikisi de kabul edilir yaklaşımlardır.

Burada mantıksal ifadeler, doğru/yanlış değeri, değil(veri) ifadesi ile beraber 1 ve 0 sayılarından bahsetmemiz kafanızı karıştırmasın. İkili sistemdeki 0 ve 1 rakamlarıyla mantıksal ifadelerdeki yanlış/doğru değerleri arasında bire bir özdeşlik var. Bu yüzden mantık köprüleri dediğimiz elektronik bileşenler bir yandan bizim bariyer kontrol sistemimiz gibi bir sistemin mantıksal kurallarını temsil etmek için kullanılırken bir yandan da ikilik sayıları işlemek için kullanılıyorlar.

Şimdi örneğimizdeki ifadeye karşılık gelen elektronik devreyi kuracağız, ve bu devrede sistemimizin mantıksal ifadesindeki VE, VEYA, değil(), gibi işlemlere karşılık gelen parçalara bakacağız. Bu parçalar bir, iki, veya daha çok girdi alıp bir tek çıktı üretiyorlar, o çıktı da bir mantıksal işlemin sonucu oluyor. Ancak bunlara bakmadan önce transistör adı verilen parçadan bahsetmeliyiz. Direnç, kapasitör gibi elektrik bileşenlerden farklı olarak transistörün iki değil üç bacağı vardır (şekil 2.4. A bacağına yüksek voltaj verilir ve akım geçişi yaratılırsa bu durum B bacağından C bacağına doğru bununla orantılı bir akım oluşmasına yolaçar. Bu iki elektrik akımının birbiriyle orantılı olma özelliği sayesinde transistörler müzik aletlerinde, örneğin havadaki zayıf radyo sinyallerini veya plağın iğnesinden gelen güçsüz sinyali büyütmek



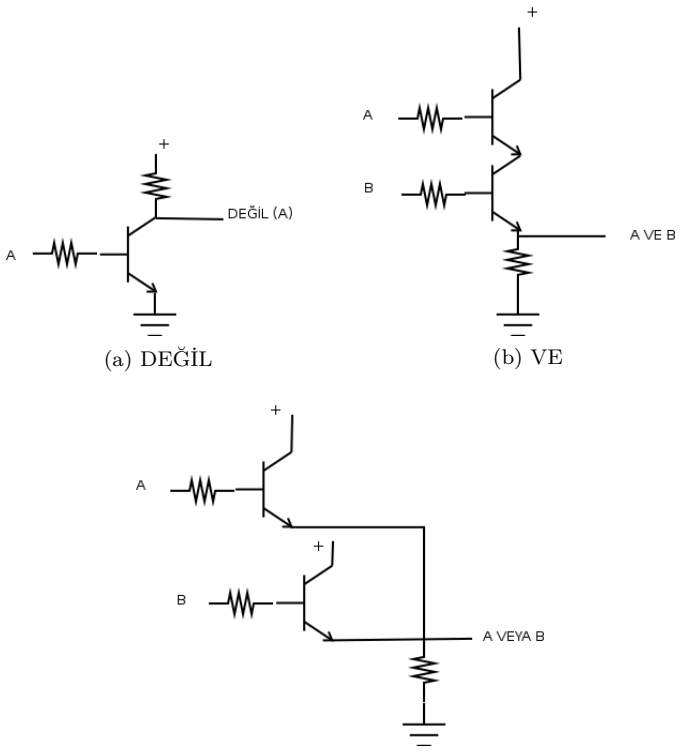
Şekil 2.4: Transistör

için, yani amplifikatör-yükseltici yapımında kullanılırlar.

Bilgisayar devrelerinde ise transistör bir anahtar görevi görür. Burada akımların orantılı olmasından ziyade A bacağında yüksek voltaj olduğunda B$rightarrow$C devresinin açılması önemlidir. Lise fizik derslerindeki elektrik konusunu hatırlamaya hevesli okurlar için bizim kullanacağımız VE, VEYA, ve DEĞİL işlemlerini yapan transistörlü elektrik devrelerinin şeması şekil 2.5'de verilmiştir. Bu devrelerin işleyişine basitçe değinelim. Fizik derslerinden hatırlarsak bir direnç üzerindeki voltaj ve akımın ilişkisi:

Voltaj=Akım X Direnç

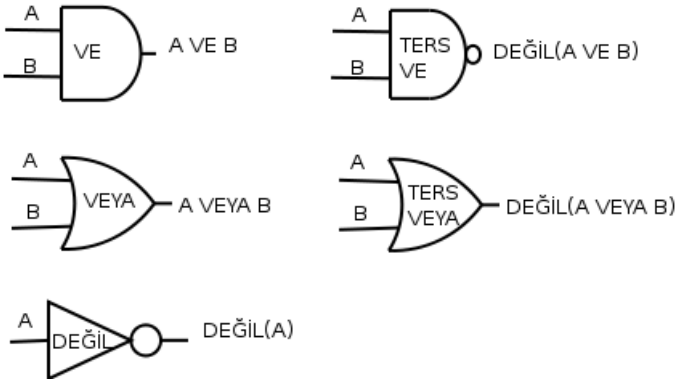
şeklindedir. Yani akım geçerse direnç üzerinde voltaj farkı oluşur. Böylece örneğin DEĞİL devresinde eğer A ucunda voltaj varsa B→C akımı oluşacağından sağ taraftaki voltaj sağdaki resistan üzerinde kalır. Böylece çıkış voltajı düşük, yani A'nın tersi olur. A'da voltaj yoksa transistör açılmayacağından sağda kım oluşmaz ve resistan üzerinde voltaj olmaz, böylece çıkış voltajı yüksek kalır. Yani her iki durumda da çıkış A2nın değili olur. VE köprüsünde ise çıkıştaki (alttaki) direnç üzerinde bir akım oluşması için her iki transistörün de açılması gerekir, ki bu da ancak hem A hem de B'de yüksek voltaj varsa olur. Böylece çıkış sadece her iki girişin de 1 olduğu durumda 1 olur, diğer bütün kombinasyonlarda 0 kalır. VEYA köprüsünde ise transistörler farklı bağlanmıştır, ve hangisi açılırsa açılsın çıkış tarafındaki direncin üzerinden akım geçer ve voltaj oluşur, böylece girişlerden sadece biri bile 1 olsa çıkış 1 olur.



(a) DEĞİL (b) VE

(c) VEYA

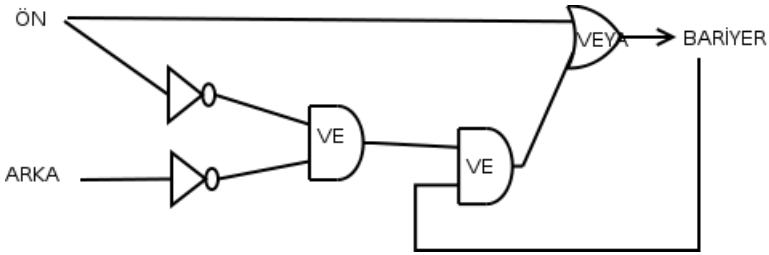
Şekil 2.5: Mantık Köprüleri



Şekil 2.6: Mantık köprüleri için kullanılan semboller

Sürekli bu devreleri çizmek çok kafa karıştırıcı olacak. Üstelik temel devreler, yani VE,VEYA, DEĞİL, ve onların benzeri birkaç başka köprü, hep aynıdır. O yüzden elektronik devrelerde bu transistör-resistans kombinasyonları bazı basit sembollerle gösterilir. Şekil 2.6'nde bunlardan birkaçını görüyorsunuz. Bir VE köprüsünün çıkışının bir DEĞİL işlemine bağlanması gibi sıkça yapılan kombinasyonlar için de bu semboller kullanılıyor ve bunlardan bazılarını da verdik.

Artık bu malzemeleri kullanarak bariyer kontrolü problemimizi çözecek elektronik devreyi çizebiliriz. Bu devre daha önce ortaya çıkardığımız mantıksal ifadenin birebir karşılığı olacak. Ancak mantık köprülerimiz VE VEYA gibi işlemlere aynı anda sadece iki girdi alıyor, o yüzden ifadeyi bazı parantezler ekleyerek eşdeğer biçimde şöyle yazalım: "(ön VEYA (((değil(arka) VE değil(ön)) VE bariyer)))". Buna karşılık gelen bir devreyi şekil 2.7'de görüyorsunuz. Bu devre mantıksal ifade olarak koyduğumuz bariyer kontrol sisteminin grafik bir çiziminden başka birşey değil, ve o yüzden üzerine birşey söylemeye pek gerek yok. Tek farkla ki bu örneğe başladığımızdan beri bahsettiğimiz basit elektriksel mantık köprülerini kullanarak onlardan bir parça daha karmaşık olan problemimizi çözecek bir makine yapabiliyoruz.



Şekil 2.7: Bariyer kontrol problemini çözen mantık devresi

2.2.3 Uygun adım marş: bilgisayar devrelerinde zamanlama

Önceki bölümde bariyer kontrol sisteminin çözümünü ve mantık köprülerinin işleyişini anlatırken birçok basitleştirme yaptık. Elektrik akımı bu devrelerin içinde son derece hızlı hareket eder. Çok sayıda mantık köprüsü içeren karmaşık bir devrenin içinde bu hareketi hesaplamak ve sistemi olumsuz etkileyecek hız dengesizliklerini öngörmek mümkün değildir (bu hızlar devrenin ısınmasıyla bile değişir).

Bu sebepten dolayı mantık köprüleri gibi elektronik bileşenlerin koordinasyonunu sağlamak için bir saatin tiktaklarından yararlanırız. Bu tür bileşenlere bağlanan ek bir kablo tüm bileşenlere aynı anda tiktakları verir. Bu saat sinyallerine bağlı anahtarlar vasıtasıyla artık mantık köprüleri uygun adım hareket ederler, çıktılarını değiştirmeden, ve dolayısıyla önlerindeki köprünün girdilerini değiştirmeden önce tiktakları beklerler. Verilen girdilerin sonuçları elektronik devrenin girdi tarafından çıktı tarafında doğru mantık köprüleri bağlantıları silsilesi içerisinden koordine edilmiş dalgalar halinde ilerler. Bu yaptığımız aslında devrelerin hızını kesmektir, ancak onları sürprizlere yer olmayan, uyumlu ve garantili bir şekilde çalıştırmanın yolu da budur.

Bilgisayar saatinin tiktaklarının sıklığı dikkatle ayarlanır. Onlarca kürekçinin kürek çektiği bir bir teknede olduğu gibi en zayıf kürekçinin de yetişebileceği mümkün olan en yüksek hız hedeflenir. Bir bilgisayar satın aldığınızda size bahsedilen hız ana işlemcinin hızıdır. Örneğin 2.66 GHz (gigahertz) dendiğinde bu işlemcinin saniyede iki milyar altıyüz altmışaltı milyon uygun adım attığı anlaşılır. Son yıllarda çiplerin içindeki transistör boyutunun iyice küçülmesiyle elektriksel özelliklerinden dolayı daha sık adım

attırmak mümkün oluyor. Buna karşılık bir bilgisayar sisteminin topyekün hız performansı ana işlemciye ilaveten teknenin diğer çalışanlarına da bağlıdır. Bellek ve aritmetik modülleri gibi. Günümüzde becerikli ve pratik kullanımlı bir bilgisayar sistemi için gerekli olan bu modüllerin organizasyonu kendi başına bir çalışma alanıdır.

2.2.4 İlk bilgisayarlar ve gölgede kalan kadınlar

Amerikan tarihçelerinde ilk bilgisayar olarak adı sıkça geçen ENIAC[5] (1940ların başı) o zaman için elde bulunan büyücek vakum tüpü transistörlerle yapılmıştı. 17000 vakum tüpü, 70000 resistans içeren devasa alet 63 metrekare yer kaplıyor ve 150 kilovat elektrik yakıyordu. Bu makine verileri okuma ve yazma işlemini delikli kartonlar kullanarak yapabiliyordu, ancak Turing'in evrensel makine kavramı henüz bu tasarımlarda yerini bulmamıştı ve bu yüzden makinenin bazı düğmeleri açıp kapatarak ve kabloları bağlayarak programlanması gerekiyordu.

İlginçtir ki bilgisayarların yapımından önce hesaplama işinde çalışan –ve computer ismi verilen– kadın teknisyenler ENIAC gibi ilk bilgisayarların programlanmasında da ön saflarda olmuşlardır. Aynı DNA'nın keşfi tarihçelerinde fena halde hakkı yenilen Rosalind Franklin gibi (Watson, 2001), programcılığın bu ilk yıllarında katkıda bulunan kadınların imgesi de zamanla erkeklerin oyun sahasına dönüşen bu mesleğin kollektif belleğinden büyük ölçüde yitip gitmiştir[6]. Son yıllarda Grace Hopper ve Ada Lovelace gibi bu alanın gelişimine önemli katkılarda bulunmuş kadınlar anısına yapılan mesleki etkinlik ve konferanslar sayesinde mesleki kültürde yavaş ta olsa bir dönüşüm gerçekleşiyor ve daha fazla kadın bu mesleğe atılmaya teşvik ediliyor.

1940ların başındaki bu başlangıç döneminde farklı ülke ve kurumlarda yapılan projeleri bir sıraya sokmak oldukça güçtğür. Farklı tarihçeler ilk bilgisayar olarak farklı projeleri anarlar (Ceruzzi, 2003). 1941'de Konrad Zuse tarafından Berlin'de yapılan bilgisayar savaş sırasında yokolmuş, mucidi ve fikirleri ise savaş ortamında batı dünyasına ulaşamamıştır. Turing'in de dahil olduğı İngiliz Colossus projesi (1943) Alman ordu şifrelerini başarıyla çözmüştü. Amerika'da ENIAC'tan önce MARK I isimli bir çalışan bilgisayar da vardı. ENIAC'ın Amerikan tarihçelerinde sıfır noktası olarak görülmesinin sebeplerinden biri daha sağlam kuramsal temellere dayanmasıdır (Turing'in kuramsal katkısını ilk başta

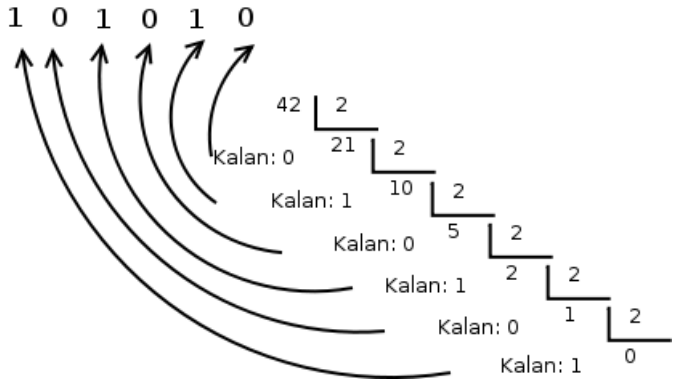
azımsamış olsalar da). Bunu takip eden UNIVAC gibi projelerde Evrensel Turing Makinesi kavramı yavaş yavaş hayata geçmiştir ve daha kullanılabilir hale gelmiştir. Grace Hopper programcıların makine kodu değil de günlük dile daha yakın kelime ve simgelerle program yazması gerektiğini öngören ve UNIVAC için bunu sağlayacak ilk derleyici programları yazanlardandır.

Bir yandan büyük ve pahalı malzemelere dayalı olmalarına rağmen Turing ve diğer fikir önderlerinin kuramlarını gerçekleştiren devasa makineler başarıyla yapılıp çalıştırılırken 1940'ların sonu ve 1950'lerin başına gelişip olgunlaşan bir teknoloji işleri oldukça kolaylaştıracaktır. William Shockley'in gelişmesine katkıda bulunduğu silikon transistörler vakum tüplü transistör yerine göre çok daha küçük cihazlardır. Shockley'in teknolojisini ticarileştirmek için kurduğu firma aynı zamanda Californiya'daki Silikon Vadisi'nin de çekirdeğidir. Ancak daha sonra ekibinden bazıları ayrılacak, yine Silikon Vadisi'nde önce Fairchild sonra da Intel işimli şirketleri kuracaklardır. Silikon Vadisi bilgisayar teknolojisinin gelişiminin bu ilk yıllarından itibaren Stanford üniversitesi gibi bilimsel kurumlarla sanayinin kaynaştığı, son derece hızlı ve yenilikçi bir fikir ortamının zemini haline geldi (Castilla et al., 2000).

2.2.5 Mantık köprüleri kullanarak ikili sayıların işlenmesi

Basit elektriksel bileşenleri kullanarak mantık köprüleri yapabildik ve bunlarla bariyer kontrolü problemini çözebildik. Ancak bilgisayarla yaptığımız işlemler bariyer dedektörlerinden ziyade sayılarla yapılıyor ve ilk bakışta mantık köprüleri ile bu iş arasında bir özdeşlik gözükmüyor.

Sayıları elektronik devrelerle işlemenin (toplama, çarpma, karşılaştırma, vb.) sırrı bu sayıların ikili tabanda ifade edilmesinden geçer. Biz günlük hayatta onluk tabanda yazıyoruz ve o şekilde okumaya alışkınız. Örneğin 42 yazdığımızda bunu hepimiz anlıyoruz: birler hanesinde 2 ve onlar hanesinde 4 var, yani 4x10+2. İkilik tabanda ise sayılar onluk tabandaki 0-9 rakamları yerine sadece 0 ve 1 rakamları ile yazılır. Burada birler hanesinin solunda 2ler hanesi, onun solunda 4ler hanesi, vb., dolayısıyla soldan n hanede 2^{n-1}ler hanesi vardır. 42 sayısını ikilik tabanda yazmak içim her seferinde ikiye bölüp kalanı yazarız, ve bölüm ile aynı işleme devam ederek kalanları ilk yazdığımızın soluna yazarak bölüm sıfır kalana kadar işlemi



Şekil 2.8: 42 sayısının ikilik tabana çevrilmesi

sürdürürüz. Bu şekilde ondalık tabanda 42 için ikilik tabanda 101010 şeklinde altı haneli bir sayı buluruz. Bu işlemi şekil 2.8'de stilize etmeye çalıştım. İşlemin tersi, yani ikilik bir sayıyı ondalık ifadeye çevirmek için ise sağdan sola doğru 2^0 (=1), 2^1 (=2), 2^2 (=4), vb. diye giden haneleri birleştirmemiz gerek. Bu durumda 101010 sayısının tersine çevrimi, sıfır olan haneleri atlayarak: $1x2^5+1x2^3+1x2^1=32+8+2=42$ olarak bulunabilir.

Böylece kağıt üzerinde 42 yazarak ifade ettiğimiz büyüklüğü altı tane kabloya yüksek (1) veya düşük (0) voltaj vererek ifade etmek mümkündür. Daha büyük sayıları ikilik tabanda yazmak için daha fazla haneye ihtiyaç duyacağız. Kitap boyunca ondalık ve ikilik sayıları birbirine karıştırmamak için ikilik sayıların önüne 'i' harfini koyarak yazalım, i101010 gibi.

Bilgisayar teknolojisinde ikilik sistemin tercih edilmesinin temel sebebi şudur: ayırdetmenin bu en basit biçimi daha karmaşık biçimleriyle eşdeğerdir. Bizler zaman zaman gündelik hayatta tarihin en eski sistemi olan birlik sistemi –adını koymasak ta– kullanıyoruz: örneğin restoranlarda adisyon tutulurken, veya evde alışveriş listesi yaparken sayıları belirtmek için ardarda çizgiler attığımızda. Bu sistemde sadece bir sembol var: /. Örneğin üç sayısını ifade etmek için /// yazdığımızda bu üç haneli bir sayı, ancak ondalık tabanda yazılan 111 (yüzonbir) sayısından faklı. Birlik sistemde sağdan ikinci ve üçüncü haneler birin kuvvetlerine karşılık geliyor, aynı ondalık sistemde aynı hanelerin onun kendisi (birinci kuvveti) ve karesine (100) karşılık geldiği gib, ancak

birin bütün kuvvetleri yine bir olduğundan bu hanelerin faydası sınırlı. Dolayısıyla yüzonbir sayısını birlik sistemde yazmak istesek yüzonbir tane çizgiyi yanyana atmamız gerekirdi. Basit işlerde bu sistemi kullansak ta bu gibi büyüklükler için kullanmıyoruz. İkilik sistemi de gündelik işlerde kullanmayız, ancak birlik sistemin aksine ikilik sistemde ayırdetmenin bütün gücü mevcuttur. İkilik sistemde haneler sağdan sola birler hanesi, ikiler hanesi, dörtler (ikinin karesi) hanesi, sekizler hanesi (ikinin kübü) diye devam eder. Ondalık sistemdeki 111 sayısını ikilik sistemde i1101111 diye ifade edebiliriz. Ondalık sistemden birazcık daha uzun bir temsil olsa da ikilik sistem birlik sistemden farklı olarak çok büyük sayıları az miktarda hane ile ifade edebilir ve esas olarak ondalık sistem kadar güçlüdür. Örneğin 99.999.999 sayısının karşılığı i101111101011110000011111111 olacaktır, ki bunun birlik sistemde ne olacağını bir düşünün.

Bilgisayar jargonunda ikilik tabandaki hanelerden herbirine 'bit' adı veriliyor. Bilgisayar satın alırken üzerinde gördüğünüz 32-bit veya 64-bit ibaresi ise o bilgisayarın içindeki ana işlemcinin bir hamlede (bir tiktakta) toplayıp çarpabildiği hane sayısını gösteriyor. Bundan daha fazla hane gerektiren sayılar için ekstra zaman harcanıyor. Tamsayılar dışındaki bilgilerin, örneğin gerçek sayıların, metinlerin, ses veya görüntülerin nasıl ikilik tabanda temsil edildiğinden bölüm 3 bahsedeceğiz.

Bu noktadan itibaren mantık köprülerini (VE, VEYA, DEĞİL) gibi işlemleri ikilik tabanda ifade edilmiş sayıları işlemek için kullanmamız kafanızı karıştırmasın. Mantıksal verilerin alabildiği doğru/yanlış değerleri ile ikilik tabandaki sayıların her bir hanesinin alabildiği 1/0 değerleri arasındaki birebir özdeşlik bizim mantık köprülerini mantıksal önermeler yerine sayıları işlemek için kullanmamıza elveriyor.

Bu işlemlerin nasıl yapıldığına bir örnek olarak toplama işlemini ele alalım. Örneğin i10 (yani 2) ile 11 (yani 3) sayıları. Bunların toplamının 5, yani i101 edeceğiniz biliyoruz. İkilik tabandaki toplama işlemi aynı ondalık işlem gibi yapılır. Önce en sağdaki, en küçük haneleri toplarız, toplam birden büyükse (ondalık tabanda dokuzdan büyükse) bir sonraki haneye aktarma yaparız. Örneğimizdeki sayıların toplamında:

```
   10
+  11
------
```

Şekil 2.9: KISITLI-VEYA köprü sembolü

```
101
```

birler hanesinde böyle bir fazlalık yok, ancak ikiler hanesinde 1+1=2 ediyor, ve o yüzdek ikiler hanesine 0 yazıp fazlalığı 4ler hanesine yazıyoruz.

Bu işlemi mantık köprüleriyle yapmak için her bir haneye karşılık bir hane toplama devresi kullanacağız. Toplama giren iki rakamın herbiri 1 veya 0 olduğu için hepi topu dört farklı durum var, ancak bir de ara haneler için sağdaki hane toplamından gelen fazlalık aktarımını gözönüne almak durumundayız.

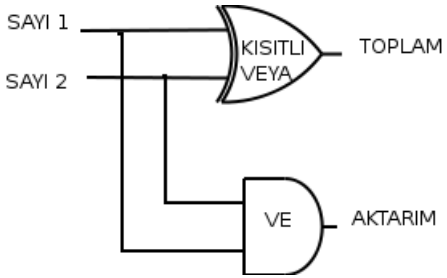
Önce en sağdaki hane için yapalım. Bu haneye kendi sağından fazlalık aktarımı olmadığı için sadece iki rakamı gözönüne alacağız. Eğer rakamlardan sadece bir tanesi 1 ise toplam bir olacak, ikisi de 0 ise 0 olacak, ikisi de bir ise o zaman toplam sıfır olacak ama bir soldaki haneye 1 aktaracağız. Yani sözkonusu devrenin çıktısı sadece hane toplamı değil aktarılacak değer de olacaktır. En sağ hane için iki girdimiz var (iki sayının en sağ haneleri), ancak ara haneler için aktarımları da gözönüne almak gerektiğinden üç girdi olacak. Bu devreyi yapmak için daha önce kullandığımız VE ve VEYA köprülerine ilaveten KISITLI-VEYA adını vereceğimiz yeni bir köprü kullanacağız. Bu köprü iki girdiden sadece bir tanesi 1 iken çıktı olarak 1 veriyor, diper durumlarda (ikisi de 0 veya ikisi de 1) 0 veriyor. Bu köprünün transistör ve direnç kullanarak yapılan devresini vermeye gerek duymuyorum, temelde diğerlerine benziyor; sembolü ise şekil 2.9'de. .

Böylelikle en sağdaki haneyi toplamak için ihtiyacımız olan devrenin mantıksal ifadesi şöyle olur:

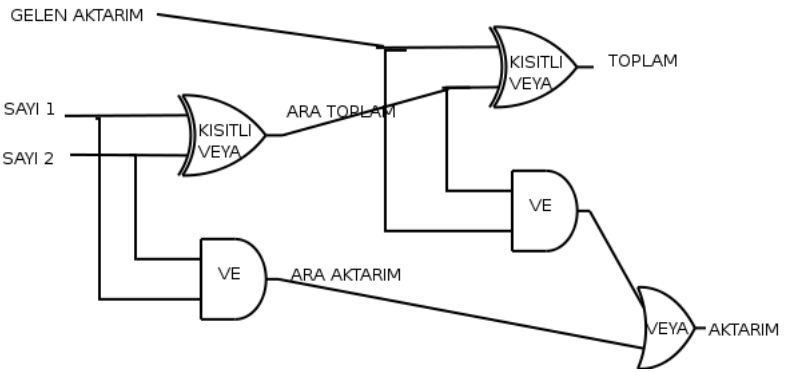
$$\text{Toplam} = \text{sayı-1 KISITLI-VEYA sayı-2}$$

$$\text{Aktarım} = \text{sayı-1 VE sayı-2}$$

Buna karşılık gelen devre ise şekil 2.10'da gösterilmiştir. Bu devreye 'yarım toplayıcı' deniyor, çünkü iki hanenin toplamında sağlarından gelen bir aktarımı gözönüne almıyor.



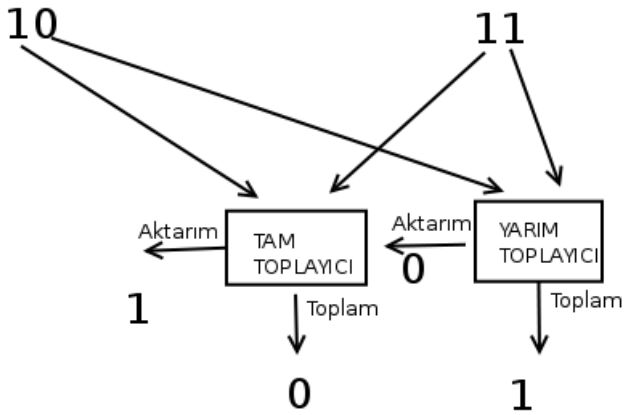
Şekil 2.10: Yarım-toplayıcı



Şekil 2.11: Tam-toplayıcı

Yarım-toplayıcı devreyi sadece en sağ hanelerin toplamı için kullanabiliriz. Diğer hanelerin toplamında 'tam toplayıcı' devre kullanmamız gerek. Şekil 2.11'daki tam-toplayıcı devre çiziminde iki adet yarım-toplayıcıyı ve onlara ilaveten bir VEYA köprüsünü teşhis edebilirsiniz. Aslında tam toplayıcı devre önce bir toplamı yapar, sonra da gelen aktarım değerini alan ikinci bir toplama yapar. Bu devrenin çalışmasına ilişkin diğer detayları size egzersiz olarak bırakıyorum. İsterseniz sayı-1, sayı-2 ve gelen-aktarım verilerinin toplam 8 kombinasyonu için köprüleri takip edip toplamın doğruluğunu görebilirsiniz.

Buraya kadar yaptığımız bileşenler kullanılarak başta ele aldığımız örnekteki iki haneli sayıların toplamını yapacak devreyi çizebiliriz. Bu devre de şekil 2.12'de gösterilmiştir. Devre iki haneli



Şekil 2.12: İki haneli iki sayının toplamını yapan devre

iki sayıyı toplayıp üç haneli bir sayı üretiyor, aynı ondalık sistemde olacağı gibi. Buna benzer şekilde toplama devresini istediğimiz kadar haneyi toplamak üzere tren gibi uzatabiliriz, çünkü toplamanın temel işlevleri yarım ve tam toplama devresinde zaten gerçekleştirildi. Yine de böyle bir toplama işlemini ve karşılık gelen devrenin temel bir problemi var: her zaman iki sayının toplamının bu sayılardaki hane sayısına sığmaması ihtimali var. Bizim örneğimizde ikişer haneli iki sayının toplamı üç haneli bir sayı oldu. Bilgisayar sistemlerinde 'taşma' denilen bu durum (İng. overflow) bir sıkıntı yaratır ve devreler düzeyinde çözümü yoktur, nu yüzden de sistemin alt düzey programlarını yazan programcılar tarafından çözülmesi gereken bir konudur.

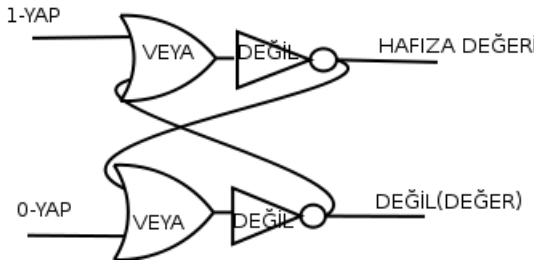
Bu örnek size mantık köprülerinin ikilik tabandaki sayılarla işlem yapmak için nasıl kullanıldığına dair bir fikir vermiştir umarım. Daha kalabalık devreler kullanılarak 32 veya 64 haneli sayıları çarpmak/toplamak veya karşılaştırmak mümkün oluyor. Ancak hane sayısını arttırmanın ciddi sonuçları vardır. Düşünün ki şekil 2.12'deki devre de toplama işlemini aynı bizim yaptığımız gibi hanelerde sırayla ilerleyerek yapıyor. Yani en sağdaki hanenin sonuçlarını bulmadan ikinci hanenin toplamı yapılamıyor. Elektrik sinyalinin toplayıcılar arasında, bilgisayar zamanlayıcısının tiktaklarını takip ederek ilerlemesi gerekiyor. Bu yüzden hane sayısı fazla bir toplama devresi daha yavaş çalışacaktır. Bu meseleyi

çözen daha karmaşık ama verimli devreler var. Ancak bunlar sınırlı bir iyileştirme sağlar ve bu devrelerin daha kalabalık olmasından dolayı çiplerin yapımını biraz güçleştirir. Ayrıca çok haneli sayıları toplama veya çarpma işlemini sayıları daha küçük bloklara bölerek yapmak mümkün, ve buna dair son derece zarif yöntemler geliştirilmiştir; Karatsuba çarpım yöntemi gibi. Son yıllarda 64-bitlik işlemcilerin yapımı aritmetik işlemlere yönelik olmaktan çok bilgisayarın hafıza kapasitesi ile ilgili. Çünkü işlemci çiplerde kullanılan hane sayısı aynı zamanda bellek adreslerini temsil etmek için de kullanılıyor.

2.2.6 Hafıza

Turing makinesinin kuramsal temelinde makinenin durumunu hatırlaması yatar. Bunun modern bilgisayardaki karşılığı ise hafızadır. Hafıza için kullanılan devre oldukça basittir. Meraklı okurlar için şekil 2.13'de bir bit, yani tek haneli hafıza için kullanılabilecek bir devre veriyoruz. Bu devrenin sırrı hafıza değerinin geri dönüş kablosu ile (geri besleme) devrenin bir girdisi olarak kullanılmasında yayıyor. Yani devrenin yaptığı hesap çıktısına bağlanarak bu çıktının korunması sağlanıyor. Hafızanın değerini '1-yap' veya '0-yap' kablolarına yüksek voltaj vererek değiştirebiliyoruz. Bir kez bir değer atandığında neler olduğuna bakalım. Değer 1 ise bu değer alttaki VEYA köprüsüne girdi olarak bağlandığı için bu VEYA işleminin sonucu (1-yap değeri ne olursa olsun) 1 olacak, ve arkasından gelen DEĞİL işlemi yüzünden çıkış kablosunda 0 değeri olacak. Bu değr de üstteki köprüye gidiyor. Eğer 0-yap girdisinin değeri 0 ise 0veya0=0 sonucu verir, dolayısıyla çıkışındaki DEĞİL işleminden dolayı hafıza değeri 1 olur. Böylece bu 1 değeri devredeki geri besleme sayesinde sağlamca 'tutuluyor'. 0-yap girdisi kullanılarak değeri değiştirildikten sonra da benzer şekilde tutulacaktır.

Geri besleme marifetiyle hafızanın bu şekilde tutulması oldukça basit bir işlemdir. Gerçekte ise bir bit'ten çok daha fazla hafıza gerekir. Örneğin bir milyar bitlik bir hafıza olsun. Bu hafızaya erişmek için 1 milyar kablo kullanamayız. Bunun yerine değerini okumak veya değiştirmek istediğimiz hafıza noktasının sıra numarası, daha yaygın adıyla hafıza adresi yine ikilik bir sayı olarak ifade edilir. Birtakım devreler bu sayıya karşılık gelen bit'i bulur ve değerini bize söyler, veya bizim istediğimiz değerle değiştirir. Örneğin 32-bitlik işlemci dediğimizde bu işlemci 32



Şekil 2.13: Tek bitlik bir hafıza devresi

hanelik hafıza adreslerini bu şekilde çözümleyebiliyor demektir, ki 32 hane kullanrak yaklaşık dört milyar farklı hafıza adresini ayırdedebiliriz.

Bunun nasıl yapıldığını anlamak için şöyle bir örnek alalım: diyelim ki 4-bitlik bir hafızayı kontrol etmek istiyoruz. Bu dört hafıza adresini 0, 1, 2, 3 diye numaralıyalım. Bu numaraları ikilik tabanda yazarsak i00, i01, i10, ve i11'e karşılık gelir. Dikkat ederseniz 2ler hanesi (soldaki hane) hafızanın ilk yarısı için 0, diğer yarısı için 1 değerini alıyor. 4 farklı adresi 2 hanelik bir sayıyla adresleyebiliyoruz. Bilgisayarın hafıza erişim çipleri hafıza adresinin olası bütün kombinasyonlarını mantık köprülerinden oluşturulan bir kırılım devresi marifetiyle tek bir hafıza bileşeniyle birleştirir. Böylece hafıza kontrol devresine bir hafıza adresi ve onunla ilgili işlem (değerini okuma, 0-yap, veya 1-yap) az sayıda kablo ile girildiğinde istenilen hafızanın değeri de az sayıda kablo ile sağlanır. Buna karşılık hafıza kontrol çipinin içi hafıza kapasitesiyle orantılı miktarda mantık köprüsününin bulunduğu kalabalık bir devredir.

Genel olarak n haneli bir sayı ile 2^n farklı hafıza noktasını adresleyebiliriz. Son yıllarda 32-bit işlemciler yerine 64-bit işlemcilerin ortaya çıkışı 32-bit işlemcilerdeki 2^{32}, yani dört milyar (4GB) sınırının artık gelişen teknoloji tarafından zorlanmaya başlaması yüzündendir. 64-bitlik işlemcilerle 2^{64} büyüklükte hafıza kullanabileceğiz, ki bu yanılmıyorsam 16 Petabyte ediyor.

2.3 Turing'in dehası: her kılığa girebilen makine

Turing makinesinin basit tarifini yukarıda bölüm 2.1'ten hatırlayalım. Önceki bölümlerde incelediğimiz hafıza devreleri sayesinde

makinenin ikilik tabanda bir sayı olarak ifade edilen 'durumunu' hatırlamasını sağlayabiliriz. İncelediğimiz mantık köprüleri hem mantıksal kuralları gerçekleştirmek hem de ikilik sayılarla aritmetik işlemleri yapmak için kullanılabilir. Böylece şekil 2.2'teki gibi bir Turing makinesinin kurallar kısmı istenilen kurallara karşılık gelecek şekilde mantık köprüleri birleştirilerek yapılabilir.

Böylece tek bir işi yapacak bir elektronik makine yapılabilir. Bu makinenin örneğin elektrik motoru veya biçerdöverden farklı olarak sayıları işliyor olması da tek başına kayda değer bir durum olur. Gerçekten de 2. Dünya savaşı sırasında inşa edilen makineler bu türdendi ve kritik önemi olan başarılara imza attılar. Ancak günümüzdeki bilgisayar teknolojisinin temelini oluşturan önemli bir unsur Evrensel Turing Makinesi kavramıdır. Turing tarafından kuramsal bir matematik problemine atfen ortaya atılan bu farazi makine istenilen sabit işlevli herhangi bir Turing makinesini 'taklit eden' bir makine olarak kurgulanmıştı. Bu kavram günümüzde istenilen 'program'ı (yani belirli bir Turing makinesini) yükleyip çalıştıran (taklit eden), işlevi önceden sabitlenmemiş bir makine olarak bilgisayara karşılık gelmektedir.

Önceki bölümde kullanılan transistör ve mantık köprülerinden böyle bir makinenin nasıl yapılacağını tarif etmek bu kitabın kapsamını aşıyor, ancak temel özelliklerinden bahsedebiliriz. Bu makinenin en temel özelliği işleyiş kurallarının sabit bir elektronik devre olarak bağlanmış olmayıp, onun yerine bu kuralların da bir dizi sayı olarak ifade edilip makineye verilmesidir. Pratikte belirli bir işi tarif eden kurallar dizisi önce hafızaya yüklenir. Genel amaçlı makine hafızanın şu yada bu bölümündeki kural dizilerini çalıştırmaya yönlendirilerek farklı programları gerçekleştirebilir.

Bunun yapılabilmesi için bu makinenin bir dizi komutu anlayabilmesi, ve kural dizilerinin sadece makinenin anladığı komutlardan oluşması gerekir. Günümüzde evlerimizde kullandığımız bilgisayarın beynini oluşturan merkezi işlemci çipleri yaklaşık 100 tane farklı kuralı/komutu anlayabilir. Bu çipler tasarlanırken kural/komut'ların neler olacağına ve bunlara karşılık gelen komut numaralarına karar verilir. Örneğin i101010 kodlu komut şu anlama gelebilir: "hafızanın A adresindeki sayı ile B adresindeki sayıyı karşılaştır, A B'den büyükse bir sonraki komuta geç, aksi takdirde iki sonraki komuta geç". Başka bir komut kodu ise "A ile B'yi topla, toplamı C'ye koy" anlamına gelebilir. A, B, C gibi sınırlı ve sembolik adres kodlarının yanısıra şöyle bir

komut ta olabilir: “bu komuttan sonra gelen dört komut kodu gerçekte hafıza adreslerini ifade ediyor, birinci adresteki değer ile ikincidekini karşılaştır, eğer ilki küçükse üçüncü adresteki komuta atla, değilse dördüncü adresteki komuta atla”. Böylece kural dizileri dış dünyadan veya hafızadan sayıları alıp, onların karşılıklı değerlerine göre kurallar arasında gerekli zıplamaları yaparak istenilen işi gerçekleştirir. Modern bir bilgisayar sisteminde istenilen kural dizisini alıp gerçekleştirebilen, Evrensel Turing Makinesine karşılık gelen bileşene merkezi işlem ünitesi, veya İngilizce isminin kısaltmasından CPU (Central Processing Unit) diyoruz. Bu bileşen bilgisayarın kapağını söktüğünüzde göreceğiniz büyücek ve çok sayıda bacağı olan entegre devredir.

Bir programcının yarattığı programın en nihayetinde CPU’nun anlayacağı, ikilik tabanda bir sayılar dizisi olarak ifade edilmesi gerekir, ve ancak o zaman makine tarafından işleme konabilir. Bu kodlamaya ‘makine kodu’ deniyor. Bilgisayar teknolojisinin ilk yıllarında programcıların birtakım kablo veya anahtarları düşük/yüksek voltaja göre ayarlayarak bu komutları bilgisayara vermesi gerekiyordu. Hem programcının yöntemindeki esasa dair hatalar, hem de bu kural dizilerinin makine koduna çevrimindeki hatalar bilgisayarın olmadık yerlere sıçrayıp ‘kaybolmasına’, veya işin içinden çıkamamasına yolaçabilir, ki nadiren de olsa böyle durumlarla karşılaşırız.

Bilgisayar teknolojisinin gelişmesiyle beraber programcılık mesleğinin icrasında kullanılan araçlar da gelişmiştir. Teknolojinin ilk yıllarından beri programcıların uğraştığı önemli bir konu bir insanın matematiksel çözümleri doğal konuşma diline yakın bir dilde ifade ettiği, okunup anlaşabilecek ve düzeltilebilecek bir metni alıp bilgisayarın anladığı kodlara çevirecek programlar yapmak olmuştur. Ancak konuşma dili bilgisayar için anlaması çok güç olduğundan daha sınırlı kelimenin, çok tertipli olarak kullanıldığı programlama dilleri geliştirilmiştir. Gitgide daha karmaşık işleri bilgisayar programına dökebilmemizi sağlayan unsurlardan biri detaya dair hataları yakalamamıza ve düzeltmemize yardım eden, ve programı esasına odaklanmamızı sağlayan bu tür programcı programları.

Bilgisayarın asli becerileri makine kodunda kullanılan komut setindedir. CPU tasarımcıları ve üretici firmalar bu konuda değişik yaklaşımlar denemişlerdir. Bu yaklaşımların oluşturduğu dağılımın bir ucunda minimalist yaklaşımlar var. Bu türden CPU’lar

olabilecek en az sayıda temel komutu anlarlar, ve o komutları görece daha hızlı bir şekilde gerçekleştirirler. Bunun zıt ucunda ise yüzlerce farklı komut içeren bir komut setine sahip maksimalist CPU tasarımları var. Bu ikinci türden CPU'lar bazı komutları daha yavaş çalıştırsalar da başka türlü bir zenginlik, komut zenginliği, sunuyor. Ayrıca birinci türden CPU'nun birkaç adımda yapacağı, ancak ikinci türden CPU'nun komut setinde tek bir komuta karşılık gelen işlemleri birinciden daha hızlı yapıyor. Günümüzde çok yaygın kullanılan Intel ailesi bazı işlemciler bu ikinci türdendir ve birinci gruba göre daha popülerdir. Geçmiş tecrübeler bu ikinci tasarım yaklaşımının genel amaçlı olarak daha iyi netice verdiğini gösteriyor. Ancak birinci türden CPU'lar da bilimsel veya endüstriyel uygulamalarda performans için kullanılıyor ve bu iki tarz farklı uygulama alanlarında yaşamaya devam edecek gibi görünüyor.

2.4 Elektronik olmayan bilgisayarlar

Prensip olarak ayırdetme, karşılaştırma-birleştirme, ve hatırlama işlevlerini gerçekleştirebilen makinelerle hesaplama yapılabilir. Bunun en üst düzeye ulaştığı elektronik ve silikon çip teknolojisi bir tarafa başka türlü makinelerle yapılması da denenmiş ve denenmektedir.

Bu deneysel teknolojiler arasında örneğin DNA-tabanlı sistemler var. Canlı genlerinin temeli olan DNA dizisi dört çeşit aminoasitten oluşur; yani ikilik değil de dörtlük tabanda yazılmış kodlardan oluşur. Hücrelerimizde bu dizilerdeki belirli kombinasyonların tetiklediği enzimlerin başrol oynadığı protein sentezleme süreçler var. Canlı biyokimyasının temelinde yatan bu süreçler ilaç sanayisi için de önemlidir, ve bu yüzden bilgisayar kuramlarının DNA kimyasına uyarlanmasına dair deneysel çalışmalar yapılmaktadır. Bu çalışmalar sonucunda örneğin hücre kimyasındaki belirli durumların tespit edilip sorunu çözecek ilacın serbest bırakılması gibi yöntemler hayata geçirilebilmekte (Benenson et al., 2004), veya bir yüzey üzerinde hareket eden moleküler nano-robotlar yapılabilmektedir ('nano' terimi son yıllarda nanometre kısaltması olarak, moleküler düzeyde detaylara dayalı teknolojiler için genel bir ad olarak kullanılıyor). Bambaşka tür bir çalışma ise matematik problemlerin DNA dizisi olarak ifade edilip çözülmesi yönündeki çalışmalardır (Adleman, 1994). Her iki türden çalışmada da DNA dizileri yukarıda bahsettiğimiz hafıza

ve girdi çıktı işlevlerine karşılık gelir, mantık köprüsü olarak ise zaten hücrelerimizde varolan ve becerileri buna karşılık gelen enzimler kullanılır. Kimyasal süreçler silikon bilgisayarlara göre görece daha yavaş, bilgi girdi çıktısı daha zor da olsa önemli bir avantaja sahipler: trilyonlarca molekülü küçücük bir alanda işe koşmak mümkün. Biyokimyasal süreçleri elektronik devrelerde olduğu gibi uygun adım yürütmek zordur, ve garantili bir işleyiş yoktur (Head et al., 2002). Bütün engellere rağmen bir yandan vaadettiği potansiyel bu araştırmaların sürmesini teşvik ediyor, bir yandan da hesaplama ile ilgili kavrayışımızın gelişmesini sağlıyor.

Son yıllarda adı geçen başka bir deneme ise kuantum hesaplama ve kuantum bilgisayarlar. Kuantum fiziğin bizim lise derslerinde gördüğümüz fizikten önemli bir farkı elektron gibi kuantum parçacıkların belirli bir yerinin olmamasıdır. Ne kadar şaşırtıcı gelse de, ve her nekadar bu tür şeyleri büyütüp görmemiz mümkün olmasa da elektronlar aynı anda birsürü yerde, daha doğrusu heryerde birden bulunurlar. Yerleri kesin değil olasılıksal olarak bilinir. Bunun bir sonucu olarak bir kuantum bilgisayardaki bit'lerin bizim ev bilgisayarımızdakiler gibi sabit bir değeri yoktur; sadece her iki değeri de alma olasılıklarından sözedilebilir. Bu durumun önemli bir sonucu var: bazı problemlerde problemin çözümü olan doğru sayıyı ararız, ve elimizdeki bilgisayarlara bu çözümü bulmak için bütün muhtemel sayıları denememiz gerekir. Diyelim ki çözümün 32-bitlik bir sayı olduğunu biliyorsak 2^{32}, yani dört milyar farklı ihtimali teker teker denemek zorunda kalırız. Örneğin endüstride süreç optimizasyonunda karşımıza çıkan problemler bu türdendir ve sözkonusu endüstri uygulamasındaki detaylar arttıkça yöntemlerimiz pratikte uygulanamaz olur. Oysa kuantum bilgisayardaki bitler, daha doğrusu ku-bitler aynı anda bütün olası değerleri alır! Bu yüzden kuantum bilgisayarlar bildik hesaplanabilirlik kuramlarını kökten sorgulamamıza neden oluyor. Bunların pratikte gerçekleştirilebilmesi halinde neyin hesaplanabilir olduğu konusunu temelden gözden geçirmemiz gerekecek.

Bölüm 3

Gerçek Dünyanın Rakamlarla Temsili: Dijital Dünya

Elektronik bilgisayarlar sadece bir tek ayrımın farkına varabilir: voltaj var veya yok. Bütün basitliğine rağmen bu ayrımın tekrar tekrar kullanımıyla oldukça karmaşık şeyler ifade edilebilir. Bizler düşündüklerimizi yazılı ifade ederken harfleri, kelimeleri, ve yazım işaretlerini kullanıyoruz. Birtakım büyüklükleri sayılarla ifade ediyoruz. Bilgisayar ise bunları hem anlamak hem de ifade etmek için sadece 1/0 kullanıyor. Bu bazen oynadığımız bir oyunu hatırlatıyor: kim olduğu bilinmeyen bir kişiyle ilgili sorular sormak ve anlatıcının sadece evet veya hayır diyerek verdiği cevaplara göre o kişinin kim olduğunu bulmak. Bilgisayarlar kendi aralarında ve bizimle iletişim kurarken sürekli ve çok defalar bu oyunu oynamaları gerekiyor. Neyse ki bilgisayar kuralı belli oyunları oynamak konusunda çok hızlıdır.

3.1 İkili sistem ve elektronik bellek: bit'ler byte'lar, ve dijitallik

Bölüm 2.2.5'te günlük kullandığımız ondalık sayıların ikilik tabana nasıl çevrildiğine bakmıştık. Günümüzde bir bilgisayarın herbiri 1 veya 0 olabilen milyarlarca ikilik tabanda rakamdan, yani

'bit'ten oluşur. Bu rakamların değeri kadar sıralanışı da önemlidir. Bu bölümde bilgisayar belleğinin düzenlenişiyle ilgili bazı temel bilgileri gözden geçireceğiz.

Belleğin içeriği gibi düzenlenişinde de ikilik taban temeldir. Çünkü belleğe erişim, yani bellek adresleri de ikilik tabanda ifade edilir ve mantık köprüleri üzerinden çatallanarak yapılır. İkilik tabanda bir haneye İngilizce 'zerre' karşılığı 'bit' diyoruz. Ancak geleneksel olarak bellek 8 haneden oluşan parçalarla ifade edilir, ki bu parçalara 'byte' deniyor[7]. Bu 8 sayısının seçiminin tarihsel sebepleri var. Bilgisayar teknolojisinin ilk yıllarında bilim insanları gündelik işlerimizde kullandığımız sembolleri (küçük-büyük harfler, 0'dan 9'a rakamlar, noktalama işaretleri) ve buna ilaveten o yıllarda bilgisayarla insanın iletişimini sağlayan temel araç olan yazıcıları kontrol etmek için gerekli sinyalleri (satır atla, sayfa atla, satır başına dön, vb.) ifade etmek için yüzün üzerinde sembole ihtiyaç duydular. Bu semboller listesinin tamamı 7 hanelik bir ikilik sayıyla ifade edilebilirdi, çünkü $2^7 = 128$ ihtiyaç duyulan sembol sayısını karşılıyordu. Ancak o günün abilim insanlarına hane sayısı olarak 7 yerine kendisi ikinin bir kuvveti olan 8'i seçmek daha anlamlı görünmüştür.

Bit ve byte terimlerinin kökeni böyle iken bir de MB (megabyte), GB (gigabyte) gibi terimleri açıklayalım. Günlük dilde bin, milyon, milyar gibi büyüklükleri kullanıyorken bilgisayar belleği miktarından bahsederken bunları kullanmıyoruz. Onun yerine herbiri iki sayısının bir kuvverti olan bu büyüklükleri kullanıyoruz. İkinin onuncu kuvveti, $2^{10} = 1024$ eder, yanı ondalık sistemde kullandığımız bin sayısına oldukça yakın. Kilobyte (KB) dediğimizde 1000 değil de 1024 byte büyüklüğünde bir bellekten bahsederiz. Benzer şekilde 2^{20}=1024x1024 (yaklaşık bir milyon) byte yerine megabyte, 2^{30} (yaklaşık bir milyar) byte'a gigabyte, ve son yıllarda yaygınlaşmaya başlayan 2^{40} byte'a terrabyte (TB) diyoruz.

Sabit disk ve RAM gibi bellekler byte ile ölçülürken örneğin İnternet'e bağlanmak için kullandığımız cihazlarda saniyelik veri aktarım hızı byte değil de bit sayısı ile ölçülüyor, örneğin 1 Mbps(İng. megabit per second, saniyede megabit) gibi. Böylece bu tür kısaltmalarda byte için B, bit için b kullandığını görüyoruz.

İkilik tabandaki sayıların bilgisayar belleğine dair kullanımıyla ilgili bu terimleri açıkladıktan sonra bir terimi daha açıklayalım: dijital. Dijit sözcüğü latin dillerinde hane anlamına geliyor. Bu

bölümün devamında yazıların, büyüklüklerin, renklerin, seslerin, vb., nasıl ikilik tabanda sayılar olarak ifade edildiğine, yani dijitalleştirildiğine bakacağız. Bu ifade etmelerin herbiri belirli sayıda hane ile olacak, ve göreceğiz ki bu durum da bir sınırlamadır. Nasıl ki $\sqrt{2}$ sayısı irrasyonel bir sayıdır ve ne kadar hane kullanırsak kullanalım kesin değerini yazmış olmayız, aynı şekilde özellikle sesler ve renkler gibi şeyleri sınırlı hanede sayılarla ifade ettiğimizde onları indirgemiş oluyoruz; ne var ki çoğu zaman o kadar çok hane kullanıyoruz ki bu indirgeme pratikte önemini kaybediyor. Genel olarak gerçek dünyadaki büyüklüklerin sınırlı sayıda hanesi olan sayılarla ifade edilmiş hallerine dijital, ve bunda yapılan indirgemeye de dijitalleştirme diyoruz. Türkçe'ye sayısal olarak tercüme ediliyor, ancak günümüzde daha yaygın olarak kullanıldığından burada dijital sözcüğünü tercih edeceğiz.

3.2 Yazılar ve sayıların dijital olarak ifade edilişi

Bilgisayar teknolojisinin ilk yıllarında bilim insanlarının yazıları bilgisayarda ifade etmek için bir ihtiyacı oldu: ikilik tabanda yazılmış 8 haneli sayıların, yani byte değerlerinin herbirinin hangi harf veya sembole karşılık geldiğini belirleyen bir 'karakter seti'. Bu karşılıklar bir standarda bağlanmalıydı ki sayılar yorumlanabilsin, yazıcılar hangi değeri gördüğünce satır atlayacağını, hangisinde büyük A harfini basacağını bilebilsinler. ABD'de 1968'de geliştirilen bu standart karakter seti ASCII kısaltması ile anılır (İng. American Standard Code for Information Interchange, Bilgi değiştokuşu için Amerikan standart kodu)[8]. Bu karakter seti hazırlanırken sembollerin kodlanacağı byte'ların en sol hanesinin kullanılmamasına, yani hep 0 olmasına karar verilmişti. Böylece 8 haneye yazılabilen rakamlardan 0-127 arasında olanlar, yani i00000000'dan i01111111'e kadar olanların herbiri yazım dilindeki bir harf veya sembole veya bir yazıcı komutuna karşılık gelecek, ancak kalan değerler kullanılmayacaktı.

Daha teknolojinin ilk yıllarından itibaren uzaktaki bilgisayarlar arasında bilgi değiştokuşunun öngörülmesi ve o yüzden bu tür standartlara önem verilmesi dikkat çekicidir. Göreli bir özgürlük dönemi olan 1960'larda Internet vizyonun ortaya koyanlar iletişim ve bu teknolojiye erişim özgürlüğüne çok önem veriyorlardı (Licklider and Taylor, 1968).

Benim gençlik yıllarımda Türkçe konuşan meslektaşlarımla bilgisayar ortamında yazışmalarımızı İngiliz/Amerikan alfabesi

ile yapıyorduk. Kaçınılmaz 'karisikliklar' bir yana, birkaç harf dışında Latin dillerinin Türk alfabesi ile benzerliği böyle bir iletişimi mümkün kılıyordu. Bilgisayar teknolojisinin Latin ve Anglosakson ülkeleri dışında hızla yayılmasıyla ciddi bir sorun ortaya çıktı çünkü Arapça, Çince gibi dillerin ASCII karakter setiyle hiçbir benzerliği yoktu ve bizim yaptığımız gibi 'idare etme'leri sözkonusu değildi. Bu soruna üretilen ilk çözüm ISO (İng. International Standards Organization, Ulusalararası Standartlar Organizasyonu) tarafından her dil veya dil grubu için bir karakter seti tanımlanması oldu. Bunu yaparken de orijinal ASCII tasarımında kullanılmamış olan 128-255 arası kodlar devreye sokuldu ve farklı dillere ait alfabeleri ASCII üzerine eklemek için kullanıldı. Örneğin Türkçe için olan karakter seti ISO 8859-9 adıyla standartlaştırıldı. Fakat bu yaklaşım bir sorunu çözerken başka bir sorun yaratıyordu: her dosyanın hangi standart karakter setini kullanarak yazıldığının hatırlanması veya kaydedilmesi gerekecekti, ki bu soruna bir çözüm önerilmemişti ve çözümü göründüğünden daha çetrefilliydi. Üstüne üstlük Microsoft ve Macintosh gibi firmalar ticari aceleyle uluslararası standartlaşma sürecinden farklı hareket ettiler ve hitap ettikleri uluslararası pazarlara uygun kendi karakter setlerini geliştirdiler. Böylelikle 80li, 90lı yıllar ve 2000lerin bir kısmı boyunca bir dostunuzun yazdığı ve e-posta ile gönderdiği metni başka bir bilgisayarda okuman ciddi bir karın ağrısıydı; hatta eski, eksik alfabeyle haberleşme yönteminden bile beterdi. Üstelik yeni yöntem Çinlilerin sorununu tam olarak çözememişti.

Son on yıl boyunca yavaş yavaş yaygınlaşan yeni ve daha iyi bir çözüm var. UTF-8 (İng Unicode Transformation Format-Tekdüze çevrim formatı) adı verilen evrensel ve tek karakter seti bilinen hemen tüm dillerin alfabe sembollerini içeriyor. Bunu yapmak için yine ASCII standardında boş bırakılan son bite dayanan bir kodlama açılımı yapılyor. Ancak eski ISO yaklaşımının tersine bu sefer karakter setinin birden fazla byte'a yayılmasına izin veriliyor. ASCII kodlanmış bir metnin UTF-8 kodlaması tamamen aynı, ancak ASCII'de kullanılmayan sekizinci bitin kullanımı hemen ardından gelen byte veya byte'ların birkaçının birden yorumlanması gerektiğine işaret ediyor. Yani semboller tek bir byte değil sayısı değişken birkaç byte kullanarak kodlanıyor. Böyle birden fazla byte'a saçılarak UTF-8 kodlaması 256 sembol sınırını aşabiliyor.

UTF-8 daha karmaşık bir kodlama şekli olduğu için yaygınlaşması zaman almıştır. Ancak günümüzde uluslararası İnternet standartlarında kilit önem taşıyor (Gençer et al., 2006), ve güncel hemen her kelime işlemci programı, veya e-posta sistemi bu standardı destekliyor. Artık metin dosyaları ve İnternet web sayfalarının pek çoğu bu karakter setini kullanıyor. Ne yazık ki zaman zaman yeni web sayfalarında miadı dolmuş karakter kodlamalarının kullanıldığını, veya iddia edilen kodlama ile gerçekte kullanılanın farklı olduğunu görüyorum. Bu da mesleki eğitimin çok temel bazı konularda yetersiz olduğuna işaret ediyor.

Şimdi de büyüklüklerin nasıl ifade edildiğine bakalım. Bilgisayar belleğinde sayı değeri kadar belleğin yapısal bölünüşü de önemlidir demiştik. Günümüzde yaygın olarak kullanılan işlemciler 32 veya 64-bitlik (yani 4 veya 8 byte'lık) işlemcilerdir. İşlemcinin kaç haneli sayılarla başedebildiğini gösteren bu ölçütün çifte anlamı vardır: (1) bu işlemciler bir hamlede ya 32 ya 64 haneli ikilik sayıları toplayıp çarpabilirler, ve (2) bu işlemciler bellek adresi olarak ya 32 ya 64 haneli ikilik sayıları kullanırlar. İşimizi basitleştirmek için tartışmamızı 32-bitlik işlemcilerle sınırlayalım.

Doğal sayıların, yani sıfır ve sıfırdan büyük sayıların bu 32-bit üzerinde yazılması kolaydır: aynı bir muhasebeci defterinde olacağı gibi sayının değeri sağa yanaşık yazılır, solda boş kalan bitlerin değeri 0 olur. 32 haneli bir ikilik sayı oldukça büyük değerler için kullanılabilir diye düşünebilirsiniz: yaklaşık bir trilyon'a kadar. Yine de çok büyük sayılarla karşılaşırız ve bu tedavisi programcılara kalmış bir problemdir, çünkü makinenin sınırlarını zorlar. Çoğu programlama dili işlemcinin doğal kapasitesinin (örneğin 32-bit) iki katı uzunlukta sayıları işlemek için özel veri tipleri sağlayarak bu sorunu çözmeye çalışır.

Tam sayılar doğal sayılardan farklı ifade edilir çünkü artı veya eksi değerler alabiliyorlar. Bu durumda akla gelen bir çözüm 32 bitten bir tanesinin sayının artı mı eksi mi olduğunu belirtmek için kullanılmasıdır. Ancak aritmetik işlemlerin sıhhatli yapılmasını ilgilendiren sorunlardan dolayı biraz daha karmaşık çözümler gerekmiştir.

Gerçek sayılar sözkonusu olduğunda fazladan bir problem vardır: ondalık noktanın nerede olduğu. Örneğin -129.25 sayısının ifadesi 12925 rakamları, eksi olduğu bilgisi, ve ondalık noktanın üçüncü rakamla dördüncü rakam arasında olduğu bilgisini gerektirecektir. Ancak bu şekilde, örneğin böyle iki sayıyı toplarken

noktaların denk getirilmesi sağlanabilir. Bu sorunun genel çözümü bahsedilen sayının -12925x10^{-}2 diye yazılmasıdır, ki bu notasyon büyük ve sağda sıfırı bol sayıların da ifade edilmesine izin vermek gibi bir avantaja sahip. Bu konuyu çözmek için yapılmış standartlardan yaygın kullanılan biri 1 biti artı/eksi işareti, 8 biti $10^{?}$ bilgisi, ve kalan 23 biti de sayının rakamları için kullanıyor. Bu notasyona kayar nokta notasyonu deniyor. Gerçek sayıların toplama veya çarpma işlemleri tamsayılara göre daha karmaşık olduğundan ve bir o kadar da sık gerektiğinden işlemcilerin (CPU) bir alt-parçası gerçek sayılar aritmetiği için ayırılmıştır, bu parçaya da kayar nokta aritmetiği ünitesi denir.

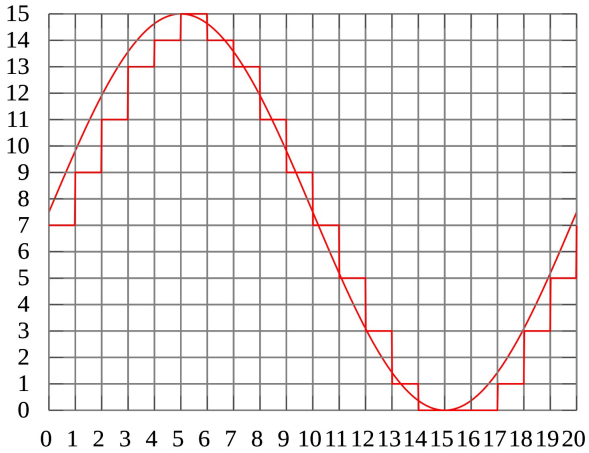
Rasyonel sayılar bilgisayar sistemlerinde nadiren kullanılıyor. Bunların matematik tarihinde gerçek sayılardan öncesinden beri yaygın olduğu düşünülürse bu ilginç bir durumdur. Benim bildiğim kadarıyla sadece Scheme/Lisp programlama dili ailesi rasyonel sayıları kullanma imkanı veriyor. İşlemcilerin elektroniği gibi programlama dilleri de kendilerini tamsayı ve kayar nokta gerçek sayı aritmetiğine sınırlamıştır. Benzer şekilde komplex sayıların kullanımı da ancak bazı programlama dillerinde görülüyor.

3.3 Ses ve görüntünün dijitalleştirilmesi

Dijitalin karşıtı 'analog'dur, kelime anlamı ise 'benzeşim'. Eski ses ve görüntü teknolojileri hep analogdu. Örneğin plağa kaydedilmiş sesler böyledir: kayıt anındaki ses dalgası bir mikrofon aracılığı ile aynen alınır ve sert plastiğin üzerine bu dalgaların benzeşimi işlenir. Öyle ki plakçaların sesi kapalıyken bile iğnenin bu dalgalar üzerindeki hareketinden müziği duyabilirsiniz. Benzer şekilde radyo yayınında da havadaki yayını oluşturan elektromanyetik dalga istenilen ses dalgasının benzeşimini sırtında taşır.

Benzeşim/analog teknoloji görece basit olmakla beraber temel bir sorunu vardır: plak çizildiği veya radyo yayınına parazit karıştığı zaman işler bozulur. Bu tür problemler karışık ve hassas cihazlar kullanılsa bile ancak kısmen çözülebilir. Ayrıca bunların kopyalanması mükemmel yapılamadığından her kopyalamada kayıt biraz daha bozulur. Zaman da plak, manyetik teyp gibi kayıt ortamlarını yıpratır ve bu yıpranma geri döndürülemez.

Sesin dijital olarak ifade edilmesi ise ses dalgasının sayılarla ve belirli bir düzende ifade edilmesini gerektirir. Şekil 3.1 bunun nasıl yapılacağını gösteriyor. Ses dalgası gibi 'sürekli' değişen bir dalganın her anını sayıya çevirip saklayamayız, bu teorik



Şekil 3.1: Sesin örneklenmesi ve dijitalleştirmesi

olarak mümkün değildir. Ancak sesin frekansına göre uygun sıklıkta seçilmiş bir örneklemi, yani belirli aralıklarla seçilmiş noktalardaki ses yüksekliğini kaydetmek büyük ölçüde yeterli olacaktır. Yeterince sık örnek alırsak ses dalgasındaki salınımları kaçırmayız. Bir imkansızlık ta seçilen noktadaki ölçümün kaydedilmesiyle ilgilidir. Ölçülen ses yüksekliğini sınırlı hanesi olan bir sayı olarak yazmak zorundayız, ve buna sığmayan küsurat varsa kaybedeceğiz. Yani hem sürekli bir dalgayı süreksiz bir örneklem ile ölçüp kaydetmekten, hem de ölçülen değeri sınırlı sayıda hanesi olan bir sayıyla yazmaktan dolayı kayıplar yaşarız.

Şekil 3.1'deki ızgaranın dikey çizgileri örnekleme, yatay çizgiler ise ölçümün sayı kodlamasında ifade edilebilecek değerlere karşılık geliyor. Dolayısıyla örneklem için seçtiğimiz noktalarda ses dalgasına ızgarada en yakın noktaları kaydetmiş oluyoruz. Yaptığımız dijital kayıt orijinal ses dalgasını şeklin sağ tarafında gösterildiği gibi kesik çizgilerle ve küsurat hatalarıyla takip eder. Oysa analog bir teknoloji ile mükemmel kayıt yapsaydık orijinal ses dalgasının her anını aynen plak veya manyetik teyp üzerine kaydetmiş olacaktık. Yani teorik olarak dijital kayıtta analog kayda göre bilgi kaybı sözkonusudur, kayıt sadece yaklaşık bir kayıttır, kesin değildir.

Pratikte ise şekil 3.1'deki ızgarayı çok ince ve sık bir ızgara yaparak bu kayıpları çok aza indirgemek mümkün oluyor. Bilgisayarınızdaki mp3 formatında kaydedilmiş şarkı dosyalarından birinin özelliklerin göz atın. Böyle dijital bir ses kaydı hakkında bilgisayarınız tipik olarak 44100 Hz (hertz), stereo (çift kanal), kanal başına 16 bit türü bir bilgi gösterecektir. Bu değerler ses örnekleminin saniyede 44100 kez yapıldığını gösteriyor. Böyle bir örneklem sıklığı insan kulağının tipik olarak duyabildiği 20Hz-16000Hz aralığındaki konuşma ve müzik sesleri için yeterlidir. Bu örnekleme sesin içindeki daha yüksek frekansta salınımları kaçırır ama biz bunları zaten duyamadığımız için bunun bir önemi olmaz. Kodlamanın 16 bit olduğu bilgisi ise bize her bir örneklem ölçümünün 16 bitlik bir ikilik sayı olarak yazıldığını gösteriyor. Bu da ızgaranın yatay çizgilerinden 2^{16}, yani yaklaşık 65.000 tane olduğunu gösteriyor. Bu özelliklerin tamamı oldukça sık bir ızgara kullanıldığını gösteriyor. Üstüne üstlük dijital kayıtları tekrar sese dönüştüren elektronik devreler bizim kesik çizgili kaydettiğimiz dalgayı tekrar yuvarlak hatlara dönüştürdüğü için orijinal sese son derece yakın bir ses elde etmiş oluruz. Bu örnekteki örneklem sıklığı ve kodlama çözünürlüğü müzik CDlerinde tipik olarak kullanılan değerlerdir.

Müziği dijitalleştirmek için bunca çabanın önemli bir ödülü vardır: sayılar hatasız olarak kopyalanabilir. Plak veya manyetik teypleri kopyalamanın aksine bir CDyi kopyalamak yüzde yüz eşdeğer kopyasını çıkartmak demektir. Yani ızgaralama ile yaşadığımız küçük kayıplara karşılık elde ettiğimiz kaydı sonsuza dek eskimeden saklama imkanını elde etmiş oluyoruz. CDler de plaklar gibi çizilebilir, ancak sayısal hata tespit/giderme yöntemleri ile belirli bir düzeye kadar bu tür bozulmalar tolere edilebiliyor.

Ses dijitalleştirmesinde örneklem ızgarasının sıklığı ister dikey (örneklem sıklığı) ister yatay (kodlama çözünürlüğü) arttıkça örneklem miktarı artar ve ses dosyasının boyutu büyür. Hernekadar elimizdeki bellek miktarı yıllar içinde hızla arttı ve ucuzladıysa da bazen bu boyut sıkıntı yaratır, özellikle de bu tür kayıtları İnternet üzerinden aktarıp çalmak gerekiyorsa. MP3 gibi saklama formatları ses kayıtlarını sıkıştırarak daha küçük dosyalara sığdırma imkanı veriyor. Detaylarına girmeyeceğimiz karmaşık matematik tekniklere dayanan bu yöntemler çok küçük ilave bir kayba karşılık bire on düzeyinde sıkıştırma yapabiliyor.

Bir de görüntülerin ve filmlerin nasıl dijitalleştirildiğine baka-

lım. Aynı ses olduğu gibi görüntü de ışığa dair sürekli bir bilgidir, yani her noktasını kaydetmemiz mümkün değildir. Gözümüz retinadaki hücrelerin yoğunluğunca, eski tip fotoğraf makinesi ise film üzerindeki atomların yoğunluğunca görüntüyü örnekler. Görüntü dijitalleştirilirken seste olduğu gibi yine bir ızgaranın köşelerine denk gelen noktaları ölçer ve sınırlı sayıda hanesi olan sayılar olarak kaydederiz. Ancak bu kez örneklediğimiz bilgi iki boyutludur, yani örneklem noktaları iki boyutlu bir ızgaradır. Ölçümün kendisi için üçüncü bir boyut eklemek gerekir. Yine seste olduğu gibi ızgaranın örnekleme karşılık gelen boyutlarındaki sıklığı ve ölçümleri kaydederken kullandığımız hane sayısını arttırarak görüntü dijitalleştirme kalitesini arttırmiş ve eski filmli fotoğraf çekme sistemine yakın bir kalite elde etmiş oluruz. Görüntüler için örneklem sıklığına aynı seste olduğu gibi çözünürlük diyoruz. Örneklem ızgarasının iki boyutu hemen her zaman eşit aralıklarla bölünür, o yüzden çözünürlükten bahsederken bu iki boyutun ızgara sıklığı ayrı ayrı zikredilmez. Onun yerine, örneğin fotoğraf baskısı yapabilen bir yazıcı için çözünürlük 600 dps (İng. dots per inch, inç başına nokta) olarak ifade edilir. Dijital fotoğraf makinelerinde ise çözünürlük 10 megapiksel gibi bir tek terimle ifade ediliyor, ki bu da ızgaradaki toplam nokta sayısını gösterir, iki boyutlu örneklem ızgarasının yatay ve dikey boyutları ayrı ayrı ifade edilmez ancak makinelerin teknik belgelerinde bulunabilir. Görüntü sözkonusu olduğunda bu örneklem noktalarına piksel adı veriliyor.

Genel olarak görüntü çözünürlüğüyle ilgili bir terim karmaşası vardır. Çünkü ses kayıtlarını normalden daha yavaş yada hızlı çalmak pek adet olmadığı halde görüntüleri duruma göre büyütüp küçültüyoruz. Bu durumda da inç veya santimetrekare başına nokta/piksel sayısı gibi bir çözünürlük ölçüsü tamamen anlamını yitiriyor, çünkü ızgarayı büyütüp zumladığınızda santimetrekare başına daha az piksel düşecektir. Bu yüzden büyük bir reklam panosu için görsel hazırlayan grafik sanatçısını normalden çok daha yüksek çözünürlükte dijital görüntülere ihtiyaç duyar.

Görüntü çözünürlüğünün diğer bir boyutu ise ızgaranın üçüncü boyutu, yani ölçümün kodlamasında kullanılan hane sayısıdır. Görüntü sözkonusu olduğunda bu ölçüm tek bir rakam değildir. İnsan gözü üç temel renge karşılık gelen retina hücrelerine sahiptir: kırmızı, yeşil, ve mavi (İng. red, green, blue, veya kısaca RGB). Yani bir rengin tonunu ifade etmek için üç sayı gerekir. Çoğu

zaman renk parlaklığı da dördüncü bir sayı ile ifade edilir. Yaygın olarak bu dört sayının herbiri bir byte ile kodlanır, ki bu 2^{8x4} farklı renk tonunu ayıretmeye imkan verir. Bu da insan gözü için gerçeğine çok yakın bir etki yaratır. Örneklem çözünürlüğü ise cihazdan cihaza değişir. Örneğin evinizdeki TVler 512x512 piksel gösterir. Bilgisayar monitörleri tipik olarak 1500x1500 piksel civarıdır. Yeni tip bir dijital fotoğraf makinesi ise 3000x3000 piksel civarıdır (yaklaşık 10 megapiksel). Her pikselin de dört byte ile kodlandığı düşünülürse bu 10x4=40 megabyte civarı eder. Böyle bir görüntü dosyası çok fazla yer kaplayacağı için aynı seste olduğu gibi matematiksel sıkıştırma teknikleri kullanılır, yaygın JPG formatı gibi.

Görüntü dijitalleştirmesini anlattıktan sonra hareketli görüntüleri, yani dijital videoları anlamak kolaydır. Aynı eski usül sinema filmlerinde olduğu gibi saniyede 24 veya daha fazla fotoğraf çekip arka arkaya gösterdiğinizde insan gözü onu akıcı bir görüntü olarak algılar. Dijital filmlerde de aynen bu teknik kullanılır. Görüntüye eşlik eden sesin ise ayrıca, yukarıda tarif ettiğimiz gibi kaydedilmesi gerekir. Bu yüzden dijital videolarda görüntü ve ses çözünürlüğü ayrı ayrı ölçülür. Yine dijital videolar çok yer kapladığından bunun için de MPG gibi sıkıştırma yöntemleri yaygın olarak kullanılır.

3.4 Dijital Dünyanın uzaysızlığı: Internet üzerinden dijital içerik aktarımı

Uzun zamandan beri ses ve görüntüleri çok uzağa taşıyabiliyoruz. Radyo yayınları ve telefon yüzyılı aşkın azamandır, Televizyon yayını ise yüzyıla yakın zamandır yapılıyor. Ancak bu yayınlar uzaklıktan etkilernir, yayının merkezinden uzaklaştıkça sinyal seviyesi düşer, parazit oluşur. İlginçtir ki bundan etkilenmeyen ve hepsinden eski bir teknoloji var: telgraf. Telgrafta kullanılan morse kodlaması dijital bir kodlama olduğundan uzaklıktan etkilenmezi. Çünkü mesaj iletimi elektrik voltajının yüksek veya düşük olması, bunun uzun veya kısa sürmesi ayırt edilmek suretiyle gerçekleşir. Yüksek voltaj parazit yüzünden biraz fazla yüksek veya azıcık düşük kalsa bile mesaj eksiksiz olarak karşı tarafa ulaşır. İnternet üzerinden veri iletimi de benzer esaslara dayandığından bozulmaya karşı dirençlidir. Tek farkla ki bunu telgrafa göre çok yüksek hızlarda yapıyoruz; ancak prensip aynıdır.

Böylece bir kez dijitalleştirilen, 1 ve 0lardan oluşan bir dizi

sayı olarak ifade edilen ses ve görüntüler çok uzaklara bile hatasızca gönderilebiliyor. Burada dijital bilginin uzak noktalara aktarmaya dirençli oluşunun sebebi sayıların mutlak bir kesinlikle ve basit, aktarım parazitlerine dirençli bir kodlamayla kopya edilebilmesidir.

3.5 İçerik güvenliği sorunu ve şifreleme

İçeriği gizli belgeler sağlam bir dolaba kilitlenebilir. Ancak dijital içerik, hele ki İnternetin her türlü kurum ve kişi tarafından bu kadar yoğun kullanıldığı bir dünyada bir dolapta tutulamaz. Bu yüzden ister metin dosyası, ister ses veya görüntü olsun, içeriğin hem depolanırken hem de İnternet üzerinden iletilirken istenmeyen kişilerin eline geçse bile anlaşılamayacak bir hale sokulması, ve istenilen ellerde bu işlemin tersine çevrilebilmesi arzu edilir. İnternet bankacılığı veya e-ticaret ödeme işlemleri gibi hassas işlemler de benzer bir durumla karşı karşıyadır.

Bunu sağlayan yöntemlerin genel adı kriptografi, yani şifrelemedir. Eski zamanlardan beri özellikle askeri amaçlarla kullanılan şifreleme yöntemlerin hemen tamamı günümüz bilgisayarlarıyla kolayca çözülebilecek türdendir. Daha güncel bir kısım yöntemlerin esası dijital içeriğin her bir byte değerinin bir şifre esas alınarak başka bir byte değeriyle değiştirilmesine dayanır. Aynı şifreye sahip başka biri, başka bir yerde bu işlemi tersine döndürebilir. Ancak İnternet iletişiminde böyle şifrelerin önceden belirlenmesi ve gizli kalması da başlıbaşına bir sorundur. Daha yakın zamanda geliştirilen bazı şifreleme teknikleri ise bu tür sorunları hafifletmiştir.

Ne var ki sayıların böyle karmaşık matematikselişlemlerden geçirilmesine dayalı kriptografi tekniklerin büyük bölümü günümüzün cep telefonları gibi görece daha zayıf CPU'ya sahip cihazlarının kapasitesini aşar. Masaüstü bilgisayarlarda dahi örneğin sesli veya görüntülü görüşmenin şifreli yapılması zordur. Bu düzeyde bir güvenlik ancak askeri amaçla yapılmış, özel çiplerle güçlendirilmiş cihazlarla yapılabiliyor. Hem bu tekniklerin hem de kullandığımız cihazların kapasitesinin yakın zamanda herkesin kullanabileceği noktaya gelmesine kesin gözüyle bakabiliriz. Ancak hem şifreleme tekniğinin zaafları vardır ve bilgisayarların hızlanması şifrelemenin kırılmasının da daha kolaylaşacağı anlamına geliyor.

Bölüm 4

Modern Bilgisayarların Yapısı: Karmaşık Bileşenlerden Kullanışlı Sistemler

4.1 Bilgisayarın evrimi: Yekpare tasarımlardan modüler bir sanayiye

Bölüm 2'de ikinci Dünya savaşı ve sonrasında yapılan ilk bilgisayarların ne kadar hantal ve büyük olduğundan, ve 1950'lerde transistör teknolojisinin gelişmesiyle bilgisayarların küçülmeye başladığından söz etmiştik. Bilgisayar teknolojisi aynı bir zamanlar buhar makinesinde olduğu gibi muazzam yenilikler ve ilerlemeler vadediyordu. Bu yüzden özel sektörün bu teknolojiyle ilgilenmesi hiç gecikmedi. IBM başta olmak üzere HP, DEC gibi firmaların ürettiği kocaman ve pahalı makinelerin ilk müşterileri başta soğuk savaş döneminde gelişmiş saldırı ve savunma sistemleri geliştirmeye çalışan ABD ordusu ve NASA olmak üzere, uzun bordro listeleriyle boğuşan büyük kamu kurumları ve dev özel sektör şirketleri, önde gelen büyük bütçeli Amerikan üniversiteleri gibi sayılı kurumlardı. Teknolojinin zamanla ucuzlamaya başlamasıyla kullanım alanı genişledi.

Ancak 1940ların sonundan 1980lerin başına kadar olan dönemde değişmeyen şey bilgisayarların monolitik, yani yekpare tasarımlarıydı. Bu yekpare tasarım tek bir şirket içerisinde yapılıyor, donanımın tüm parçaları aynı yerde üretilip birleştiriliyor, ve müşterilerin sistemi kullanması için gerekli yazılımlar da yine aynı yerde üretiliyordu. Aslına bakılırsa dikkatlerin makineye, yani donanıma yoğunlaştığı bu dönemde bilgisayar programılarına ciddi bir ticari değer atfedilmemiş olmalıdır ki bunlar üniversite, kamu kurumları ve özel sektördeki programcılar arasında serbestçe değiştokuş ediliyor ve kısmen ortaklaşa geliştiriliyordu (McKusick, 1999). İlk yıllarda sıkıntı kaynağı olan program yükleme ve sonuçları görme işlemleri zamanda gelişen monitörler, kartlarla program yükleme teknikleri ve –ortaklaşa geliştirilen– kullanışlı programlama dilleri sayesinde kolaylaşacaktı. Yine de yekpare tasarım döneminin gidişatı genel olarak sıkıntılı olacaktır. Örneğin dönemin IBM üst düzey yöneticilerinden Peter Brooks'un (1995) anılarında aktardığı tecrübeler gitgide karmaşıklaşan yazılım katmanını geliştirmenin yekpare model ile içinden çıkılmaz bir iş haline geldiğini, ölçeklenemediğini gösteriyor.

Her endüstride olduğu gibi bilgisayar endüstrisinin de doğumunu takip eden yıllarda bir uzmanlaşma dönemi olmuştur. Müşteri sayısı artar ve sistemler daha kullanışlı olabilmesi için gitgide daha karmaşık hale gelirken, kimi parçaların üretiminde uzmanlaşmış şirketler, ve alt-sektörler doğmaya başlar. Bunların ilki İntel firmasının başı çektiği mikro-işlemci (CPU) sektörüydü. Ancak sonuçları açısından daha ilginç bir gelişme uzakdoğuda, özellikle de Japonya'da mikro-elektronik sektörünün gelişmesidir (Langlois, 1990). Önce Amerikan yapımı mikroçipleri kullanarak hesap makinesi gibi küçük ürünler pazarına giren uzakdoğu şirketleri daha sonraları hem mikroçip pazarında önemli gelişme göstermişler hem de son kullanıcı donanım ürünlerinde (kişisel bilgisayar, yazıcı, bankamatik cihazları, cep telefonu, vb.) günümüzde zirveye ulaşan bir başarı grafiği sergilemişlerdir.

IBM gibi firmalar yekpare tasarım döneminin sonlarına doğru tasarımlarında kullandıkları parçaların önemli bir bölümünü özellikle uzakdoğuda ortaya çıkan ve son derece cazip fiyatlar sunan tedarikçilerden alıyorlardı. Ancak 1980lere gelinirken piyasa yapısında önemli bir değişiklik gerçekleşiyordu: kişisel bilgisayarların ortaya çıkışı. Yaşı büyük olanlarımızın hatırlayacağı Commodore gibi 'oyun bilgisayarı' sınıfı cihazlar birçok eve girecek kadar

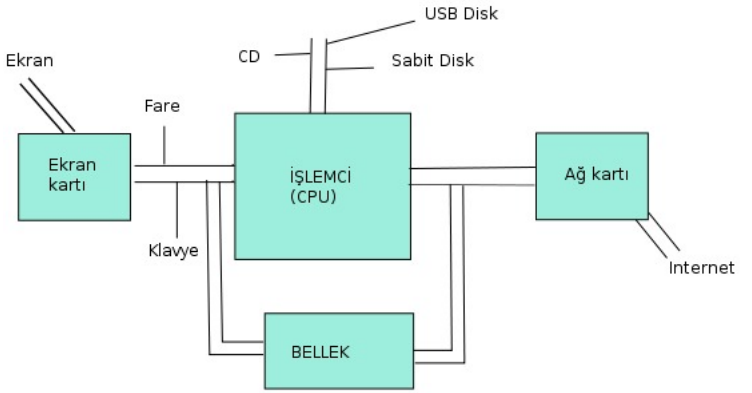
ucuzdu, ve sadece Amerika'dan değil dünyanın farklı yerlerindeki şirketlerden bu sınıfta birçok ürün ortaya çıkmaya başlamıştı. Kurumsal bilişim pazarı açısından çok anlamlı gözükmeseler de bu ürünlerin ortaya çıkışı ve evlere girişi IBM gibi pazara yön veren dev şirketleri alarma geçirmiş olmalıdır.

IBM'in bu baskıya cevabı son derece yaratıcı ve hem sektörün hem de firmanın kendisinin önünü açacak türdendir: kişisel bilgisayar, yani PC (İng. personal computer). IBM bu dönemde ISA (İng. Industry Standard Architecture, Endüstri Standardı Mimari) standardını tasarlayıp sanayideki diğer oyuncularla paylaşır. ISA standardı daha sonraları anakart (İng. Motherboard) olarak anılan, farklı üreticilerden alınmış IBM uyumlu parçaların üzerine takılmasına uygun bir donanım tasarımına dayanıyordu. Böylece IBM'den PC satın alanlar kendi ihtiyaçlarına uygun parçaları başka firmalardan temin edip PC'deki anakartın üzerine ekleyerek istedikleri özellikte bir bilgisayar elde edebiliyordu.

IBM'in açık standartlara dayalı bu stratejisi son derece canlı, ihtiyaca göre esneyebilen bir kişisel bilgisayar pazarının oluşmasına katkı sağlamıştır. Parça uyumluluğu sayesinde hem Dünya ölçeğinde pazarın işleyişi kolaylaşmış, hem de müşterinin isteğine uygun bir bilgisayar sistemi oluşturmasına izin verdiği için 'IBM uyumlu' tabir edilen PCler bilgisayar piyasasının en hızlı büyüyen ürün pazarı oluvermişti. Açık standartlar konusunda bu başarılı deneyimi birçok firma örnek alacaktır. Zamanla bazılarının ismini duymuş olabileceğiniz ATA, SATA, IDE, PCI, SCSI gibi birçok açık standart ortaya çıkacaktır. Son olarak USB standardı da bilgisayar piyasasındaki bu başarılı geleneğin devamı niteliğinde. Bu standartlar ister bir büyük firma tarafından tasarlanıp açık ve şeffaf bir süreçle piyasaya sürülmüş olsun, ister firmalararası bir konsorsiyum veya bağımsız bir meslek örgütü tarafından geliştirilmiş olsunlar hep aynı soruna çözüm getiriyorlar: ürün uyumluluğu. Yekpare tasarımdan böyle modüler bir tasarıma geçiş uzakdoğudan (Japonya, Tayvan, Singapur, vb.) ABD'ye uzanan pazardaki ürünlerin modüler biçimde birarada kullanılmasına, son kullanıcı için uyumlu bir çeşitlilik oluşmasına, ve dolayısıyla pazarın büyümesine izin veriyordu.

4.2 Modern bir bilgisayar sisteminin bileşenleri

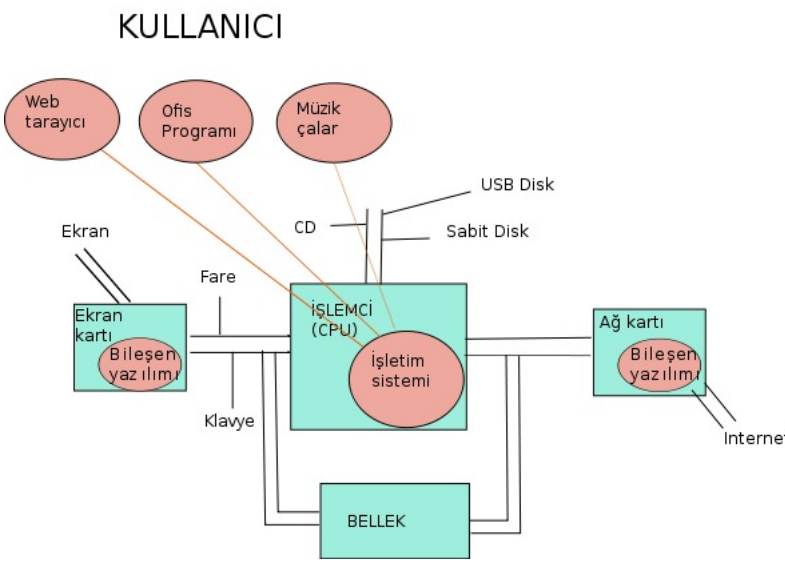
Günümüzde bu standartları kullanan bir bilgisayarın iç yapısı bir otoyol ağıyla birbirine bağlanmış, yarı özerk bir yönetime



Şekil 4.1: Bilgisayarın temel anatomisi

sahip mahalleleri andırır (Şekil 4.1. Veri yolu denilen bu bilgi otoyolu anakartın yüzeyi boyunca yayılır ve mahalleler arasında yüksek hızlı veri seyahatine izin verir. Aslında veri akışının hemen tamamı cihazlar arasında değil cihazlarla bellek ve işlemci arasındadır. Bu yüzden mahalleleri bağlayan bütün yollar şehir merkezinden, yani işlemci ve bellekten geçer. ISA ve USB gibi standartlar da aynı trafik kuralları, yol ve araç standartlarında olduğu gibi bu veri akışının sorunsuz ve hızli olmasını sağlıyor. tek farkla ki bilgisayarın parçaları sözkonusu olduğunda herkes kurallara uyuyor ve neredeyse hiç kaza yaşanmıyor. Yine de sistemin işleyişi için trafik polislerine ihtiyaç olmaktadır. Şehir merkezinde bu işi ana işlemci üstlenir. Bunu da işletim sistemin denilen temel yazılımın koyduğu kurallar çerçevesinde çalışarak yapar. Ayrıca parçaların kendi içinde de mahalli veri yolları vardır; eğer bir ağ veya ekran kartı elinize geçerse bu yolları bakır kanallar olarak görebilirsiniz. Mahalli trafik sistmin diğer parçalarını etkilemediğinden dolayı bu trafik herkartın üzerindeki işlemci çipler tarafından koordine edilir ve o çipe özel kurallar uygulanır, ki bu esas itibarıyle sistemin genel işleyişini anlamak konusunda bizi ilgilendirmiyor.

Böylelikle modern bi bilgisayra tek bir elde tasarlanmış bir nesneden çok bir lego oyuncağın serbestçe birbirine eklenmiş parçalarına benzemektedir. Bilgisayar sistemi bağımsız geliştirilmiş, ancak standat bir veriyoluna bağlanmaya uygun yapılmış



Şekil 4.2: Bilgisayarın bileşenlerinin federatif yapısı

sistemlerin federasyonudur. Bu federasyonun bileşenleri belirli konularda merkezi otoriteye mutlak itaat ederler ve böylece koordineli çalışırlar. Ancak bunun ötesinde hem merkezi otoritenin hem de federasyonu oluşturan parçaların belirli bir otonomisi vardır. Bu parçalar kendi işlemci çiplerine yüklü programlar tarafından yönetilirler (Şekil 4.2). İşin bu tarafı biz kullanıcıları hiç ilgilendirmez. Bizim kullandığımız yazılımlar sadece işletim sistemiyle muhatap olur. Diğer herşey, özellikle parçaların koordinasyonuyla ilgili bütün sorumluluk işletim sistemi tarafından (örneğin Microsoft Windows, Ubuntu Linux, MacOS, Android, vb.) üstlenmiştir. Bu yüzden bilgisayara yeni bir parça takılırsa işletim sistemine bu parçayla konuşup koordine olabilmesi için bir sürücü program (İng. device driver) eklenmesi gerekir. Bundan sonrasında bütün detayları bizim için işletim sistemi halledecektir. Bağladığımız parça ister bir USB disk, isterse süper hızlı ve büyük bir sabit disk olsun işletim sistemi onları bize bir dosya dizini, saklama alanı olarak gösterecek ve benzer şekilde kullanmamızı sağlayacaktır. Veya görüntü kartı ister basit ister süper yüksek

çözünürlüklü olsun işletim sistemi program pencerelerinin her türlü ekrana aynı şekilde yansıtılmasına olanak verir. Unix/Linux ailesi işletim sistemlerinde bu soyutlama öyle bir noktadadır ki her türlü cihaz bir dosya olarak kullanılabilir. Örneğin ses kartından ses çıkartmak o cihaza karşılık gelen dosyaya dijital ses bilgisini yazmakla yapılabilir, vs.

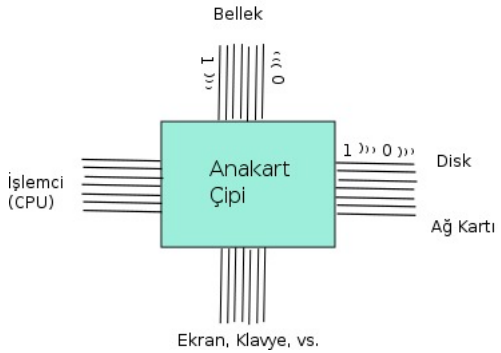
Sistemin farklı özellikteki parçaları arasındaki benzerlikleri böyle soyutlamalar ile bize sunması ve gereksiz detayları gizlemesi sayesindedir ki bizler kullanıcı veya programcı olarak bilgisayarlara gitgide daha iyi hükmedebiliyoruz. Yine bu sebeptendir ki işletim sistemi denilen yazılım son derece kilit bir konumdadır.

İç yapısı giderek karmaşıklaşan bilgisayarlar sistemi birarada tutan standartlar ve karmaşayı örten işletim sistemi sayesinde daha ilkel atalarından çok daha kullanışlı olabilmişlerdir. Yine de onları daha iyi kullanmak, neden bazen iyi bazen kötü ve yavaş işlediklerini anlamak için sistemin iç işleyişine bir göz atacağız.

Bilgisayarın işleyişi için birtakım temel işlevler gerekir. Öncelikle sayıların, yani 1 ve 0lara karşılık gelen elektrik sinyallerinin bilgisayarın parçaları arasında çarpışmadan kazasız belasız gidip gelmesi ve aritmetik işlemlerin yapılması gerekir. Ayrıca sayılar hatırlanabilmeli, bir yerlerde saklanabilmelidir. Bu sayılar yazı, ses, görüntü olarak kullanıcıya iletilmelidir. Bütün bunların yapılmasını sağlayan parçalar bilgisayarın mekanik, fiziki yapısını oluşturur ki bunların toplamına *donanım* diyoruz (İng. hardware). Aşağıdaki bölümde modern bir bilgisayar donanımının ana parçalarını inceleyeceğiz.

Sözkonusu olan, örneğin çamaşır makinesi gibi tek işlevli bir makine olsaydı makinenin mekanik çizimine bakarak üç aşağı beş yukarı nasıl işlediğini anlayabilirdik. Oysa bilgisayar gibi birçok farklı ve karmaşık işi yapan bir makine sözkonusu olduğunda işleyişi anlamak için bundan fazlası gerekiyor. Parçaların hareketini yönlendiren şey bilgisayar programları, yani yazılımlardır. Yazılım bir orkestra şefi gib enstrümanları/cihazları yöneterek ortaya anlamlı bir iş çıkmasını sağlar. Bunu ayrı bir bölümde inceleyeceğiz.

ISA ve USB gibi standartlar da aynı trafik kuralları, yol ve araç standartlarında olduğu gibi



Şekil 4.3: Anakart üzerinde veriyolu.

4.3 Donanım: Sistemin mekaniği

Bilgisayar diğer makinelerden ziyade düşünen bir varlık olan insanla karşılaştırılagelmiştir. O yüzden bu bölümde bilgisayarın fiziki yapısını incelerken bazı uyumsuzluklara rağmen insan bedeni ile benzeştirmeler kurarak anlatacağım. Bunun sebebi makineyi insanla bir tutmak niyetim değil, ama hesap, karşılaştırma, hatırlama gibi diğer makinelerden çok bize benzeyen işlevlerin az da olsa bize benzeyen organlarda karşılık bulmasıdır.

4.3.1 Sinir sistemi: Anakart, veriyolu ve RAM bellek

Bilgisayraın veriyolundan biraz önce bahsetmiştik ve standartlar sayesinde bu veriyoluna parçaların kolayca eklenebildiğini söylemiştirk. Veriyolunun merkezinde veri trafiğini koordine eden bir işlemci çip bulunur (ana işlemciden farklı bir çip). Bilgisayarın tüm parçaları (ana işlemci dahil) arasındaki veri akışı bu veriyolu üzerinden ve anakart çipinin polisliğinde gerçekleşir. Bir anakart üzerinde görebileceğiniz veriyolu çok sayıda kanaldan, birbirine paralel ilerleyen iletken bakır yollardan oluşur.

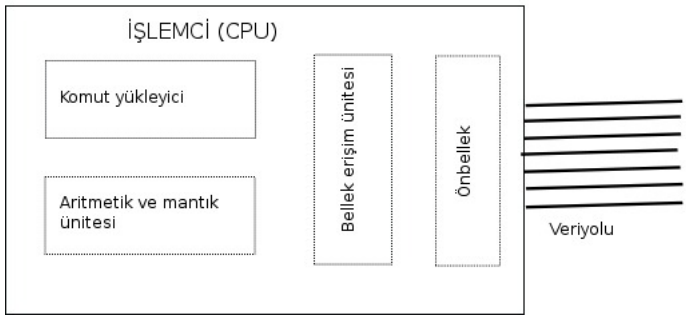
Şekil 4.1'de kabaca verilen yol yapısını daha gerçeğe yakın olarak şekil 4.3'de görebilirsiniz. Bu kanalların/yolların yarısı verinin hedef adresini, diğer yarısı belleğe yazılacak veya okunmuş sayıyı, ve bir kanal da istenilen işlemin okuma mı yazma mı olduğu bilgisini taşır. Böylece bilgisayarın içindeki parçalar diğer parçalara veri gönderebilir veya onlardan veri okumak istediğini belirtebilir. Karışıklık olmasın diye bu parçaların herbirine ad-

resler ayrılmıştır, ancak veri alışverişi çoğu zaman diğer parçalar ile RAM, yanı uçucu ve hızlı erişimli bellek çipleri arasında gerçekleşir. Veri alışverişinde kullanılan adreslerin büyük bölümü RAM'e tekabül eder, ancak kimi zaman ekran kartı gibi parçalara doğrudan veri göndermek gerekebilir. Örneğin bilgisayar oyunlarının güçlü bir görsel deneyim gerektiren koşulları oyun programı ile ekran kartı arasında RAM'den geçişin getireceği azıcık yavaşlamaya bile izin vermez. Bu yüzden genel işleyişe ters ve istisnai bu tür durumlarda parçalar RAM üzeinden veri paylaşmak yerine doğrudan iletişim kurabilirler (yeter ki buna izin veren bir adres ayrılmış olsun).

Bilgisayarın sinir sisteminin işleyişinde rol alan önemli bir unsur veriyolundaki kesme işaretini taşıyan ayrı bir kanaldır. Anakart çipi gerekli durumlarda merkez işlemci çipe bir kesme sinyali göndererek dikkatini çekmeye çalışır. Örneğin siz klavyenin tuşlarına bastığınızda veya fare'yi hareket ettirip tıkladığınızda bu eyleminiz önce klavye kablosu sonra veriyolu üzerinden geçerek anakart çipine ulaşır, sonra da kesme işareti ile her daim pek meşgul olan merkezi işlemcinin dikkati bu olaya çekilir, bu şekilde size tepki vermesi sağlanır. Bu kesme (İng. interrupt) mekanizması olmasaydı merkez işlemci çipinin sürekli paranoyak bir şekilde bütün cihazların durumunu dönüp dönüp kontrol etmesi gerekirdi, ki o zaman hiçbir işini tam göremezdi.

4.3.2 Kalp: anakart çipi

Yukarıda bahsettiğimiz veriyolu polisliği yapan anakart çipinin önemli bir rolü daha vardır. Bu çipten verilen saat tiktakları bütün parçaların uygun adım çalışmasını sağlar. Deyim yerindeyse bu saat sinyali bilgisayar için kalp atışı gibidir. Bu kalp atışının temposu hem ana işlemcinin hem de diğer parçaların yetişebileceği bir tempo olmalıdır. Öte yandan olabildiğince de hızlı olması arzu edilir ki bilgisayar kullanıcısına iyi bir performans versin. Anakart çipi bu ikisi arasında uygun bir kalp atış hızına ayarlanmıştır. Bazı sistemler bu atış hızını (İng. clock frequency) sizin ayarlamanıza izin verir, ama bunun sonucu cihazın fazla ısınması veya olabilecekten daha düşük performans vermesi olabilir. Bazı modern sistemler de çiplerin çalışma yoğunluğu ve ısısına göre bu kalp ritmini dinamik olarak ayarlıyorlar.



Şekil 4.4: Ana işlemci çipin iç yapısı.

4.3.3 Beyin: ana işlem çipi

Ana işlemci modern bir bilgisayarın içindeki tek işlemci çip değildir. Ama tek başına diğer bütün çiplerden (örneğin anakart çipi, ekran kartı çipi, disk sürücü çipi, vb.) daha fazla işlem gücüne sahiptir, ve bütün işlerin merkezi kontrolü ona aittir. Bu yüzden bilgisayardaki diğer çipleri vücüdumuzun organlarındaki sinir düğümlerine benzetirsek, ana işlemci çip te beyine karşılık gelir. Anakart üzerine baktığınızda en büyük ve çok sayıda bacağı olan bir çip olarak onu görebilirsiniz.

Ana işlemcinin önemli bir becerisinin sayılarla işlem yapabilmek ve onları karşılaştırabilmek olduğundan bölüm 2'te bahsetmiştik. Bu yüzden ana işlemci çipin bir bölümü matematik ve mantık ünitesi adı verilen, aritmetik ve karşılaştırma işlemleri yapan bölümdür (Şekil 4.4). İşlemcinin bu ve benzeri bölümleri göremeyiz; bunlar çipin içine lazer teknolojisi ile yakılarak oluşturulmuş ve üstü kapatılmış hassas parçalardır. Çok sayıda direnç ve transistör içerdiklerinden küçük bir çipe göre oldukça fazla elektrik enerjisi kullanılır ve bu yüzden üzerine soğutucu bir metal parçası ve fan konulur.

İşlemcinin temel görevi kullanıcının dilediği programların çalıştırılmasıdır. Bu sebeple iç parçalardan bir kısmı program komutlarının sırayla yüklenmesi, işini gören komut yükleyici bölümüdür. Hem komutlar hem de bu komutların çalıştırılması sonucu üretilen sayılar RAM bellekte depolanırlar. Merkezi işlemcinin RAM'e erişimi de ayrı bir ünite tarafından koordine edilir.

Merkezi işlemcinin çalışma hızı oldukça yüksektir ve bu ça-

lışma sırasında bellekle veri alışverişi (anakart çipinin başarılı trafik polisliğine rağmen) yavaşlatıcı bir undur olur. Bu sebepten dolayı RAM belleğin o an için sık kullanılan yerlerinin bir kopyası ana işlemci çipin içindeki önbellek alanında bulundurulur (bilgisayar reklamlarında 'CPU cache' olarak anılan bellek). Nu önbellek asıl RAM'e göre çok daha küçüktür çünkü işlemciye ayak uyduracak bir hızda çalışır ve bu yüzden biraz pahalıya malolur. Buna karşılık tipik olarak RAM'in çalışma hızı ana işlemcinin sadece beşte biri kadardır. Satın aldığınız bilgisayarın işlemcisindeki önbellek miktarı, yaptığınız işe bağlı olarak bilgisayarın performansında önemli bir etken olabilir.

Aynı anda birden fazla işle meşgul olma Modern bir ana işlemci aynı bizim beynimiz gibi birden fazla meseleyle uğraşabilir. Gerçekten de bilgisayarımızda aynı anda birkaç program birden çalıştırırız. Fakat beynimizin aksine merkezi işlemcinin bu becerisi 'esastan' değildir. Aslında işlemci aynı anda sadece bir işle uğraşabilir, fakat hız avantajını kullanır ve çok sık aralıklarla, dönüşümlü olarak dikkatini bir işten diğerine yönelterek herbirinin kesintisiz yürüdüğü izlenimini verir. Bütün bu sistemin çalışmasında işletim sistemi denilen temel programa büyük iş düşer, ki bunu ileriki bölümlerde inceleyeceğiz. Merkezi işlemci bu işlerden herhangi birine dalıp gitmemek için sistemin temposunu tutan anakart işlemcisinin saatinden faydalanır ve alarm kura. Böylece bir işe başlarken, örneğin 0.1 saniye sonraya alarmını kurar, alarm çalınca ikinci işe geçer ve yine alarmını kurar, vs. Böylece aynı anda dörtbeş kişiyle birden oynayan bir satranç usatası gibi sırayla her işle biraz ilgilenir ve bize hepsinin birden kesintisiz yapıldığı izlenimi vermeyi başarır.

Birden fazla işlemcili veya çift/dört çekirdekli bilgisayarlar Sürekli daha hızlı bilgisayarlara ihtiyaç duyuyoruz. Ancak elektronik teknolojisinin fiziki sınırlamaları sebebiyle bir işlemci ancak belirli bir hızda çalıştırılabilir. İstenilen hız artışını sağlamanın yollarından biri bir anakartın üzerine birden fazla ana işlemci çipi koymaktır. Tabi bunu yaptığımızda ek işlemcilerin herbirinin sisteme entegre olabilmesi için veriyolunda yeni yol bağlantıları açılması gerekir ve anakart oldukça sıkışır. Ayrıca birden fazla işlemcinin yarattığı veri trafiğini destekleyecek beceride anakart işlemcileri ve veriyolu tasarımları gerekir. Bu

yüzden birden çok ana işlemcili anakartların yapımı zor ve pahalı, konulabilecek ek işlemci sayısı sınırlıdır. Ayrıca böyle bir sistem iki kafası olan Siyam ikizlerine benzer. İki beynin beraber çalışması büyük bir problemdir ve işletim sistemi denilen temel programın da bu beyinler arasındaki eşgüdümü sağlayacak şekilde inşa edilmesi gerekir. Herşeye rağmen bu tür sistemler başarıyla üretilmiş ve tipik olarak bir anakart üzerinde dört ana işlemci çipe kadar çıkabilmiştir. Yine de böyle bir sistemde beyin dışındaki organların performansı genel performansı etkilediğinden sonuçtaki hız artışı dört kat değil daha düşük olacaktır.

Hız artışı talebini karşılamak için ana işlemci üreticilerinin son yıllarda geliştirdikleri bir teknoloji ise çok çekirdekli işlemcilerdir. Bu işlemcilerde bir ana çipin içine iki veya dört tane aritmetik ve mantık ünitesi sığdırılır ve bunlar paralel çalışırlar. Böylece sistemin aritmetik işlem yapma hızı katlanmış olur ama öte yandan anakart ve veriyolunun çok az değişiklikle, sıkış tepiş bir hale gelmeden çalışması mümkün olur. Son yıllarda dual core (çift çekirdekli) işlemciler oldukça yaygınlaştı. Yine çok işlemcili sistemlerde olduğu gibi çok çekirdekli işlemci kullanan sistemlerde de performans artışı çekirdek sayısındaki artıştan daha düşük olacaktır, çünkü ana işlemcinin çekirdekleri (aritmetik ve mantık üniteleri) çip içindeki program yükleyici, bellek erişimi modüllerini ve önbelleği paylaşırlar.

Piyasada satın alınabilecek yüksek performanslı bilgisayarlarda her iki yöntem de kullanılmaktadır. Örneğin dört ana işlemci çipi olan ve bu çiplerin herbiri dörder çekirdekli olan bir bilgisayar bulabilirsiniz.

Bundan da öte bir hesaplama gücü gerektiğinde birden fazla bilgisayarı eşgüdümlü olarak işe koşmak gerekir. Bu apayrı bir programlama disiplinidir. Özellikle Unix/Linux işletim sistemi ailesi bu tür paralel çalıştırmayı kolaşlaştıran bileşenler sunar.

4.4 Duyular ve dış dünya ile etkileşim: Çevresel bileşenler

Nadir durumlar hariç bütün bilgisayarların bir kullanıcı ile iki yönlü iletişim kurabilmesi gerekir. Bu iletişim sesli, görüntülü, veya yazılı olabilir Örneğin bilgisayar yapılan işin sonuçlarını sesli olarak kullanıcıya bildirebilir (ki görme engelli biri için en iyi yol bu olacaktır), veya istenilen müzik parçasını çalabilir. Bunun tersi olarak kullanıcı bilgisayara sesli komut verebilir

(ki hareket engelli bir kullanıcı için bu gerekecektir). Görsel iletişimde bilgisayar bize resimleri veya hareketli görüntüleri monitörde gösterir, ve fare aracılığıyla bir görsel simgenin üzerine tıklayarak diğer yönde iletişim kurabiliriz. Bunlara ilaveten klavye kullanarak bilgfisayara yazılı komutlar verebilir veya bir metni ekranda girip inceleyebiliriz. 1990'lardan itibaren yaygınlaşan grafik kullanıcı arayüzleri yüzünden insan-bilgisayar arasındaki yazılı komut yoluyla iletişim hemen tamamıyla ortadan kalkmış ve yerini fareyi ekranda oynatıp iki tuşunu tıklamaya kısıtlı bir görsel iletişime bırakmıştır. Yine de klavyeden yazılı komut vererek bilgisayarı yönlendirmek profesyoneller için zengin ve gerekli bir iletişim yolu olarak kalmıştır.

Bütün bu iletişim biçimleri bilgisayar anakartında bir bileşen aracılığıyla gerçekleştirilir. Sesli iletişim için sesleri sayılara ve sayıları ses dalgasına çeviren bir ses ses bileşeni vardır. Görsel iletişim için sayıları renklere dönüştürüp ekranda uygun noktalara yansıtan bir video/ekran bileşeni vardır, ki bunların bir kısmı bilgisayar oyunlarında ihtiyaç duyulan görsel performansı sağlamak için ek işlemcilerle ve bellekle donatılmıştır. Kalvye ve fare kontrolü de daha basit olan çevresel cihaz işlemci çiplerine bağlanmıştır. Daha eski yıllarda bu bileşenlerin herbiri ek bir sistem olarak, örneğinvideo veya ses kartı olarak alınıp anakarta takılıyorduç Şimdilerde elektronik teknolojisinin gelişimi ve bunların yaygınlaşmasıyla bu bileşenler küçülmekle kalmamış anakartın üzerinde sabit entegre parçalar haline gelmiştir. Örneğin bir dizüstü bilgisayar anakartında bunların herbiri için bir çip var ama ek bir kart takılı değil. Yine de müzisye grafik tasarımcısı, vb., profesyoneller için yüksek performans ve özelliklere sahip ses ve görüntü kartları satın alıp sisteme ekleme imkanı vardır.

Bunların dışında bilgisayarın başka bilgisayarlarla da iletişim kurması gerekir. Ev veya ofisimizdeki yazıcı, tarayıcı gibi aletlerin kullanılmasının yanısıra İnternet üzerinden uzak yerlerdeki bilgisayarlarla da iletişim gerekir. Aynı mekandaki cihazlarla iletişim doğrudan bir kablo bağlantısıyla yapılabilir, ki günümüzde bu iş standart olarak USB denilen bağlantı ve kablo standardıyla yapılıyor. Esas olarak bu bağlantı anakart üzerindeki soketlere parça eklemekten pek farklı değildir. İnternet iletişimi ise (ister aynı ofis içinde, ister uzaktaki bilgisayarlarla) ağ arayüzü ile yapılır. Bu bileşen de eskiden ayrı bir kart olarak yapılıyordu ancak günümüzde anakartın ayrılmaz bir parçası haline gelmiştir.

Kablosuz sistemler prensipte aynıdır, ancak radyo veya kızılötesi dalgalarla iletişim farklı güvenlik önlemleri gerektirir.

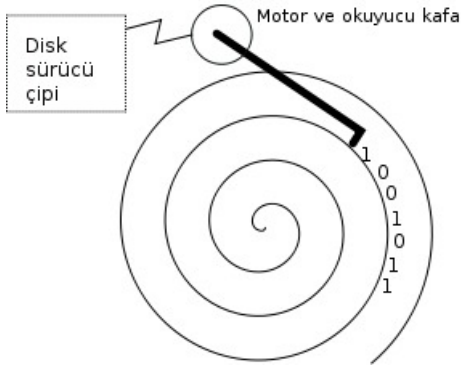
Bilgisayar anakartı bu duyu organlarının herhangi birinden bir duyum aldığı zaman bu durumu kesme sinyali aracılığıyla ana işlemciye iletir. Böylece ana işlemci işine biraz ara vererek bizim fare tıklamamız veya İnternetten beklenen verinin gelmesi gibi durumlarla ilgilenip gereğini yapar.

4.4.1 Kalıcı bellek ve veri coğrafyasının düzenlenişi: Sabit diskler ve dosya sistemleri

Hem RAM denilen bellek hem de ana işlemci çipinin ön belleği uçucu birer hafızadır, çünkü bilgiyi elektrik akımı halinde depolarlar. Bu yüzden bilgisayar kapanıp açıldığında içlerindeki herşey uçar gider. Öte yandan tam da elektriksel oldukları için son derece hızlı çalışırlar ve bilgisayarın hızlı işlemesi için gereklidirler. Ancak bunların dışında kalıcı bir bellek olmasaydı bilgisayar hıç hafızası olmayan bir insan misali her açıldığında şaşkın bir şekilde herşeye baştan başlardı.

Kalıcı bellek olarak çoğunlukla sabir diskler kullanılır. Bizim anakartın kenarında kasaya vidalanmış bir kutu olarak gördüğümüz bu bileşen içinde metal bir disk bulunur. Bu diskin yüzeyine kaplanmış olan manyetik malzeme elektrik akımına tepki veren türdendir. Elektrik akımı verilmek suretiyle şekillendirilir ve şeklini muhafaza edebilir, böylece 1 ve 0 değerlerinden (bit'ler) çok miktarda depolamak için kullanılır. Depolanana veriler zayıf bir elektrik akımı uygulanakara bozulmadan okunabilirler. Böylece bilgisayar kapatılsa bile manyetik malzeme verileri hatırlamaya devam eder. Bu yüzden sabit diskler kalıcı beller oklarak tabir edilirler. Esasen CD de benzeri bir bellektir, ancak sabit diskten farklı olarak verilerin değiştirilmesi imkanı yoktur (veya olsa da çok yavaş ve sıkıntılı bir iştir), o yüzden da farklı amaçla kullanılırlar. Son yıllarda yaygınlaşan flash bellekler de (örneğin USB bellek) farklı bir teknoloji kullanılmasına rağmen prensip olarak aynı işi görürler.

Bir sabit diske depolanan veriler manyetik yüzeye eşit olarak yayılmalı ve belirli bir düzeni olmalıdır. Bu yüzden depolanan bit'ler şekil 4.5'teki gibi bir spiral şeklinde diskin yüzeyine serilmiştir, ve disk döndürülürken hareketli bir manyetik kafa yardımıyla bu veriler okunur veya yazılır. Günümüzde tipik bir masaüstü bilgisayar diski 250 GigaByte (iki trilyon bit) civarında bilgi

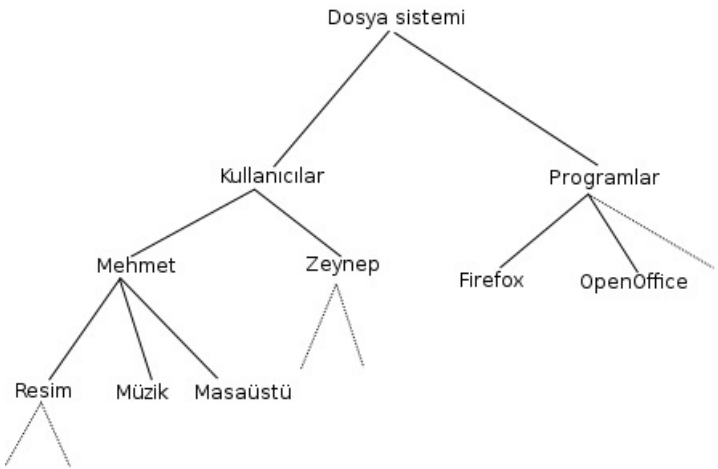


Şekil 4.5: Sabit disk üzerinde bit'lerin yerleşimi

depolayabilir.

Eğer sabit diski olduğu gibi kullanmaya kalkışsaydık bu ip gibi dizili milyarlarca biti düzenlemek çok zor olurdu. Dosya sistemi adı verilen son derece karmaşık bir veri yapısı sayesinde biz bu verileri hiyerarşik, anlamlı bir düzeni olan bir ağaç yapısına yerleşmiş dosyalar olarak kullanabiliyoruz. Bir dosya sisteminin bize sunduğu düzen bizim bilgileri tasnif etme şeklimize benzer, ve aşağı yukarı bilgisayarın 'başlangıç' menüsündeki düzeni yansıtır (Şekil 4.6). Bu düzen bir nevi arşiv odası veya kütüphane gibidir, nasıl ki bir kütüphanede koridorlar bölmelere, bölmeler raflara ayrılıyorsa, dosya sisteminde de böyle bir ana girişten çatallanarak tasnif edilmiştir. Ancak bir farkla ki kütüphanede bir raf dolup taştığı zaman yaşanan sınıntı burada yaşanmaz. Dosya sisteminde bu rafların karşılığı olan dizinler biz içine malzeme koydukça büyür. İçi kalabalıklaşan dizinlerde de daha iyi bir tasnif için alt dizinler açabiliriz. Sadece toplam sabit disk kapasitesi tün sistem için bir üst sınırdır.

Disk bölümleme Eğer bir dosya sisteminin üzerine oturduğu sabit disk fazla dolar ve yer azalırsa, aynı kafası şişmiş biri gibi iş göremez hale gelir veya çok yavaşlar. Bu gibi durumları kontrol altına almanın bir yolu diski iki veya daha fazla bölüme ayırıp her bir parçanın üzerine ayrı bir dosya sistemi oturtmaktır. Bu şekilde örneğin kişisel dosyalarımızı programlardan ayrı bir dosya sistemine koyabiliriz. Her bir dosya sistemi ancak üzerine

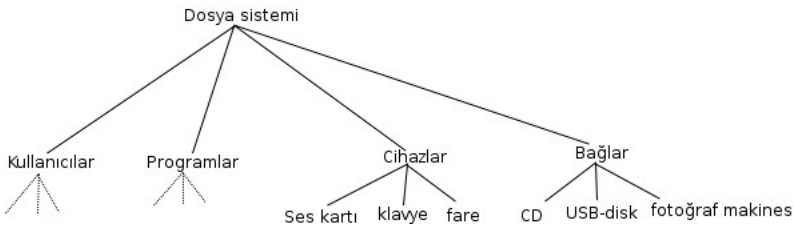


Şekil 4.6: Dosya sisteminin düzenlenişi

oturduğu disk bölümü kadar büyüyebilir, diğerlerinin alanına taşamaz. Böylece örneğin müzik ve filmlerimizi koyduğumuz dosya sistemi şişse bile sistemin kalanını etkilemeyecek ve işleyişi zora sokmayacaktır.

Disk birleştirme Disk bölümlemenin tam tersi olarak bazen bir dosya disteminin çok büyük olmasını isteriz, ve bunu yapmak ancak birden fazla sabit diski yanyana kullanarak mümkün olur. LVM (İng. Logical Volume Management, Mantıksal Hacim Yönetimi) tabir edilen modern bir metod bu şekilde birden fazla sabir diskin kapasitesinin tek bir dosya sistemi olarak kullanılmasını sağlar. Unix/Linux işletim sistemi ailesinde sistemin ücretsiz bir parçası olan bu özellik diğer çoğu işletim sisteminde de ücretli yazılımlar kullanarak elde edilebiliyor.

Dosya sistemlerinin farklılıkları Birçok dosya sistemi çeşidi vardır. Bunlardan bir kısmı belirli bir endüstriyel kullanım amacına yöneliktir. Örneğin CD'lerde kullanılan ISO-9660 böyle bir uluslararası standart dosya sistemidir. Ancak bunlar bir yana farklı bilgisayar işletim sistemi üreticisi firmaların geliştirdiği, vadettiği özellikler birbirine benzeyen birçok dosya sistemi vardır ve bunlar birbiriyle uyumlu değildir. Burada Microsoft'un kullandığı



Şekil 4.7: Linux'un bileşik dosya sistemi görünümü

dosya sistemleri ile akademik kökenli Unix/Linux ailesi işletim sistemlerinde kullanılan dosya sistemi tiplerini karşılaştıracağız.

Microsoft Windows işletim sistemi NTFS, FAT32 gibi firmaya özgü birkaç alternatif dosya sistemini destekler, ancak Linux ve Max sistemlerine özgü dosya sistemlerini desteklemez ve ek programlar olmadan okuyamaz. Bu yüzden fark sistemlerde kullanılacak USB disk gibi taşınabilir kalıcı bellek cihazları üzerinde çaresiz FAT32 dosya sistemi kullanılıyor, çünkü Microsoft'un ketumluğuna karşın diğer sistemler bu dosya sistemini destekliyor. Yine aynı sebepten ötürü değişik dosya tiplerini üç harfli dosya uzantısı (dosya adının noktadan sonraki kısmı) ile belirtmek adet olmuştur. Mesela düz metin dosyaları için İngilizce text, yani metin sözcüğünü kısaltması olarak txt uzantısı kullanmak gibi. Microsoft dosya sistemlerinin üç harften fazla dosya uzantısına izin vermemesinden kaynaklanan bu sınırlama Unix/Linux sistemlerinde de sistemin böyle bir kısıtlaması olmamasına rağmen kullanıcıların dosyalarını başka sistemlerde kullanırken sıkıntı yaşamaması için gözetiliyor.

Unix/Linux ailesi işletim sistemlerinde çok fazla sayıda dosya sistemi alternatifi vardır. Ancak bunlar bir yana Linux'un önemli bir farkı bilgisayarın veri coğrafyasının birleşik bir halde sunulmasıdır. Şekil 4.7'te gösterildiği gibi sistemin veri akışına konu olan bütün yönleri tek bir hiyerarşik ağaç yapısının altına yerleşir. Böylece sisteme takılan bir USB disk bu ağacın bir dalı olarak ekleniverir. Ayrıca klavye, ses kartı gibi cihazlarla olan veri alışverişi de bu ağacın altında belirli bir dosyaya okuma/yazma işlemi yapar gibi gerçekleştirilebilir.

Unix/Linux ailesi sistemlerin örneğin Microsoft'tan önemli bir diğer farkı da sistemleri güvenliğidir. 1970'li yıllardan bu yana

bu sistemler birden fazla kullanıcının paylaştığı büyük bilgisayar sistemleri için kullanılageldiğinden her bir kullanıcının dosya sisteminin nerelerine erişip nerelerine erişemeyeceği ile ilgili köklü bir güvenlik kontrol mekanizması vardır. Kullanıcılar birbirlerinin dosyalarına erişemedikleri gibi sistemin hassa yerlerine de sınırlı şekilde erişebilirler. Oysa başlangıcından beri tek kullanıcılı masaüstü bilgisayar sistemlerini hedefleyen Microsoft Windows sisteminde kullanıcı fiilen tüm sistem üzerinde sonsuz haklara sahiptir. Ancak özellikle bilgisayarların İnternet'e bağlı hale gelmesinden sonra kullanıcıları aldatan virüs programlarının sonsuz kullanıcı yetkileriyle çalışıp tüm sistemi kullanılmaz hale getirmesi gibi sıkıntılar olmuştur. Buna Microsoft'un getirdiği çözümler genellikle tali yamalar olmuş, sistemin esas özellikleri çok zor değiştiğinden virüs sorunu bir türlü tam olarak çözülememiştir. Unix/Linux sistemlerinde baştan veri varolan güvenlik algısı tek kullanıcılı masaüstü sistemlere göre esnetilse de bu tür sorunları yaşamamaktadır.

Linux işletim sistemindeki dosya sistemi alternatifleri farklı koşullarda kullanım için tasarlanmışlardır. Örneğin ReiserFS sistemi çok sayıda ama herbiri küçük dosyaları saklamaya uygun tasarlanmıştır. Ext3 ve Ext4 dosya sistemleri ise genel amaçlı kullanım içindir. Bunlar dışında Linux NTFS ve FAT gibi Microsoft'a özgü dosya sistemlerini rahatça okuyup yazabiliyor.

RAID Sabit disklerde bir tane manyetik kafa bulunur ve bu yüzden okuma yazma hızı bu kafanın ve diskin fiziki hareket hızı ile sınırlıdır. Bunun bir sıkıntı olduğu durumlarda olası bir çözüm RAID teknolojisiyle birden fazla sabit diski eşzamanlı işe koşup tek bir disk gibi çalıştırmaktır. Böylece örneğin sekiz disk kullanıyorsanız sekiz disk kafası aynı anda okuma/yazma yapabilir. Ayrıca disklerdeki bilgilerin özetini biribirine kopyalayarak içlerinden biri bozulduğunda bütün bilgilerin uçması engellenebilir. Birden fazla sabit diski RAID ile birleştirme işlemi işletim sistemi marifetiyle yapılabilir. Ancak en iyi performans anakart üzerinde bu işlemi yapmaya tahsis edilmiş çiplerin bulunduğu durumda elde edilir. RAIDF ile birleştirilen sabit disk alanları istenirse tek bir fiziksel diski böler gibi bölümlemeye tabi tutulabilir.

Dağıtık disk alanları Bazı durumlarda sabir disk alanı ihtiyacı bir bilgisayarın içine birsürü sabit disk tıkıştırmakla karşılanama-

yacak kadar yüksektir. Örneğin Google arama motorunun bilgileri ancak binlerce bilgisayara dağıtılarak depolanabilir. Bazı bilimsel projelerde de bu tür ihtiyaçlar ortaya çıkar. Bu tür dağıtık dosya sistemleri yapmak için hazır bazı yazılımlar vardır (AFS gibi), ancak kimi zmana ihtiyaca özgün sistemler yapılması gerekir. Bazı durumlarda da disk değil ama kullanımı dağıtık olabilir. Örneğin Windows'da dosya paylaşımı adı altında yapılan uzaktan erişim gibi. Benzer bir sistem Linux'ta NFS adı altında mevcuttur. Bu tür sistemler hem kullanıcıların bir dosya sistemini ve içindeki kimi ortak dosyaları paylaşmasına izin verir hem de sistemin tek bir noktadan yedeklenmesine imkan tanır.

4.5 Yazılım: Sistemin Dinamiği

Bilgisayar sisteminin fiziki varlığını oluşturan donanım bileşenleri ancak onların hareketini yönlendiren yazılımlar ile faaliyete geçerler. Sistemin dinamiği bir yandan bu hareketlerin birbiriyle uyumlu olmasına, bir yandan da bileşenlerin olabildiğince birbirinden bağımsız işlemesine bağlıdır.

Bölümün başında bahsettiğimiz ve Şekil 4.2'te betimlediğimiz gibi bilgisayarın bileşenleri kendilerine özgü yazılımlara sahiptir. Örneğin bir sabit disk veya ağ kartı gibi bileşenler üzerinde yeralan ve bileşeni kontrol eden küçük işlemci çipler bu yazılımlarla çalışır. Bu bileşen yazılımlarının işlevi bir yandan sözkonusu bileşenin diğer bileşenlerle standart bir dille konuşmasını sağlamak, öte yandan dışarıdan verilen komutları cihazın kendine özgü sistemine tercüme etmektir. Bu tür bileşen yazılımları üretici firma tarafından yüklenirler. Bizim kontrolümüzde değillerdir ve burada onlar hakkında söyleyebileceklerimiz oldukça kısıtlı.

4.5.1 BIOS

Bileşen yazılımlarından bir tanesi var ki onu anlamak bizim için önemli. BIOS (İng. Basic Input-Output System, basit girdi-çıktı sistemi) yazılımı anakart kontrol çipinin üzerinde çalışan küçük bir yazılımdır ancak önemli birkaç iş görür. Bilgisayar açıldığında ilk çalışan program budur. BIOS programı anakart üreticisi tarafından sisteme konulmuştur ve anakart üzerindeki bileşenlerin düzenine dair bilgiler içerir. Çalıştırıldığı zaman önce bu bileşenlerin sağlıklı olup olmadığına dair kontroller yapar (bu arada açılış ekranında bu bileşenler dair bazı ayarları değiştirmemize imkan verir). Daha sonra da bilgisayarda kullanılacak

işletim sistemi adını verdiğimiz temel yazılımın ana işlemciye yüklenip çalıştırılmasını sağlar, ve o noktada da görevi sona erer. Ancak BIOS'un etkisi bilgisayarın çalışması boyunca devam edecektir. Örneğin uyku durumuna geçmiş bir sistemde uyanışı gerçekleştiren yine odur.

BIOS marifetiyle bir bilgisayar açılırken farklı işletim sistemlerinden birini seçip çalıştırmak mümkündür. BIOS ayarları farklı disk bölümlerine veya CD'ye yerleşmiş işletim sistemleri varsa bunlardan birini seçecek şekilde değiştirilebilir. Bu sayede sistem örneğin bir CD'den, hatta modern bir BIOS sayesinde ağ bağlantısı üzerinden uzak bir bilgisayardan nakledilen bir işletim sistemiyle açılabilir[9].

4.5.2 Sistemin koordinatörü: İşletim sistemi

İşletim sistemi (İng. Operating System) ana işlemci çipi üzerinde çalışan ve dolayısıyla bütün bilgisayarın çalışmasını koordine eden programdır. Aslında bizim, yani kullanıcının çalıştırdığı programlar işletim sisteminin geçici bir süre için ve son derece sıkı kontrol dahilinde ana işlemci çipi kullanma hakkını bu programlara devretmesi marifetiyle çalışırlar. İşletim sisteminin iki temel görevi vardır. Bunlardan ilki sistemin kullanıcı tarafından arzu edilen bir veya birden fazla programı yükleyip çalıştırmasını sağlamaktır. Bunu yapabilmek için işletim sistemi anakartın alarmlı saatinden faydalanarak, aynı birkaç topu havada çeviren bir jonklör gibi, birsürü programı sırayla biraz biraz çalıştırır ve hepsinin birden eşzamanlı çalıştığı izlenimi yaratır. Alarmın her çalışında işletim sistemi merkezi işlemcinin dikkatini bir programdan diğerine vermesini sağlar, alarmı tekrar kura vedevreden çıkarak kullanıcı programının ana işlemciyi kullanmasına izin verir. Bilgisayarın dikkatini hangi programa yönelteceği hassas bir konudur ve işletim sisteminin zamanlama yöntemi programların ihtiyaçlarına ve önceliklerine göre işlemciden pay almalarını temin eden esnek bir yöntemdir. Ayrıca bunu yaparken bu programların bilgisayar belleğinde (RAM) birbirinin işini bozmamaıs için gerekli tedbirleri alır. Bunlar olup biterken donanım bileşenlerinden birinin müdahaleye ihtiyacı olursa bu durum kesme sinyali vasıtasıyla işletim sistemine intikal eder ve sistem kullanıcı programlarını çalıştırmaya kısa bir ara vererek bu durumun gereğini yapar.

İşletim sisteminin ikinci temel görevi de donanımın parçalarının kullanıcı programları tarafından kullanımını hem koor-

dine etmek hem de basitleştirmektir. Bu basitleştirme işlevi çok önemlidir çünkü bir kullanıcı yazılımın çok farklı tipte donanım ve cihazlarla çalışabilmesi beklenir. Örneğin bir müzik çalma programı bilgisayarın tipi, ses kartının marka/modeli değişse de aynı şekilde çalışmalıdır. Bu yüzden işletim sistemi, örneğin gerçekte kullanılan ses kartı bunu kullanıcı programına soyut ve temel bir ses cihazı olarak sunar. Sonuçta kullanıcı programı donanımdaki ses kartı ne olursa olsun ses sistemine aynı şekilde ses gönderir ve alır. Bu alışverişi ses kartının anlayacağı özgün dile çevirip bileşene göndermek işletim sisteminin işidir. Bu yüzden işletim sisteminin böyle bir donanım ile birlikte çalışabilmesi için o donanım bileşenine özgü sürücüye sahip olması gerekir. Webcam, TV kartı gibi cihazların satın aldığımızda bunları elimizdeki programlarla kullanabilmemiz için cihazın paketi içerisinde bizim işletim sistemimize uygun sürücü yazılımı da bulunmalıdır. İşletim sistemi ve sürücülerin birlikteliği donanımla ilgili bütün pis ve gereksiz detayları üstünde çalışan kullanıcı programından gizleyerek bu programların makul bir basitlikte yazımını mümkün kılar.

Linux ve Windows işletim sistemlerinin bu konuda farklı yaklaşımları var. Linux karşılaşabileceği hemen her türlü donanım bileşeninin sürücüsü ile yüklü olarak geliyor, ve sistem bir bileşenle karşılaştığında elinde varsa gerekli sürücüyü kullanıcı müdahalesine gerek olmadan devreye sokuyor. Windows sistemi is bazı bileşenleri bu şekilde desteklemesine rağmen çoğu durumda sisteme yeni bir cihaz eklendiğinde ona uygun sürücünün de işletim sistemine yüklenmesini gerektiriyor. Bu da firmaların birbiriniz yazılımlarını dağıtmasından doğabilecek telif sıkıntılarıyla ilişkili bir durum gibi görünüyor. Bu farklara rağmen günümüzde hemen her türlü donanım bileşeni bu iki sistemle de kullanılabiliyor.

İşletim sisteminin donanımla ilgili bir görevi de koordinasyondur demiştik. Örneğin aynı anda hem bir müzik çalar programı çalıştırıyor hem de sesli bir video oynatıyorsanız bu iki programın da eşzamanlı olarak ses sistemini kullanması gerekecektir. Benzer bir durum iki kullanıcı programı aynı anda sabit diskten bazı dosyaları kullanmaya çalıştığında da ortaya çıkar. İşletim sistemi bu tür durumlarda bileşenlerin kullanımını uygun sıraya sokarak veya eşzamanlı kullanım mümkün olmadığında programlara bu durumu uygun şekilde bir hata mesajıyla bildirerek koordinasyon görevini yerine getirir.

Bütün bunlar sayesinde işletim sistemi kullanıcı programlarını donanımla ilgili detaylarla uğraşmaktan kurtarır ve eşzamanlı çalışmanın kazasız belasız gerçekleşmesini sağlar. Ancak işletim sistemi o kadar temel bir programdır ki diğer programlar onunla konuşmayı, hizmetlerinden yararlanmanın yolunu bilmeden çalışamazlar. Bu yüzden bizim kullanacağımız programların kullandığımız işletim sistemi ile uyumlu olması gerekir. İşte bu durum ciddi bir sıkıntı konusudur. Kimi zaman ihtiyacımız olan programın kullandığımız işletim sistemi ile uyumlu bir sürümünü bulamayabiliriz. Linux sistemini geliştiren programcılar bu problemi hafifletmek için Wine (İng. Windows Emulator kısaltması) adlı bir program geliştirdiler. Bu program sistemin Windows taklidi yaparak, o dili konuşarak Windows uyumlu programların Linux üzerinde çalışmasına imkan veriyor. Mac firmasının bilgisayarlarında çalışan işletim sistemi de (Max OS-X Darwin) Unix/Linux ailesiyle ortak kökenlere sahip olduğundan bu iki sistem büyük ölçüde aynı yazılımları çalıştırabiliyor.

Şirketlerüstü bir kurum olan IEEE yıllardır sürdürdüğü bir çalışmayla işletim sistemlerinin kullanıcı programlarıyla etkileşiminin ortak bir standarda oturtulmasına ve bu sıkıntıların giderilmesine uğraşıyor. POSIX adı verilen bu standardın gelişmesi ve yaygınlaşması sınırlı kalmış ve bir miktar rahatlama sağlanmasına rağmen tam bir yumluluk çözümüne dönüşmemiştir.

4.5.3 Sistem ve kullanıcı yazılımları

Bir bilgisayar sisteminde BIOS ve işletim sistemi dışında çok sayıda yazılım çalıştırılmaya hazır şekilde bulunur. Bu yazılımlar çalıştırılabilir son kullanıcı programlarının yanısıra birçok programın paylaştığı becerileri içeren kitaplıklar, görsel unsurları içeren grafik dosyaları (örneğin program menülerindeki grafik semboller), programların kullanım kılavuzları, farklı dillerde kullanılmalarını sağlayan çeviri dosyaları gibi birçok parçadan oluşurlar. Benim kullandığım Ubuntu Linux işletim sisteminde onbinden fazla kuruluma hazır yazılım mevcut, ve dört binin üzerinde çalıştırılabilir program halen kurulu durumda.

Bu yazılımların bir kısmı altyapıda çalışırlar. Biz onları doğrudan kullanmasak ta sistem tarafından devreye sokulurlar. Bunlar bilgisayarı kullanmamızı sağlayan oturum yöneticisi, ağ bağlantı sihirbazı, vb., yazılımlardır. Başka bir kısım ise üstyapıda çalışan her programın vazgeçilmez ihtiyacıdır. Bunlarda örneğin dosya sis-

temini kullanmamızı sağlayan dosya yöneticisi gibi programlardır. Bu iki gruptan programlar o kadar sistemin ayrılmaz parçaları haline gelmişlerdir ki teknik olarak işletim sisteminden bağımsız olmalarına rağmen onları da işletim sisteminin bir parçası kabul ediyoruz. Karışıklığı önlemek için işletim sisteminin kendisine çekirdek (İng. kernel) adı veriliyor.

Bunların dışında kalan yazılımlara ise kullanıcı programları adını veriyoruz. Bunlar her kullanıcının ihtiyacına göre ediniop kullandığı programlardır. Bu programların bir kısmı çoğumuzun aşina olduğu ofis programları (kelime işlemci, tablo düzenleyici, vb.) veya web tarayıcısı gibi programlardır. Bunlar dışında bireysel veya kurumsal ihtiyaçlara yönelik birçok program bulunuyor. Günümüzde oldukça büyümüş olan yazılım sektöründeki şirketlerin ürettiği ve bir ticari ürün olarak satılan programların yanısıra vakıf, üniversite veya bireysel geliştiriciler tarafından üretilen ve ücretsiz dağıtılan kamu lisanslı programlar da mevcuttur. Üstelik oldukça da kalitelidirler. Kaliteli bir şeyin ücretsiz olamsı size oldukça şaşırtıcı gelebilir. Yazılım dünyasına özgü bu durumun ekonomik kökenlerinden son bölümde bahsedeceğiz.

Aynı bilgisayarı anlamanın zor oluşugibi bir bilgisayar programının kalitesini anlamak da oldukça zordur. Özellikle gözönünde bulundurulması gereken bir unsur yazılımın başka yazılımlarla uyumlu olup olmadığıdır. Ticarai yazılımların bir bölümü bu konuda sıkıntılıdır. Üretici firmalar müşteriyi kendi ürün yelpazesinin içine hapsetme hevesinde olabilirler. Bir başka konu ise işiletim sistemleri ile uyumlu olmasıdır. Çoğumuz birkaç yılda bir bilgisayarımız değiştirip yenisini alıyoruz. Bunu yaparken bilgisayarımızın temel mimarisi değişebilir (örneğin 32 bit yerine 64 bit), hatta farklı bir işletim sistemi kullanmayı isteyebiliriz. Ancak bunu yaparken alıştığımız yazılım araçlarını da kullanmaya devam etmek isteriz. Bu yüzden uzun vadede sıkıntıları önlemek için seçtiğiniz yazılımların farklı bilgisayar mimarisi ve işletim sistemleriyle çalışabilen sürümlerinin mevcut olmasına dikkat etmenizde yarar var.

4.6 Bilgisayar sistemlerindeki değişimin yönü

Bu bölümün başında 1980lerde PC standardının ortaya çıkışında bahsetmiştik. Genel olarak bilgisayarın işleyişiyle ilgili birçok şeyin standartlaşması eğilimi devam etmektedir. Bu hem donanım hem de yazılım ve veri alışverişi konularıyla ilgilidir. Örneğin son

yıllarda yaygınlaşan USB standardı taşınabilir cihazların bilgisayarla ve birbirleriyle bağlantısını son derece kolaylaştırmıştır. Web sayfalarıyla ilgili HTML standardı ise farklı web tarayıcısı kullanan kullanıcıların aynı sonucu almalarını sağlar.

Genel olarakstandartlaşma pek çok üretici firmanın varolduğu ve kullanıcı ihtiyaçlarının çok çeşitli olduğu bu alanda işlerin sorunsuz gelişmesini sağlar ve innovasyonun önünü açar, ki bu konuya da son bölümde değineceğiz. Son kullanıcılar açısından da satın aldığımız yazılımların standartlara uygunluğuna dikkat etmek önemlidir. Standartlara uygun yazılımlar hem diğer yazılımlarla uyumlu çalışma, veri alıp verme konusunda daha az problem çıkartırlar, hem de yerine muadil bir yazılım kullanma özgürlüğümüzü kısıtlamazlar.

Bölüm 5

Bilgisayarlar arası iletişim ve İnternet

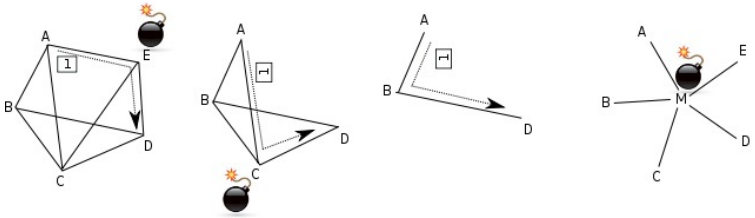
İnternet sözcüğü inter-network, yani ağlar-arası teriminin kısaltmasıdır. İlli ki Türkçe'ye çevirmemiz gerekse 'ağlar-ağı' demek çok uygun düşerdi. İsmi İnternet'in doğasına çok uygun düşüyor, çünkü gerçekten de birbirinden bağımsız işleyen bilgisayar ağlarının birbirine bağlanmasından ortaya çıkmaktadır. En belirleyici özelliği de bir merkezinin ve hiyerarşik bir kontrol sisteminin olmayışıdır. Bu özellikler sayesinde (1) hem sisteme dahil bazı ağlarda arıza olsa bile İnternet'in kalanı işlemeye devam eder, (2) hem de esas olarak bu küresel sistemde sınırlar çizmek ve engeller koymak pek mümkün değildir. Ancak merkezsiz ve kontrolsüz olmasına rağmen İnternet üzerinden iletişim gayet sistematik ve hızlı olarak yapılabiliyor. İnternet'in işleyişi büyük ölçüde eski usul posta sistemine benzer ve temel prensiplerinin anlaşılması sanıldığından daha kolaydır, ve bu bölümde bu işleyişi anlatacağız. Ancak bu işleyiş prensiplerine girmeden önce kısaca İnternet'in tarihçesinden bahsedeceğiz; çünkü günümüzdeki teknolojinin temel karakterinin oluşumu bu tarihçede gizli. Ayrıca oldukça ilginç bir hikayesi var.

5.1 İnternet'in tarihi

Günümüzdeki İnternet'in çekirdeği olan teknolojik gelişmelerin ortaya çıkışı 1960'larda gerçekleşir. Sovyetlerin 1957'de ilk uzay aracı Sputnik'i göndermesinden sonra açığı kapatma ve teknolojik rekabet gücünü arttırma derdine düşen ABD ordusu 1958'de ARPA (Advanced Research Projects Agency, İleri Araştırma Projeleri Ajansı) isimli bir ofis kurar. Bu ofisin amacı sivil dünyada, özellikle üniversitelerde ortaya çıkan yenilikçi araştırmaların desteklenmesi ve bir şekilde ordu teknolojilerine inovasyon katkısı sağlanmasıydı. Ancak ordu bütçesinden desteklenmesi bir yana bu ofis çalışmalarını akademi dünyasında, büyük ölçüde bağımsız olarak ve sivil bir tarzda sürdürüyordu. İnternet teknolojisinin doğuşu bir yandan bu ofisin kendine özgü konumlanması ve mali desteği, bir yandan da bu çalışmalara katılan bilim insanlarının özgürlükçü vizyonu ve becerisi ile ortaya çıktı (Castells, 2001).

Bu bilim insanlarından biri 1962'de ofiste çalışmakta olan J.C.R. Licklider idi. Licklider 'inter-galaktik bilgisayar ağı' adını verdiği tasarımı hayata geçirmek için ofis yönetimini ikna etti. İlginçtir ki bu ağın tasarlanmasındaki amaç ARPA çalışmalarına katılan birkaç üniversitedeki bilgisayarların birbirine bağlanması ve böylece bilim insanlarının daha rahat birlikte çalışabilmesi idi. Yani tamamen gayri-askeri bir amacı vardı. Öte yandan ARPA'daki ekip bu projeyi hayata geçirmek için ordu bünyesindeki başka bazı araştırmalardan da haberdar olabiliyor ve yararlanabiliyordu.

Ekibin yararlandığı bu araştırmalardan biri Amerikan ordusunun yaptırdığı, o zamanların en büyük paranoyası olan nükleer saldırı durumunda bile çalışabilecek bir askeri haberleşme ağı tasarımı idi. Paul Baran'ın bu proje için yaptığı tasarım dağıtık bir iletişim essasına dayanıyordu. Baran'ın tasarımının iki temel özelliği vardı: (1) bilgisayarlar arasındaki veri alışverişi küçük paketlere bölünerek yapılacak, ve (2) her bilgisayar en az üç başka bilgisayara bağlı olacaktı. Bu tasarımın amacı şuydu: haberleşme ağına bağlı iki cihaz veri alışverişi yaparken, sistemdeki bazı cihazlar nükleer saldırıda havaya uçsa bile sağlam kalan cihazlarla olan bağlantıları kullanılabilecek, ve kalan veri paketleri bu alternatif bağlantılardan geçerek hedefine ulaşabilecekti. Bu fikri şekil 5.1'de göstermeye çalıştım. Bu senaryoda A ve D noktaları veri alışverişi yapıyor, ama aralarında doğrudan bağlantı yok. A'dan D'ye gönderilen veri paketleri önce -belki en kısa ve hızlı yol olduğu



Şekil 5.1: İnternet tasarımının başlangıcı olan, merkezsiz paket veri ağındaki haberleşmenin bazı cihazların tahrip olmasına rağmen işleyişi. En sağda gösterilen merkezli ağ yapısı merkez tahrip olunca çalışamaz.

için- E aracılığıyla teslim ediliyor. E noktası bombalanınca veri trafiği sağlam kalan C aracılığıyla yapılıyor. O da bonbalanınca veri akışı B aracılığıyla hala sürdürülebiliyor. Eh, bütün bunlar olurken bu bombardımana karşı savunma yapılacaksa bunu hala sözkonusu ağ aracılığıyla yapmak mümkün olurdu. Buna karşılık şekilde gözterilen merkezi ağı düşünün. Merkez (M noktası) bombalanırsa hiçbir veri haberleşmesi yapılamayacaktır. Baran'ın tasarımı verinin küçük paketlere bölünmesini öngördüğünden ağdaki bir bağlantının havaya uçması en fazla bir veri paketini etkileyecekti. Büyük ve sürekli bir veri akışının başarısız olup baştan tekrarlanması durumu olmayacaktı. Her ne kadar verinin bölüp paketlenmesi ve sonra birleştirilmesi başlıbaşına bir iş olsa da bu esnekliğe değerdi.

Bu tasarımdan haberdar olan ve beğeniyle karşılayan ARPA ekibi, ortada böyle bir askeri savunma durumu olmamasına rağmen kurmak sitedikleri üniversitelerarası bilgisayar ağı için bu dağıtık ve paket veri tabanlı iletişim tasarımını esas aldılar. Bu vizyonun hayata geçmesi uzun zaman alacaktır. 1960'lar boyunca üniversitelerdeki bilgisayarları birbirine bağlamak için birçok deneme yapılır. Ancak bunların hepsi telefon bağlantısına benzer, ne bugünkü İnternet'le ne de baran'ın paket verilere dayalı temel tasarımıyla tam örtüşmeyen denemelerdir. Yine de 1968'de bir makalede Licklider gelecekte insanların yaygın olacak bilgisayarlar aracılığıyla uzaktan iletişim kuracağına inandığını yazacaktır (Licklider and Taylor, 1968). Bundan da öte bu tür iletişim imkanlarına herkesin eşit erişim hakkının sağlanması

gerektiğini vurgular. Licklider bu yaklaşımında yanlız değildir. Kendisi proje tamamlanmadan ARPA'dan ayrılmasında rağmen İnternet'in gelişiminde rol alan pek çok bilim insanı bu 'açık erişim' idealini taşıyordu. İnternet'in gelişimi sadece teknik bir ayrıntı değil ortaya çıktığı dönemin bilim camiasındaki özgürlük ve demokrasi ruhu ile içiçe geçmiş bir kollektif tasarım öyküsüdür.

Dağıtık paket veri iletişimi esasına dayalı tasarımlar ancak 1968-69'da gerçekleşecektir. 1969'da Kaliforniya'da dört üniversite bilgisayarı birbirine bağlanıp ilk denemesi yapılır. Bu ilk denemede "login" kelimesinin ağ üzerinden aktarımı sırasında ikinci harften sonra sistem çöker[10]. Yine de bu tarih dağıtık paket veri iletimi konusunda bir dönüm noktası olacak ve takip eden başarılı denemeler ardarda gelecektir. ARPANET adı verilen bu ağa bağlı bilgisayar sayısı 1971'de onbeşe çıkar(Castells, 2001) ve ABD'nin öbür kıyısında, MIT'deki araştırmacılar da İnternet'i geliştirme sürecine dahil olur. Birçok kurumun paralel çalışmalarının etkileştiği 1970'lerin sonunda, 1978'de İnternet'in bugün de kullanılan temel standardı olan TCP/IP standardı UNIX işletim sisteminin bir parçası olarak yayınlanır ve sistemi kullanan birçok kurumun uyumlu bir şekilde ARPANET ağına bağlanıp birbirleriyle veri transferi yapmasına olanak tanır.

Bu teknolojiden bir ölçüde bağımsız gelişmekte olan ve İnternet'in dağıtık çoklu bağlantı tasarımından farklı olarak, telefon şebekesi üzerinden birebir bağlantı ve veri transferi için tasarlanan BBS, MODEM gibi teknolojiler 1970'lerde popülerlik kazanır. Bunlar 80'lerde İnternet teknolojisi ile entegre olmaya başlayacaktır. 80'lerin başından itibaren bu teknolojilerin sivil kullanımının yaygınlaştığını gören ABD savunma bakanlığı bir yandan özel sektör kullanımının yaygınlaştırılması için gerekli önlemleri ve düzenlemeleri ele alır, bir yandan da ordu kullanımı için ayrı bir ağ oluşturur.

İnternet'in gelişim öyküsü aynı zamanda ağa bağlanan herkesin aynı kurallarla konuşmasına imkan sağlayan standartların geliştirilmesi öyküsüdür. Bunlar hemen her zaman 'protokol' olarak adlandırılıyorlar çünkü iki taraf arasındaki etkileşimin aynı resmi devlet protokolündeki gib sıkı davranış kurallarına bağlanması, sürpriz davranışlar olmaması ve tarafların bu yüzden ne yapacağını şaşırmaması gerekiyor. 1983'te tasarlanan e-posta değiş-tokuş protokolü gibi standartlar hala yaygın olarak kullanılmaktadır. Buna karşılık İnternet veri paketi trafiğini düzenleyen TCP/IP

standardı yerine uluslararası standartlar örgütünün (ISO: International Standards Organization) tasarladığı alternatif standardın kabul görmemesi gibi durumlar da yaşanmıştır. Sonuçta küresel ölçekte birçok paydaşı ilgilendiren bu standartlar resmi kurumlardan ziyade büyük ölçüde akademik ve profesyonel camia tarafından, konsensus ile şekillendirilmeye uygundur. Bu durumu takdir eden dönemin ABD resmi otoriteleri 1989'da hükümetten bağımsız çalışan, meslek erbabının katılımıyla işleyen IETF (Internet Engineering Task Force) gibi sivil örgütleri öne çıkartarak İnternet standartlarının sağlıklı bir şekilde evrilmesinin önünü açmışlardır. 1990'larda İnternet'in ABD dışında da yaygınlaşmaya başlamasıyla birlikte hem IETF hem de konuyla ilgili diğer sivil örgütler de küresel katılıma açılacaktır.

1950'lerin sonunda ARPA'nın kuruluşundan itibaren akademi ve sivil meslek camiası ile işbirliğini teşvik eden ve ortaya çıkan teknolojiyi ordu kullanımına sınırlamak yerine sivil kullanımı kolaylaştıracak önlemler almayı tercih eden ABD bu stratejinin karşılığını almış ve nihai olarak Sovyt nükleer tehdidine karşı bambaşka bir şekilde, İnternet'in başarısıyla büyük bir üstünlük sağlayarak yarışta öne geçmiştir.

5.2 İnternet'in İşleyişi

Ağ teknolojilerinden bahsedilirken en çok duyacağınız terim muhtemelen 'protokol' terimidir. İletişimin her biçiminde olduğu gibi iki bilgisayarın iletişiminde de en başta bir ortak dil gereklidir. Ağ protokolleri işte bu ortak dili sağlar. Telin iki ucundaki bilgisayarlar ne kadar farklı model ve işleyişte olsalar da aynı protokolü (ya da protokolleri) konuştukları sürece iletişim mümkündür. Protokoller karşılıklı olarak nasıl davranılacağına, hangi tür veriye ne tür verilerle cevap verilebileceğine dair bir repertuar sağlarlar, ve bunun dışına çıkılmadığı sürece telin ucundakiler ne konuşacağını şaşırmaz.

Bu protokoller uluslararası araştırma komisyonları tarafından biçimlendirilir ve son şekli verilir. Bir protokol bir kez kullanılmaya başlandı mı hem ağı oluşturan elektronik cihazlar ve bilgisayar parçaları, hem de bunları kullanan bilgisayar programları buna uygun yapılır. Bu yüzden de protokolde değişiklik yapmak çok zordur! Ağ protokollerinin geliştirilmesi ve standartlaşmasında etkili olan kurumlar arasında IEEE (Elektrik ve Elektronik Mühendisleri Enstitüsü), ISO (Uluslararası Standartlar Örgütü),

IETF (Internet Engineering Taskforce), ISC(Internet Standards Consortium) sayılabilir. Genellikle bir üniversite laboratuarı veya teknoloji şirketinde denemelerle başlayan protokol geliştirme süreci bu meslek örgütlerinde karşılıklı görüş alışverşleri ile şekillenir ve belirli bir olgunluğa ulaştığında teknolojik standart halinde duyurulur.

İnternet iletişiminin temel protokolü olan IP de (İng. Internet Protocol) 1970'lerde ARPA desteğiyle geliştirilmeye başlanmıştır. Yıllar içerisinde görülen sorunları, özellikle İnternet'in hızla büyümesi ve güvenlikler ilgili ortaya çıkan sorunları giderecek şekilde düzeltilmiştir. 1981'de IPv4 adıyla kısaltılan dördüncü sürümü bugün de kullanılan olgun standart olarak kullanıma girmiştir [11]. Hem IP'nin hem de onun üzerine yerleşen yüzlerce İnternet standardının resmi tanımı IETF'in web sitesinde bulunabilir (`http://www.ietf.org`).

IP protokolünün işleyişi posta sistemimize son derece benzer, en önemli fark IP'deki veri paketlerinin kablola (veya radyo dalgaları) üzerinden çok hızlı hareket etmesidir. Posta sistemindeki bir mektubu ele alalım. Mektubun zarf üzerinde gönderici adı ve adresi, alıcı adı ve adresi, bir de postaya verildiği tarih bulunur. Gönderdiğimiz şey zarfı yırtıp çıkan münasebetsiz bir nesne olmadığı sürece, zarf üzerindeki bilgiler içindekinin alıcısına teslim edilmesi için yeterlidir. Zarf içine sığacak sayfa miktarı sınırlıdır. Bu zarf postaneye teslim edildikten sonra birçok elden geçecek ve her seferinde alıcı adresine bakılarak o yöndeki bir posta dağıtım noktasına doğru ilerleyecektir. Verilen adreste alıcı bulunamaz ya da belirli bir süre içerisinde zarf teslim edilemezse geri döner (ya da PTT'de kaybolur).

İnternet protokolü ile iletişim de paket tabanlı olup tamamen posta zarfı örneğine benzer. Bir IP paketinin zarfına göz atalım:

```
Gönderen: TCP-2056
Gönderici adresi: 194.27.145.72
(Tarih yerine) geri sayım sayacı: 10

                    Alıcı: TCP-80
                    Alıcı adresi: 212.175.10.5
```

Bir İnternet paketinin bu zarfında da gönderici ve alıcı adresleri var. Ancak bunlar dört parçalı sayılardan oluşuyor. İnternet'te veri alışverişinin sağlıklı işlemesi için kimse bir başkasıyla aynı

adrese sahip olmamalıdır ki karışıklık çıkmasın. Adres tekilliğinin koordinasyonu ABD merkezli ICANN (Internet Corporation for Assigned Names and Numbers, İnternet Tahsisli Sayılar ve İsimler Kurumu) adlı kurum tarafından yapılır. Bu işleyişte örneğin 212.x.x.x adres bloğu -arada ülke yetkili kurumları olmak üzere- TTNET adlı şirkete tahsis edilir. Bu yetkili şirket te her müşterisinin bu adres bloğu içinden farklı bir adres kullanmasını sağlamakla yükümlüdür; ve isterse bu bloğun içinden küçük blokları da kurumsal müşterilerine toptan tahsis edebilir, vs. Bu kademeli koordinasyon marifetiyle örneğimizdeki IP adresi İstanbul Bilgi Üniversitesi'ndeki bir bilgisayara tahsis edilmiş.

Geleneksel posta sistemindeki adresler (Örneğin "Moda caddesi, No:1, Kadıköy, İstanbul, Türkiye") coğrafi bir yere karşılık geldiği için anlaşılması problem değildir. Peki ama 212.171.10.5 adresinin hangi kablonun ucu olduğu nasıl bulunuyor? Burada da adres dağıtımını yapan kurumlar sorumluluk alırlar. ICANN'den başlayıp TTNET'e kadar inen adres dağıtım yetki silsilesindeki kurumlar verdikleri adres bloklarında bir adrese gönderilen veri paketlerini yerine ulaştırmak için sorumludurlar. Bu kurumlar posta sistemindeki postaneler gibi hizmet verir. Kendi hizmet alanlarındaki alıcılara gelen zarfları tasnif edip topluca ilgili postanelere yönlendirirler. TTNET gibi İnternet hizmet sağlayıcısı kurumların bünyesinde yönlendirici (İng. router) adı verilen çok sayıda böyle bölge dağıtım noktasi işi gören cihaz bulunur. Bu cihazlar sürekli olarak birbirleriyle bilgi alışverişi yaparlar ve bu sayede hangi komşu dağıtım noktasını hangi alıcı adresleri için kullanacaklarını keşfederler. Ayrıca dağıtım kanallarını kullanırken trafik durumunu da gözönüne alırlar ve alternatif dağıtım yönlerini kullanabilirler.

Bizlerin böyle sayılardan oluşan İnternet adreslerini hatırlamamız çok zor. Bunun için eskiden beri bir 'alan adı sistemi' (İng. DNS, Domain Name System) kullanılır. Alan adı sistemi www.yahoo.com gibi bizim için hatırlaması kolay adreslerin 87.248.122.122 gibi gerçek karşılıklarını veren bir sistemdir. Bilgisayarlar sayılarla daha rahat ve hızlı çalışırken biz sözcükleri daha rahat hatırladığımız için böyle bir adres tercüme sistemi gerekmiştir. Günümüzde kullandığımız web tarayıcısı gibi programlar bu sistemi bize hissettirmeden kullanıyorlar. Benim gençliğimde İnternet'e bağlı bilgisayar sayısı o kadar azdı ki alan adı bilgisi arada sırada ABD'deki bir merkezden yenisi gönderilen

küçük bir dosyaya sığıyordu. Ancak günümüzde her ülkede birçok yetkili kurum bu alan adlarını -ücret karşılığı- tahsis ediyor, hem de çok sayıda. Aynı sayısal IP adresinde olduğu gibi bu adların kullanımında da bir karışıklık olmaması için kademeli bir yetkilendirme sözkonusu. Günümüzün alan adı sistemi bu yetki hiyerarşisi içinde dağıtık bir şekilde alan adı bilgilerini bulup getiriyor. Örneğin '.tr' diye biten alan adlarının karşılığı ODTÜ'deki Türkiye alan adlarından sorumlu ofisin bilgisayarlarına soruluyor. Onlar da, örneğin 'edu.tr' alt grubu ile ilgili sorguları ULAKBIM'e yönlendiriyor, onun dışında kalan hemen her sorguyu kendi kayıtlarına göre cevaplıyorlar.

Şimdiye kadar sadece IP veri zarfının üzerindeki gönderici ve alıcı adresinden sözettik. Bu zarfta ayrıca bir geri sayım sayacı var. Posta zarflarındaki tarih bilgisi yerine geçen bu sayaç ta İnternet'in saplıklı işleyişi için önemlidir. Zarfın üzerindeki alıcı adresi şu ya da bu sebepten dolayı hatalı (sahibi olmayan) bir adres ise, veya adreste olması gereken bilgisayar bpzuk/kapalı ise ne olacak? İnternet'in merkezi ve müdüriyeti olmayan dünyasında bu duruma hükmedecek bir yetkili merci yoktur. Böyle bir durum ortaya çıkarsa yönlendirici cihazlar bu veri paketini yerine teslim etmek için umutsuzca birbirlerine paslayıp dururlar. Bu tür durumları önlemek için kullanılan geri sayım sayacı her paslamada bir eksiltilir. Eğer sayaç sıfıra kadar düşerse veri paketinden umut kesilir ve yokedilir. Böyle bir önlem alınmasaydı umutsuzca ortalıkta gezinen paketler yüzünden İnternet veri paketi dağıtım sistemi birkaç saate kalmadan işlemez hale geliverirdi. Ancak bu önlem bazen yanlışlara da yolaçar. Yönlendiriciler paket paslamalarında farklı alternatifler arasında trafik durumunu da gözönüne alıyorlar. Yönlendiricilerin verdiği anlık kararlar neticesinde bir veri paketi meşru ve ulaşılabilir bir alıcı adres olmasında rağmen sırf yolu uzadığı için geri sayım sayacı sıfıra düştüğünden dolayı adrese teslim edilemeyebiliyor. İnternet Protokolünün basit ve hızlı işlemesi için paketler için yerine ulaşma garantisi verilmemiştir, ancak sistemin ayarları (örneğin geri sayım sayacının ilk değeri) paket dağıtımının çok büyük oranda başarılı olmasını sağlıyor.

Bu tür paket kaybolma durumlarıyla nasıl başa çıkıldığını biraz sonra anlatacağım. Ancak bunu yapmadan İnternet'te veri paketlerinin dağıtımını sağlayan bu adres sisteminin bazı kısıtlamalarından ve özelliklerinden bahsedelim. İnternet adresle-

rini 212.175.10.5 şeklinde dört parçalı yazıyoruz, çünkü esasen dört byte'tan oluşuyorlar. Her byte ta sekiz bit olduğuna göre $2^8.2^8.2^8.2^8 = 2^32$, yani yaklaşık dört milyar farklı adres mümkündür. Ancak bu adreslerin bir kısmı bazı özgün ihtiyaçlar için bir kenara ayrılmıştır. Örneğin zaman zaman evdeki ADSl modem ayarlarını yaparken rastlamış olabileceğiniz 192.168.x.x gibi adresler yerel ağda kullanmak içindir ve küresel İnternet sisteminde kullanılamaz. Keza 127.x.x.x adres grubu da bilgisayarın iç haberleşmesinde IP kullanıldığı durumlar için ayrılmıştır. Bir grup adres ise multicast denilen veri dağıtımı türü içindir. Tek bir alıcı değil de bir kanal numarası ifade eden multicast adresleri İnternet üzerinden TV veya radyo yayını gibi amaçlar içindir ve kanal adresini dinlemek istediğini belirten uçlara bu alıcı adresine sahip veri paketlerinin birer kopyasının ulaştırılmasını gerektiren karmaşık (ama trafik açısından verimliliği yüksek) bir sistemi hedefler. Bu sistem ancak yönlendirici cihazların gelişmesiyle yeni yeni kullanılabiliyor.

Bütün bu kendine özgü ayarlamalar nedeniyle IP adresleme sisteminde en fazla iki milyar farklı adres kullanılabiliyor (Stallings, 2006). 1980'lerin başında yapılan tasarımda son derece yeterli görülen bu adres miktarı günümüz koşullarında çok sınırlı kalıyor çünkü her birimiz ofis bilgisayarı, dizüstü, cep telefonu gibi birsürü cihazın herbiri için bir IP adresine ihtiyaç duyuyoruz. Bu durumu karşılamak için IP standardının çok daha fazla sayıda adrese izin veren yeni bir sürümü (sürüm 6, ya da IPv6) geliştirilmiştir. Günümüzde birçok bilgisayar sistemi bu yeni standardı destekliyor. Ancak onlarca yıllık kullanımı olan IPv4'ten öyle bir hamlede vazgeçilemiyor ve bunun gerçekleşmesi yavaş ve yıllara yayılmış bir sürece olacak.

5.2.1 Tek bir veri ağı, binbir çeşit veri ve servis

İnternet'teki veri dağıtımının -hızı hariç- posta sistemine benzediğini söylemiştik. Fakat önemli bir fark var ki o da İnternet adreslerinin her biri bir ev ya da apartman dairesine değil de daha çok kendi başına koca bit gökdelene karşılık geliyor. Geçen bölümde gördüğümüz veri zarfının üzerindeki alıcı adresini hatırlayalım:

```
Alıcı: TCP-80
Alıcı adresi: 212.175.10.5
```

Burada 212.175.10.5 adresinde buluna gökdelen İnternet'e bağlı bir bilgisayardır. Bu gökdelen 2^16 (yani 65536) katlı, ve her katta iki daire var (TCP ve UDP isimli). Örnekteki zarfın içindeki verile 80. katta TCP dairesine gelmiş. Alıcı bilgisayar bu veri paketini aldıktan sonra gökdelenin kapıcısı (ki bu rolü işletim sistemi üstlenmiştir) paketi ilgili daireye teslim eder. Tabii bu dairelerin hepsi dolu değil, çoğu boştur.

Bu dairelerin herbirinde ayrı bir dil konuşulmaktadır. Ancak kural şu ki her gökdelenin aynı kapı numaralı dairesinin kapısını çaldığımızda kapıyı açan (bir bilgisayar programı) aynı dili konuşur. Bizim örneğimizdeki kapı numarası, yani TCP-80, HTTP denilen ve web sitesi içeriğinin alışverişi için kullanılan yaygın bir dildir. Biz web tarayıcımızda `http://www.google.com` veya `http://www.facebook.com` gibi bir adres yazdığımızda tarayıcımız önce alan adı sistemi ile bir IP adresi buluyor, sonra o adresteki TCP-80 numaralı kapıyı çalıyor ve karşısına çıkan bilgisayar programıyla (bu durumda bir web sunucusu) HTTP dilinde konuşmaya başlıyor. Her adresin TCP-80 nolu dairesinde (eğer birisi yaşıyors) aynı dil konuşuluyor.

Daire numaraları için de sayı yerine hatırlaması kolay olsun diye http, https, ftp gibi kısaltmalar kullanılıyor. Web tarayıcısı gibi programlar bu kısaltmaların karşılığının hangi daire olduğunu bilir. Bazen alışılmışın dışında servis veren bir daire numarası varsa bunu `http://ornek.com:8080` gibi bit ifadeyle webtarayıcısını yönlendirerek kullanmak mümkündür. Verdiğimiz zarf örneğinde kapıyı çalan göndericinin (istemci, İng. client) hangi kapıdan çıktığının önemi olmadığını belirtelim. Burada standart olan sunucu daire numaralarıdır.

Bu bahsettiğimiz senaryoda web tarayıcısı ile uzaktaki web sunucusunun konuşması ile ilgili temel bir sıkıntı var. Sözkonusu konuşmada sarfedilen cümleler oldukça uzun olabilir, özellikle web sunucusundan gelenler. Bir web sayfası, içindeki fotoğraflar, veri dağıtımından sorumlu IP sisteminin kabul edeceği büyüklükte zarflara sığmazlar. Bu yüzden veriyi fasiküllere bölüp ayrı zarflarda göndermek gerekir. Bu da bir dizi sıkıntı doğurur. İnternet'in karmaşık ve zaman zaman sıkışan trafiği içinde bazı zarflar uzunyollra sapıp kaybolur (geri sayım sayacını hatırlayın), diğerleri ise karmakarışık bir sırada karşı kapının önüne yığılır. Gökdelenin her kapısında yaşanabilecek bu sorunlara bulunan cevap aktarım kontrol protokolüdür (İng. Transmission Control Protokol,

yani TCP). Bu protokol IP ile üstüste beraber İnternet trafik düzeninin temelini oluşturur. TCP protokolü böyle bölünmek zorunda kalınan veriler için fasikülleri numaralandırmak, varış noktasında sıraya dizmek, eksikleri karşı taraftan tekrar isteyip tamamlamak işlevlerini üstlenir ve bütün bunlar gökdelenin çok gelişmiş kapıcısı tarafından halledilir. Böylece uzaktan konuşan iki muhatap (mesela web tarayıcısı ile web sunucusu) bir diğerinin sözlerini ağzından çıktığı sırada ve eksiksiz duyar (zaman zaman bunun gerçekleşmesi için kapıcı tarafından zorunlu olarak biraz bekletilse bile).

Gökdelenin UDP kanadından bahsetmedik. Bu taraftaki kapılarda konuşulan diller tek soru tek cevap şeklinde ve kısa cümlelerden oluşur. Bu yüzden veri bölme, sıralama, eksik tamamlama gibi sıkıntılar yoktur. Bu taraftaki kapılar alan adı sorgulaması (DNS), saat senkronizasyonu (NTP) gibi tek ve kısa soru cevap ile yapılabilen işler için kullanılırlar.

Bütün bu düzenlemede IP gökdelen kapısına kadar teslim işini üstlenir. TCP ve UDP ise gökdelen içi adresleme, ve anlattığımız gibi paketlerin sıraya dizilmesi işini üstlenir. Çok nadir durumlarda IP paketi bir daireye değil doğrudan kapıcıya da gönderilir. Örneğin ICMP (İng. Internet Control Messaging Protocol) İnternet noktaları, özellikle yönlendiriciler arasında bilgi alışverişi için kullanılır ve içeriğinin muhatabı kapıcılardır. Bu protokolle taşınan içerik kapıcıların veri paketlerini daha hızlı, daha müsait kanallardan göndermelerine yarar.

5.3 Yaygın İnternet Servisleri

Şimdiye kadar sadece veri paketlerinin zarfından bahsettik. Şimdi biraz da içeriğine bakacağız.

80'li yıllar boyunca İnternet trafiğinin çok büyük kısmı elektronik posta transferi için kullanılıyordu (SMTP,İng. Simple Mail Transfer Protocol, Basit Posta Transferi Protokolü, TCP-25 nolu kapıda konuşulur). Bu da ARPA ekibinin orijinal hedeflerine çok uygun düşmektedir. Yine aynı dönemde dosya aktarım protokolü de (FTP, İng. File Transfer Protocol) özellikle bilimsel camiada oluşmaya başlayan elektronik belge kitaplıklarına erişimi sağlıyordu. Bu ve benzer protokollerin herbiri sözkonusu işe uygun bir diyalog şablonu ve komut dağarcığı içerir. Örneğin SMTP protokolü e-postanın kimden, kime gittiğini, kopyalarının kimlere gönderileceğini ifade edecek komutlar tanımlar. FTP protokolü ise

karşı traflan elindeki dosyaların listesini istemek, bir dosyayı bir taraftan diğerine aktarmak için gerekli komutları.

80'li yıllardaki durumda ister e-posta içinde ister dosya sunucusundan gelsin, belgelerdeki görsel ögelerin (fotoğraf, grafik, bölüm başlığı, formül, vb.) görüntülenmesi için bir dizi işlem gerekiyordu. 80'lerin sonunda İsviçre'deki CERN fizik araştırma merkezinde çalışan genç bir araştırmacı olan Tim Berners Lee, fizikçiler arasında bilimsel belgelerin paylaşımını kolaylaştırmak için bir sistem geliştirir. Bu sistemin bir tarafı HiperText Transfer Protokolü (HTTP) adını verdiği ve varolan TCP sistemi üzerinde çalışan bir protokoldü. Bu protokol bir tarafın (istemci) diğer tarafa (sunucu) elindeki bir belgeyi veya belgeyle ilgili bilgileri (boyutu, ne zaman güncellendiği, vb.) talep ettiğini belirtmesine, ve diğer tarafın da bu talebe cevaben hem belgeyi hem de onunla ilgili bazı bilgileri (boyutu, tarihi, ve en önemlisi de ne tiğ bir içeriği olduğu) göndermesine imkan veriyordu. Sadece HTTP tasarımının bu yönü bile yıllardır kullanılan FTP'ye göre ciddi avantajlara sahipti, çünkü bir dijital kitaplıktaki dökümanların güncellenip güncellenmediğini dökümanı tekrar transfer etmeksizin anlaşılmasını sağlıyordu ve bu da İnternet hızlarının düşük, transferlerin uzun ve zahmetli olduğu o dönemde çok önemliydi.

Tim Berners Lee'nin tasarımının ikinci yanı ise HTML (İng. HyperText Markup Language, HiperText İşaretleme Dili) adını verdiği bir içerik formatıydı. Bu içerik formatı göze hitap etmeyen ve daktilo yazısına benzeyen düz metinler yerine bir metinde başlık, büyük/koyu kelimelerin yerlerini belirtmeye, tabloları ifade etmeye, ayrı dosyalarda bulunan görsellerin metindeki yerlerini ve nereden alınıp gisterileceklerini belirtmeye, metinde bahsedilen ve aynı (veya farklı) bir dijital kitaplıktaki başka metinlerin tam yerini belirten, hiperlink adını verdiği belirteçler koymaya imkan sağlıyordu. O dönemde bir bilgisayar üzerinde zengin metin yazımı ve gösterimine imkan veren yazılımlar olmasına rağmen, sadece hiperlink fikri bile bu tasarımı son derece ilginç kılıyordu, çünkü bu linkler tüm İnternet içerisinde bir belgenin yerini mutlak olarak tarif ediyordu. Örneğin `http://www.w3.org/Amaya/Amaya.html` şeklinde bir link www.w3.org isimli bir dijital-kitaplık/sunucu içerisinde Amaya dizininde bulunan Amaya.html isimli belgeye işaret eder ve o sunucuya da http protokolü ile erişileceğini belirtir.

Bu tasarımını meslek camiası ile paylaşan Berners Lee camiadan aldığı yoğun tezahürat ve desteğin de yardımıyla kısa

zamanda hem HTTP dilini konuşarak döküman servisi yapabilen bir sunucu programı, hem de HTML diliyle işaretlenmiş metinleri sunucudan alıp ekranda gösterebilecek ilk tarayıcı (İng. browser) programı hayata geçirecekti. Berners Lee hiperlinklerle birbirine bağlanmış belgelere atfen tasarladığı bu sisteme 'World Wide Web' (www, Dünya Çapında Ağ) adını takmıştı. 1991'den itibaren hızla yaygınlaşan HTTP ve HTML hala İnternet'in kısa ama hareketli tarihindeki en önde gelen sıçrama olarak durmaktadır.

World Wide Web yaygınlaşır ve birçok şirkete servetler kazandırırken Tim Berners Lee bundan uzak durmuş ve geliştirdiği teknolojinin kamusal alanda kalması, bilim insanları ve öğrenciler için ulaşılabilir olması için çaba göstermiştir (Castells, 2001). Kendisi halen bu teknolojinin gelişimini destekleyen ve yönlendiren WWW Konsorsiyumunun (kısaca W3C, `http://w3c.org`) yöneticisi olarak çalışıyor. 1991'den bu yana HTTP tasarımı çok az değişti ve şu anda 1.1 numaralı sürümü kullanılıyor. Ancak tüm dünyadaki web kullanımı ve bu teknolojiden beklenen içerik zenginliği hızla arttığı için HTML dili sık sık yenilenip genişletilmiş ve şu anda beşinci sürümüne ulaşmıştır. Bu iki teknoloji İnternet'teki zengin kaynaklara erişmemizin en önemli aracı olmaya devam ediyor. W3C bir yandan bu standardın değişen kullanıcı ihtiyaçlarına uygun olarak gelişmeye devam etmesini sağlıyor, bir yandan ise herkese ulaşılabilir olamsı için kamu lisanslı ve ücretsiz programlar geliştirip dağıtıyor. Bu programlar arasında web sunucu programlarının yanısıra web tarayıcı ve düzenleyici Amaya programı var. Amaya bizim gibi kullanıcıların web sayfası inşa etmesini son derece kolaylaştırmakla kalmıyor, birçok ticari üründe bulunmayan gelişmiş özellikler de sunuyor. W3C ayrıca web teknolojilerini öğrenmek isteyenler için bol miktarda eğitim mazemesini web sitesinde ücretsiz olarak paylaşıyor.

1990'ların sonuna doğru web teknolojisinin e-ticaret, yani İnternet üzerinden mal ve hizmet satışı için de kullanılmaya başlaması bazı güvenlik sorunlarını ön plana çıkardı. Bu sorunları gidermek için dönemin önemli İnternet teknolojisi şirketi Netscape tarafından tasarlanan güvenli bağlantı katmanı (SSL; İng. Secure Sockets Layer) HTTP trafiğinin şifreli yapılmasına izin veriyordu. Bu teknoloji sadece HTTP değil, TCP/IP sisteminde taşınan her türlü veri trafiğini şifrelemeye imkan veriyordu. Bu sayede birçok protokolün SSL ile birlikte kullanımını ifade etmek için secure/güvenli anlamında HTTPS, SMTPS, FTPS gibi varyant-

ları kullanılabiliyor. Web tarayıcınızda Yahoo, Gmail, Facebook gibi sitelere kullanıcı girişi yaparken şifrenizi verdiğiniz için bu giriş sayfasının adresinin http yerine https ile başladığına dikkat edin. Şifre alışverişinin güvenliği için bu gereklidir. E-ticaret güvenliğinin SSL ile beraber çalışan bir bileşeni de güvenlik sertifikalarıdır. Bu sertifikalar dijital nüfüs cüzdanı yerine geçiyor ve yetkili, güvenilir kurumlar tarafından veriliyor. Ancak bu sertifikaların pahalı olması bireysel kullanıcılar için bir sıkıntıdır. Bu güvenlik sistemi bireysel kullanıcıları değil e-ticaret yapan şirketleri gözeterek tasarlanmıştır. Programcılar arasında geliştirilen bir topluluk çözümü ise PGP (İng. Pretty Good Privacy, Oldukça İyi Güvenlik) adı verilen bir sistem. Bu sistemde sertifikalar bir yetkili kurum tarafından verilmeyip herkes kendi sertifikasını üretiyor. Ancak camia içerisindeki bireylerin birbirlerinin sertifikasını onayladığını/tanıdığını belirtmesiyle bu dijital kimlikler güvenirlik kazanıyor.

HTTP yaygınlaşırken e-posta sistemi farklı bir işlevi (bireyden bireye haberleşme) görmeye devam etmiş ve popülerliğini yitirmemiştir. Anca ke-posta kullanımını kolaylaştırmak için bireylerin uzakta bir sunucudaki posta kutusuna erişimini sağlayan POP, POP3, ve en son, en kullanışlısı olan IMAP gibi protokoller geliştirilmiştir.

Bu protokol ve tekniklerin geliştirilmesine üniversiteler kadar özel sektör de katkı vermiştir. Ancak İnternet'in çok farklı paydaşları aynı ağda birbirine bağlayan doğası herhangi bir şirketin kendi teknolojisini diğerlerine dikte etmesine imkan vermiyor. Bu tür protokollerin standartlaşması her zaman IETF, ISC gibi meslek/sektör örgütleri bünyesinde, çok paydaşlı süreçlerle olabilmiştir. Örneğin Microsoft'un öteden beri kendi web tarayıcı ürünü olan İnternet Explorer'da pazardaki büyük payına güvenerek standart dışı/aykırı özellikler uygulama girişimleri her seferinde geri tepmiştir. Buna karşılık artık bir meslek camiası projesi olarak kollektif geliştirilen, kamu kaynak lisanslı Mozilla-Firefox web tarayıcısı ticari ürünler karşısında cazip, zengin özelliklere sahip bir alternatif olabilmektedir. İnternet teknolojisinin tüm gelişimine işlemiş olan kollektif kara verme, ve herkese açık olma özellikleri bir şekilde sürekli evrilerek yeniden ortaya çıkıyor.

IETF standartlarının gelişimine baktığımızda son yıllarda yaşanan önemli bir dönüşümün İnternet'in küreselleşmeyle beraber bununla ilgili teknolojilerin de farklı dilleri desteklenmesi

ihtiyacıyla ilgili olduğunu görürüz (Gençer et al., 2006). Sadece latin/Avrupa dilleri değil Asya ve Afrika dillerinin de İnternet içeriğinde/trafiğinde, alan adlarında kullanılabilmesi ihtiyacı İnternet yaygınlaştıkça kendini dayatmıştır. Günümüzde birçok teknoloji yerelleşme sorununu da gözönüne alarak geliştiriliyor.

Son dönemde Facebook, Twitter gibi web alanlarının popülerliği artsada bunlar yeni bir iletişim protokolüne ihtiyaç duymazlar. Varolan HTTp standardı ile çalışırlar. Öte yandan İnternet üzerinden telefon görüşmesi, TV/radyo yayını gibi uygulamaların yapılabilmesi için VoIP, RTP gibi bazı yeni protokoller geliştirilmiştir. Bu uygulamalardaki özgün ihtiyaç gerçek zamanlı, gecikmelerin çok az tahammül edilebilir olduğu bir veri transferidir (kimse kesik kesik gelen bir İnternet TV yayınına para vermez). Sözkonusu protokolle İnternet'in karmaşık, değişken trafik yoğunluğuna sahip ortamında bu tür bir veri transferine imkan verecek şekilde tasarlanmaktadır (örneğin veri zarflarının üzerine öncelikli geçiş hakkı verilmesi için bir işaret koyarak).

Yine burada bahsetmeye değer bir gelişme de Torrent, Kazaa, eMule gibi dosya paylaşım sistemleri. Doaya paylaşım trafiği şu anda İnternet trafiğinin en büyük ve en hızlı büyüyen kısmını oluşturuyor. Artık tamamen dijital olarak satıldıklarından dolayı bu paylaşım sistemlerinin kolay hedefi olan müzik ve filmlerin, ayrıca bilgisayar yazılımların paylaşımı, telif haklarına dayalı bu sektörlerin bütün feryat figanına rağmen engellenemiyor. Aynı İnternet'in kendisi gibi bir merkezi olmayan, bu yüzden peer-to-peer/eşler-arası tabir edilen bu sistemleri durdurmak mümkün değil çünkü kapısına kilit vurabileceğiniz bir merkez sunucuya ihtiyaç duymadan çalışıyorlar. Zaman zaman katılımcılardan birinin yakalanıp telif haklarından dolayı ceza aldığı haberleri çıksa da bu sistemleri kullananlar önüne geçilemezliğinin farkında. Bu durum bazı sektörleri ciddi bir değişime zorlamaktadır. Genel olarak İnternet iletişimininayrılmaz bir parçası olan serbest veri dolaşımı ilkesi dijital içerikle ilgisi olan bütün paydaşları (müzik, film, haber, e-kitap yayıncıları, sanatçılar, ve yazılım firmaları) telif hakları konusunda yeniden düşünmeye ve mevzilenmeye itiyor.

Bölüm 6

Bilgisayarı Kullanmak için Temel Beceriler: Yaygın yanlışlar ve alternatif doğrular

Bilgisayar kullanıcılarının çok büyük bir çoğunluğunun bilgisayarla yaptıkları işler benzerdir ve sadece birkaç yazılım kullanırlar. Çoğumuz bu programları kullanmayı okul arkadaşlarımızdan, dostlarımızdan, veya aile bireylerinden öğreniyoruz. Kulaktan kulağa aktarılan birçok konuda olduğu gibi bu konuda da doğrular kadar yanlışlar da yaygınlaşıyor ve daha da önemlisi birçok temel bilgi eksik kalıyor. Bu bölümde okurların büyük çoğunluğunun en azından bazılarını yaptığı ya da yapmayı denediği bazı işlerin nasıl yapıldığına ve bunları yapmamızı sağlayan bazı yazılımlara değineceğiz. Burada amacımız kapsamlı bir temel beceriler eğitimi vermek veya referans eğitim malzemesi sunmak değil. Amacımız daha ziyade birçok okurun gözünden kaçmış olabilecek bazı meselelere ve bunların çözümlerine işaret etmek, ayrıca yaygın kullanılan yazılımlar dışındaki kullanışlı bazı alternatiflere dikkat çekmek, böylece bilgisayarla ilişkinize eleştirel bir bakış getirmenizi ve bu ilişkiyi iyileştirici bir sorgulama yapmanızı

teşvik etmek. Bunu yaparken ne kadar nesnel olmaya çalışsam da bir bilgisayar kullanıcısı olarak kişisel deneyimlerimin işe karışması kaçınılmazdır. Bunun için en doğrusu bazı yorumların kişiselliğini vurgulama ve farklı durumlarda sizin deneyiminizin farklı olabileceğini belirtmek olacak.

Bu bölümde bahsedeceğimiz işler ve yazılımlar nelerdir? Herşeyden önce hepimizin, üstelik sürekli kullandığımız iki yazılım var: işletim sistemi ve pencere/masaüstü yöneticisi. Özel bir durumu olan işletim sistemi bir tarafa hemen hepimizin en çok kullandığı program web tarayıcısı ve e-posta programlarıdır. Bunun ardından 'oris işleri' diye adlandırabileceğimiz metin yazımı ve hesap tablosu kullanımı geliyor. Buraya kadar saydıklarımız sanırım okurların büyük bölümünün bilgisayar kullanım deneyimini kapsıyor, ve aşağıdaki bölümlerde bunlara teker teker değineceğiz. Elbette hemen hepimiz bunlara ilaveten bilgisayarda müzik dinliyor, resim veya video izliyoruz; ancak bunları yaparken pasif (izleyici, dinleyici) durumdayız ve bu konuda genel okuyucuya anlatılacak fazla birşey yok gibi görünüyor. Yine de mesleği gereği ses veya görüntülere müdahale etmesi gereken okuyucular için bu konuya da bir bölümde kısaca değineceğiz. Bir bölümü de bazı okurların, özellikle üniversite öğrencilerinin ihtiyaç duyabileceği istatistik hesaplama yazılımlarına ayırdım. En son bölüm ise binlerce alternatif yazılımın bulunduğu bu dünyada kendimize uygun araçları nasıl seçebileceğimize dair.

6.1 İşletim sistemi ve pencere/masaüstü yöneticisi

Çoğu okuyucu için işletim sistemi ve pencere yöneticisi ayrımı birşey ifade etmeyebilir. Bu da başlı başına bir sorun. Bölüm 4.5.2'i incelemiş okurlar işletim sistemi denilen yazılımın birçok önemli görevini saydığımızı, ancak bunlar arasında 'pencere yönetimi' diye birşey olmadığını farkedeceklerdir. En yaygın kişisel bilgisayar işletim türleri olan Mac veya PC tipi bir bilgisayar satın aldığınızda bir sistemle yüklü gelir, ve bu sistem hem üstlendiği işler gözümüze gözükmeyen işletim sistemini hem de ekranda gördüğümüz program pencerelerini ve masaüstünü kullanmamızı sağlayan pencere/masaüstü yöneticisini içerir. Ticari olarak satılan sistemlerin çoğunda bu ikisi birbirine o kadar yapışık bir paket olarak verilir ki ayrımını farketmeyiz.

Ticari sistemlerdeki bu duruma karşın birçok gayri ticari sistem kullanıcıya büyük ve değiştirilemez bir paket vermek yerine,

kullanıcının kendi zevk ve ihtiyacına göre seçtiği parçalardan bir sistem oluşturmasına imkan verir. Bahsettiğim sistemlerin tamamı ilk işletim sistemi olan (1960'lardan beri var) ve hala üniversitelerin ve bir kısım şirketlerin desteğiyle gelişmeye devam eden UNIX ailesine mensup işletim sistemlerini temel alıyor. Aradan geçen onyıllar içerisinde UNIX ailesine mensup pek çok işletim sistemi ortaya çıkmıştır, ancak aralarında çok yüksek bir doku uyumluluğu var. Yani birinde çalışan yazılımlar diğer hepsinde de çalışabiliyor. Üstelik daha çok üniversitelilerin katkısıyla geliştirildiklerinden bu yazılımlar kamuya açıktır, yani ücretsiz olarak edinip kullanabilir, hatta programlama biliyorsanız ihtiyacınıza göre değişiklikler yapabilirsiniz. Bütün bunlar da bu sistemleri kullandığınızda geniş bir yazılım geliştirme camiasının birikimlerinden faydalanabileceğiniz anlamına geliyor.

UNIX ailesi işletim sistemlerinden bir kısmı son derece özel ihtiyaçlar içindir. Örneğin NetBSD isimli bir UNIX İnternet trafiğini yöneten ağ cihazlarını çalıştırmak için kullanılıyor. Bilgisayar mühendisliği eğitiminde kullanılan eğitim amaçlı küçültülmüş çeşitleri, veya bazı şirketlerin sunucular için ayarladıkları yüksek performanslı çeşitler de var. Kişisel kullanım için uygun olan bir UNIX varyantı ise Linux adlı işletim sistemidir. İlk kez 1990'ların başında Finlandiyali genç bir üniversite öğrencisi olan Linus Torvalds tarafından geliştirilmeye başlanan bu sistem daha sonra birçok programcının, üniversitenin, ve büyüklü küçüklü bilişim şirketlerinin desteğiyle hızla gelişmiştir. Linux son yıllarda hızla yaygınlaştı ve kendi içinde birsürü çeşidi ortaya çıktı. Muhtemelen evinizdeki ADSL model veya uydu TV cihazı üzerinde, hatta belki cep telefonunuzda zaten bir Linux kullanıyorsunuz! Ancak bizi ilgilendiren kişisel bilgisayarlar için olan Linux çeşitleri. Ben bunların en yaygınlarından olan RedHat, Debian, ve Ubuntu dağıtımlarını kullandım ve hepsinden memnun kaldım. Esasen hepsi aynı çekirdek teknolojiyi içeren, kullanıcıya kolaylık için paketlenmiş, hedef kitleleri az biraz farklı dağıtımlar. Ubuntu Linux dağıtımını (bkz. `www.ubuntulinux.org` yeni başlayanlar için kurulumu kolay bir ürün olarak tavsiye ederim.

Önemli bir UNIX çeşidi de Mac marka bilgisayarların üzerinde çalışan OS-X sisteminin çekirdeiğin oluşturan, Darwin adı verilen işletim sistemi. Bu sayede UNIX ailesi işletim sistemlerinde çalışan yazılımların çoğu sorunsuzca Mac sistemlerinde de çalışabiliyor.

Linux başta olmak üzere UNIX varyantı işletim sistemleri

pencere/masaüstü yöneticisi olarak kullanabileceğiniz birden fazla alternatif sunarlar. Benim kişisel tercihim daha klasik bulduğum GNOME masaüstü yöneticisi, ki Ubuntu sistemini kurduğunuzda öntanımlı olarak bu kuruluyor. Bir de KDE isimli bir pencere yöneticisi çok yaygın, ki özellikle Mac'e alışkın olanlar bunu daha çok seviyor. Bunun dışında bazı özel ihtiyaçlara yönelik (örneğin DVD oynatıcı, uydu TV cihazı gibi özel koşulları olan cihazlarda kullanılanlar, veya çok düşük bellği olan sistemlere uygun olanlar) pencere yöneticileri var ancak burada bizi pek ilgilendirmiyorlar. İşin en güzel yanı şu ki eğer bunlardan birini kullanıp beğenmezseniz sisteminizi baştan silip kurmaya gerek olmadan başka bir pencere/masaüstü yöneticisi kurup kullanabilirsiniz. Ayrıca hem GNOME hem de KDE için geliştirici camiasının üretip paylaştığı her zevke uygun renk ve görünüm temaları bulunuyor.

Ubuntu Linux sisteminin kurulumu oldukça basittir, ancak bunun denemeden önce sabit diskinizde 10-12 GB kadar yer olmasına dikkat edin. Ubuntu kurulum CDsini web sitesinden (`www.ubuntulinux.org`) indirip bir CDye yaktıktan (veya bir arkadaşınızdan böyle bir CD edindikten) sonra bu CD bilgisayarınıza takılı halde iken bilgisayarı kapatıp açın. Bilgisayarınız CDdeki kurulum programını açacaktır (çok istisnai durumlarda bu gerçekleşmezse bilgisayr açılırken BIOS ayarlarına girip book seçeneklerine gözatmanız gerekebilir). Kurulum programı birkaç basit soruyla sizi yönlendirecektir. Bu kurulum sırasında size varolan sisteminizin disk kullanım alanını biraz küçültüp sabit diskte yeni sisteminiz için biraz yer açma imkanı verilecektir. Diskinizin büyüklüğüne ve doluluk durumuna göre bir miktar seçmelisiniz. Linux kurulumunun güzel yanı varolan sisteminizi tahrip etmemesidir. Eğer yeni sistemden hoşlanmaz veya başka bir sebepten eskisini açmanız gerekirse bilgisayar açılışında bunu yapabilirsiniz. Linux diğer sistemin kullandığı sabit disk bölümünü okuyabilir, bu yüzden dosyalarınızı aktarmak sorun olmaz. Linux ile kurulan pencere/masaüstü yöneticisi diğer sistemlerle büyük ölçüde benzer. ben Linux'a geçen birçok tanıdığımın ilk açılıştan itibaren ekranda aradıklarını kolayca bulabildiklerini gözlemledim.

Aşağıdaki bölümlerde bahsedeceğim yazılımların hemen tamamı hem Linux hem de diğer sistemler üzerinde çalışabiliyor. Bu yüzden buradaki malzemeden yararlanmak için Linux kullanmanız bir zorunluluk değil. Yine de kişisel tecrübem bu sistemle

yazılım kurmanın ve kullanmanın diğerlerine göre daha rahat olduğu yönünde; özellikle de bu sistemin tamamı binlerce kamu lisanslı ve ücretsiz yazılım paketine erişmek için ayarlandığından dolayı.

6.2 Web tarayıcısı ve e-posta kullanımı

Web sayfalarına erişmek ve e-posta alip vermek hepimizin en sık ihtiyaç duyduğu işlerden. Web sayfalrına erişmek için pek çok web tarayıcı yazılım mevcut. Bunların bazıları Internet Explorer gibi ticari ürünl, bazıları Opera gibi ticari olup ücretsiz sürümü de bulunan, başka bir bölümü ise kamu lisanslı ve ücretsiz ürünler. Ücretsiz web tarayıcıları arasında en bilineni Mozilla vakfının desteklediği Firefox isimli tarayıcı (`www.mozilla-europe.org`). Firefox geliştirici camiası HTML5 gibi yeni standartları ürüne hızla entegre ediyorlar, ayrıca ürüne çoklu panel gibi kullanışlı özellikler eklemek konusunda iyi bir geçmişleri var. Firefox'un bir başka avantajı da dünya çapındaki kullanıcı ve geliştirici camiasının ürettiği ve paylaştığı birçok eklentisi olması. İhtiyacınıza göre indirip kurabileceğiniz, çoğu ücretsiz bu ektentiler web sayfalarını tercüme etmekten, arama motorlarını veya sosyal ağları daha kolay kullanmaya kadar pek çok işe yarayabiliyor.

Bazı e-posta kullanıcıları e-posta alıp vermek için de Gmail, Yahoo gibi servislerin web arayüzlerini kullanıyor. Eğer bu işi daha kolay yapmak isterseniz bunun için bir e-posta istemci yazılımı kullanabilirsiniz; özellikle de birden fazla e-posta hesabını tek bir program penceresinden kontrol etmek istiyorsanız. Bunun için kullanabileceğiniz kamu lisanslı bir yazılım -yine Mozilla vakfının- Thunderbird programı. Ayrıca Linux kullanıcıları için sadece bu ssitemde çalışab Evolution da iyi bir seçenek, çünkü ayrıca ajanda yönetimi sağlıyor.

Hangi e-posta programını kullanırsanız kullanın e-posta hesaplarınız için ayarları yapmak biraz karmaşık gelebilir. E-posta gönderimi (SMTP) ve alımı (POP3 veya IMAP) için ayrı sunucu ayarları gerekir. Bu ayarlar gönderme ve almada kullanılan farklı protokol standartlar, güvenli bağlantı seçenekleri, ve tabi kullanıcı hesap adı ve şifresini içerir. E-posta servis sağlayıcınız (Gmail, Yahoo, veya işyerinizin bilgi-işlem departmanı) size bu bilgileri sağlayacaktır. Bu zahmeti göze almanın karşılığında e-postalarınızı daha rahat oluşturmak, hızlı arama yapmak, mesajları belli kurallara göre yönlendirmek, dosyalara tasnif etmek gibi

avantajlarınız olacaktır.

Hepimiz e-posta kullanmamıza rağmen çoğumuzun az dikkat ettiği bir nokta alıcı adreslerini 'Kime' (To), 'İlgili kopyası' (CC), veya 'Gizli kopya' (BCC) olarak işaretleme meselesidir. Bunlar özellikle kurumsal yazışmalarda işinize yarar. 'Kime' olarak işaretlenen alıcılar mesajın doğrudan muhatabıdır. 'İlgili kopyası' alıcılar ise cevap vermeleri beklenmeyen ama yazışmadan haberdar olmaları istenen alıcılardır. Bir mesaj aldığınızda sizin dışınızda her iki türden de alıcıların kimler olduğunu görürsünüz. 'Gizli kopya' alıcılar ise diğerlerinden farklıdır. Diğer alıcılar mesajın gizli alıcıya/alıcılara da gittiğinden kesinlikle haberdar olmazlar.

6.3 Metin yazımı

Metin yazımı bu kadar yaygın bir ihtiyaç olmasına rağmen çok az standartlaşabilmiş bir alandır. Burada metinden kastımız zengin metinler, yani içinde büyük başlıklar, koyu-italik kelimeler, tablolar, fotoğraf veya diğer görsel malzeme bulunan metinler. Buna kraşılık bir de düz metin var ki bu metin türü sadece harfleri, rakamları, ve klavyeye basarak elde edebildiğimiz diğer sembolleri içerir. Görsel bir zenginleştime barındırmayan düz metinleri sisteminizdeki basit düzenleyici (İng. editor) programlardan biriyle kolayca oluşturabilirsiniz (örneğin Microsoft Windows sistemindeki notepad, Linux'taki gedit programları). Düz metinleri not almak için kullansak bile okunakli ve cazip olmadıklarından bize yetmezler.

Zengin metinler oluşturmak için birbiriyle yarışan birçok yöntem ve bunlara karşılık gelen farklı yazılımlar var. Bu farklı yöntemlerin herbirinin kullandığı bir metin biçimi veya diğer adıyla formatı vardır. Bizim en sık karşılaştığımız zengin metin formatları HTML ve PDF denilen formatlar. Ne var ki metin yazarken bunlardan ziyade Microsoft Word veya OpenOffice kelime işlemci programları ve formatları kullanılıyor.

Bu ayrılığın sebebini açıklamakta yarar var. İlk kez Adobe şirketi tarafından geliştirilen PDF İngilizce Portable Document Format, yani taşınabilir metin formatı olarak adlandırılmıştı çünkü ortaya çıktığı dönemde Microsoft Windows, Mac, ve Linux sistemlerinde kullanılan farklı zengin metin formatları mevcuttu, ancak bir sistemin formatı/formatlarıyla oluşturulmuş metin başka bir sistemde okunurken sıkıntı oluyordu. PDF'e taşınabilir format

denmesinin sebebi bu sorunu çözmesiydi.Adobe şirketinin PDF formatı konusunda liberal bir patent ve açıklık politikası benimsemesinin de etkisiyle bu zengin ve yeterli format çok kısa sürede yaygınlaştı.Ancak PDF formatı aslı başka bir formatla ve yazılımla hazırlanmıl metinleri başkalarıyla paylaşmak için kullanılıyor. Hemen her zengin metin formatı PDF'e çevrilebiliyor. Bir metni e-postayla birine gönderirken bu şekilde PDF'e çevirerek gönderirseniz herkesin hayatı daha kolay oluyor. Ama asıl metni doğrudan PDF olarak yazmak için Adobe şirketinden satın aldığınız bir yazılım kullanmanız gerekir, ki çoğu bilgisayar kullanıcısının böyle bir imkanı yoktur. Bu yüzden PDF bir metin hazırlama formatı değil metin değiş-tokuş formatı olarak kalmıştır.

HTML konusunda PDF'teki gibi bir sıkıntı yok. Elinde web tarayıcısı olan biri (herkes!) bir HTML metni görüntüleyebilir. HTML metinleri düz metin düzenleyicileriyle hazırlamak mümkün olduğu gibi (bireaz sonra bunu yapacağız) bunun için kullanılabilecek herkesin rahatlıkla bulabileceği ücretsiz yazılımlar da var. Ancak HTML'in diğer metin formatlarından önemli bir farkı kağıda baskı için değil de ekranda gösterim için olmasıdır. Bu yüzden HTML'de 'sayfa' kavramı yoktur. Oysa zengin metin formatlarının önemli bir özelliği metni sayfalara yayması, metindeki bölümlerin, şekillerin, dipnotların, vb., yerleşimini sayfa kesmelerine göre ayarlamasıdır. Bu yüzden HTML formatı kurumsal yazışma, mektup, makale, veya kitap yazımı gibi işlere pek uygun düşmez. Biz yine de ilk olarak HTML'i ele alacağız çünkü bu format hem işinize yarar hem de zengin metin yazımıyla ilgili bazı meseleleri incelemek için çok uygun.

6.3.1 HTML ile içerik ve stil ayrımına bir bakış

Bu bölümde yapacağımız HTML örneklerini bir düz metin düzenleyici program kullanarak yazabilirsiniz. Daha sonra bu dosyaları web tarayıcısı ile görüntüleyeceksiniz. Bir kez bazı konuları inceledikten sonra, bölümün sonunda HTML metinleri çok daha kolay oluşturmanızı sağlayacak bazı yazılımlardan bahsedeceğim. Burada HTML'in özelliklerinin çok küçük bir bölümünü ele alacağım. Meraklı okurlar `http://w3c.org` sitesindeki zengin eğitim malzemelerinden yararlanarak bu konudaki bilgi ve becerilerini arttırabilirler.

Bütün HTML formatı düz bir metinin zenginleştirmek istedi-

ğimiz parçalarını birtakım etiketler ile işaretlemeye dayanır. Bir örnekle başlayalım:

```
<html>
 <head>
  <meta content="charset=utf-8" />
 </head>
 <body>
  <h1>HTML'e Giriş</h1>
  <h2>HTML nedir?</h2>
  <p>HTML ilk kez Tim Berners Lee'nin geliştirdirdiği
  bir zengin metin
  formatıdır</p>
  <p>Şu anda HTML'in 5. sürümü kullanılmaktadır</p>

  <h2>HTML nasıl kullanılır?</h2>
  <p>Web tarayıcınızla dosyayı açık ve keyfini
  çıkartın</p>
 </body>
</html>
```

Bu ornekteki etiket kalabalığı sizi ürkütmesin. Etiketlenen bütün parçalar başına <etiket> sonuna da </etiket> konularak işaretleniyor. Bu metinde her zengin metinde karşılaşacağınız bazı bileşenler var. En başta paragraflar, ki bunlar .. olarak etiketlenmiş. Düz metin dosyasında satır kesmeleri ve boşluklar koysanız bile web tarayıcısı bunlara aldırmaz, sadece paragraf etiketlerini dikkate alır. Web tarayıcısında sayfanın görünüşü şekil 6.1'de verilmiştir. İkinci zenginleştirme bölüm başlıkları, ki bunlar h1 ve h2 etiketleriyle işaretlenmiş. Buradaki h İngilizce heading, yani başlık kısaltması. Bu örnekte 1. ve 2. seviye başlıklar kullandık. Bunları bir kitaptaki ünite ve bölüm başlıklarına karşılık düşünebilirsiniz. HTML standardı 6. seviyeye kadar (yani h6) gitgide yazıtipi küçülen başlıklar kullanmamıza izin veriyor; yani çok büyük metinleri de gözönüne alarak bölüm, altbölüm, alt-altbölüm diye devam edebilir.

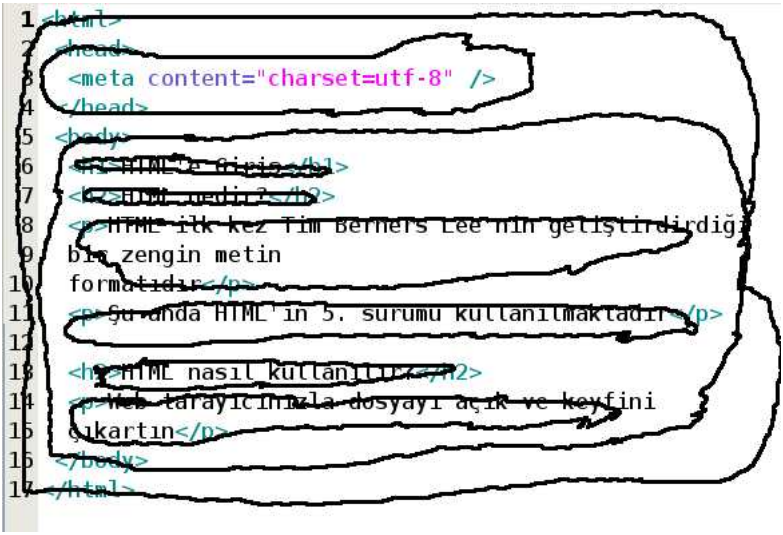
Örneğimizdeki diğer etiketler html, head (İng. baş), ve body (İng. vücut). HTML belgenin tamamı html etiketine alınıyor, asıl metin vücut etiketi ile çevrelenmiştir. Baş etiketi metnin bir parçası olmayan ancak genel özelliklerine dair bilgiler verir. Bu örnekte bas etiketinin içindeki 'meta' etiketi metnin utf-8 karakter seti, yani İngilizce dışındaki dillerde bulunan harfleri de yazmaya

Şekil 6.1: Basit HTML örneği

uygun bir kodlama içerdiğini belirtiyor. Baş kısmını az sonra yine kullanacağız.

HTML etiketlerinde dikkat etmeniz gereken bir özellik işaretlenen parçaların birbirini içerimesi ancak asla kesişmemesi. Bu HTML belgesindeki etiketleri içiçe kümeler gibi çizersek bu daha açık görülür (şekil 6.2). Şekilde gördüğünüz gibi HTML etiketlemesiyle oluşan parça sınırları asla birbirini kesmiyor. Aynı bir kitaptaki gibi bir bölüm bitmeden diğeri başlamıyor, veya başlık sınırı bitmeden paragraf başlamıyor, vb. Bu mantıksal ve kesin bir kuraldır.

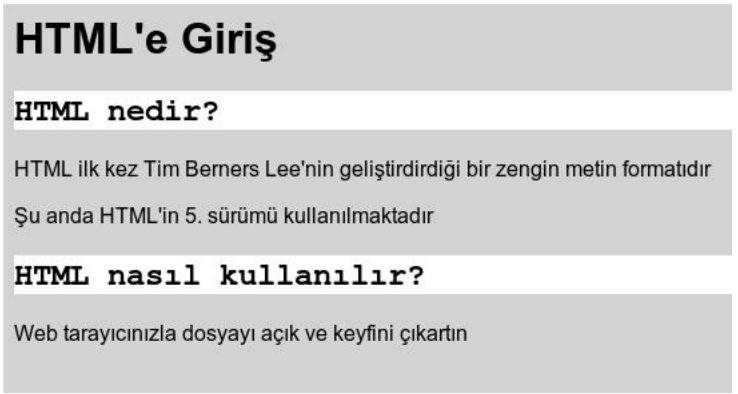
Şimdi metin yazımıyla ilgili pek çok kullanıcının yanlış yönlendirildiği bir konuya gelelim: metin stili. Metin stili bir metnin yazı tipi, büyüklüğü, başlık biçimi, renkleri gibi özellikleridir. Pek çok okur örneğimizdeki gibi bir metnin bu özelliklerini kendi zevkine veya metnin kullanım amacına uygun olarak değiştirmek isteyecektir. Birçok tanıdığımı bu işi yaparken gözlemledim. Kullanıcılar metne istedikleri stili vermek için metin parçalarını fare ile işaretleyip teker teker bu parçaları istedikleri görünüme kavuşturmaya çalışır. Ancak bu son derece verimsiz bir yöntem. Çünkü tüm metnin stilini değiştirmek, örneğin bütün bölüm başlıklarında farklı bir yazıtipi veya büyüklüğü kullanmak isterseniz o zaman dönüp yine herbiri için bu zahmetli işlemi tekrarlamak gerekecektir. Üstelik metnin içeriği ile stil işaretleri (ortaya çıkacak



Şekil 6.2: HTML kaynak metnin yapısı

HTML etiket karmaşasını bir düşünün!) karmakarışık olacaktır. Bugün bu yanlış yöntemle yapılmış birçok web sayfası vardır ve bu sayfaların HTML kaynak koduna bakarsanız metnin bazı parçalarının `<font>` etiketi ile bu şekilde işaretlendiğini görürsünüz. Ne ironi ki HTML standardının yeni hazırlanan beşinci sürümünde bu etiket kaldırılıyor; çünkü varlığı bile bu yanlış kullanımın web sayfalarında gereksiz ve çok sayıda kullanılmasına sebep oluyor.

Burada metin stilinin içeriğinden ayrı belirlendiği bir yöntem göreceğiz. HTML örneğimizdeki head/baş bölümünün asıl metnin bir parçası olmayan, ancak onun genel özelliklerine dair bilgiler veren bir bölüm olduğunu söylemiştik, ki bunların en önemlisi metin stilidir. Metin stilini değiştirirken metindeki farklı türden bileşenler için kullanılacak stilleri belirleriz. Örnek dosyamızı şu şekilde değiştirelim. Stil bilgilerini baş bölümüne bir 'style' etiketi içinde vereceğiz:



Şekil 6.3: HTML stil kullanımın görünüme etkisi

```
<html>
 <head>
  <meta content="charset=utf-8" />
  <style>
    body {font-family: Arial;
          background-color: lightgray; }
    h2   {font-family: Courier;
          background-color: white;}
  </style>
 </head>
 <body>
  <h1>HTML'e Giriş</h1>
  <h2>HTML nedir?</h2>
  <p>HTML ilk kez Tim Berners Lee'nin geliştirdirdiği
  bir zengin metin
  formatıdır</p>
  <p>Şu anda HTML'in 5. sürümü kullanılmaktadır</p>

  <h2>HTML nasıl kullanılır?</h2>
  <p>Web tarayıcınızla dosyayı açık ve keyfini
  çıkartın</p>
 </body>
</html>
```

Bu dosyanın ekranda görünümü şekil 6.3'de verilmiştir. Örne-

ğimizdeki metin stilinde iki bileşenin stilini değiştirdik. Birincisi metin gövdesi stili için yazıtipi ve arkaplan rengini değiştirdik, ki burada belirlediğimiz stil bütün metni etkiliyor. Çünkü şekil 6.2'te gördüğümüz üzere gövde etiketi başlık ve paragraf etiketlerini sarmalıyor. Ancak ikinci stil bileşeni sadece 2. seviye başlıları etkiliyor. Gördüğünüz gibi stili belirleyerek bütün 2. seviye başlıkların görünümünü bir hamlede değiştirmiş olduk. Metin yazımında verimli çalışmak için, hele hele uzun bir metin sözkonusuysa içerik ve stil ayrımını bu şekilde gözetmek elzemdir. Örneğimizdeki stil tanımının sözdizimini, veya stil için başka hangi seçeneklerin sözkonusu olduğunu burada ele almayacağım. Oldukça fazla stil seçeneği sözkonusudur. Bunları da yine `w3c.org` sitesindeki eğitim dökümanlarında arayıp bulmak mümkün.

HTMl metinleri bu şekilde yazmak eğiticidir, ama bir kez temel kavramları öğrendikten sonra daha hızlı çalışmak sitersiniz. Bunun için OpenOffice programı kullanılabilir, çünkü HTML formatında içe/dışa aktarım yapabiliyor. Ancak benim tavsiyem World Wide Web Konsirsiyum'un ücretsiz dağıttığı Amaya programı. Bu program hem çok becerikli, hem de HTML standartlarına tam uyumlu metinler oluşturuyor. Bu programı da konsorsiyum web sitesinden edinebilirsiniz.

6.3.2 OpenOffice kelime işlemci ile metin yazımı

OpenOffice kamu lisanslı bir yazılım olduğu için kolayca edinip kullanabilirsiniz (`www.openoffice.org.tr`). Bu yazılımdan uzun uzun sözetmeyeceğim, çünkü menüleri zaten kullanıcıyı kolayca yönlendiriyor. Ayrıca öğrenmek için çok kullanışlı kılavuzlar da web sitesinden bulunabiliyor. Ancak birkaç noktaya değinelim. Aslında OpenOffice gibi metin formatları da HTML'e benzer; metnin parçaları istenen görsel etkiye göre etiketlenmiştir. Ancak HTML'den farklı olarak bu tür metin formatları bir düz metin düzenleyici ile incelenemez ve değiştirilemez çünkü klavyede olmayan semboller kullanır. Bunun için bu tür formatlara sahip metinler anca o formata uygun bir yazılımla oluşturulur ve gmrüntülenir. OpenOffice programının benim beğendiğim bir yönü pek çok değişik formatı gösterebilmesi ve düzenleyebilmesi.

OpenOffice kelime işlemci ile metin yazarken HTML metinleri için sözettiğimiz içerik ve stil ayrımını kullanmayı mutlaka deneyin. Muhtemelen alışık olduğunuz şekilde metnin bir parçasını fare ile işaretleyip yazıtipi ve büyüklüğünü değiştirmek yerine Format

menüsünden 'Stil ve formatlama'yı seçin ve burada 'öntanımlı metin' (HTML'deki body/gövde karşılığı), başlık 1,2, vb., stilleri değiştirerek metinlerinizin nasıl değiştiğini görün.

OpenOffice'in kullanışlı özelliklerinden biri de metinleri PDF formatında dışarı aktarabilmesidir. Ayrıca bir metin üzerinde birden fazla kişi çalışıyorsanız herkesin metinde ne değiştirdiğini işaretleme imkanı (Düzenle->Değişiklikler menüsü), hassas belgeler için belgeyi şifreleme imkanı, vb., özellikleri vardır, ki bunların hemen hepsi diğer zengin metin düzenleyici programlarında da bulunuyor.

6.3.3 LaTeXile metin yazımı

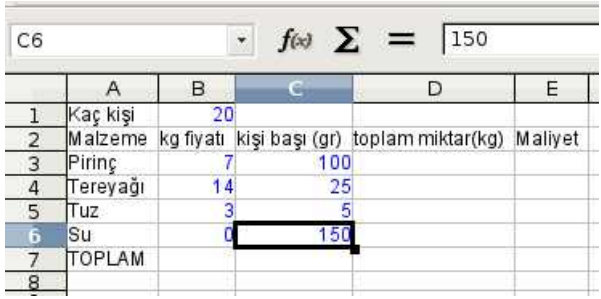
Pek çok okurun muhtemelen adını duymadığı LaTeXsistemine metin yazımının nirvanası demek yanlış olmaz. Bu sistem 1970'li yıllarda ünlü bilgisayar bilimci Donald Knuth tarafından bilimsel makaleler, özellikle matematik ve fizik metinlerinin yazımı için geliştirilmişti. matematik formüllerin yazımına imkan vermesinin yanısıra bölümlerarası referanslar, kaynakça referansları, vb., konularındaki becerileri bu sistemi hala büyük metinler (kitap, rapor, vb.) için en uygun araç kılıyor. Elinizdeki kitap ta bu sistemle hazırlanmıştır.

LaTeX'i kullanmaya başlamak HTML'den pek farklı olmamakla beraber bir nebze alışması zor olabilir. Çünkü bu sistemde düz metin düzenleyici ile yazdığınız LaTeXformatlı metni bir yazılımla işlemeniz, ve bir başkasıyla görüntülemeniz gerekir. Yine de üniversite öğrencileri ve akademisyenlerin bu sistemi öğrendiklerinde vazgeçemeyeceklerini düşünüyorum. LaTeXöğrenmek için gerekli yazılım ve malzemeyi `tux.org` ve `http://www.latex-project.org/guides/` sitelerinde bulabilirsiniz.

6.4 Veri tabloları

Okurların bir kısmı muhtemelen bazı işler için veri tablosu kullanmıştır (diğer adıyla hesap çizelgesi); mesela firmanızın bazı hesaplarını tutmak veya evde yaptırdığınız kapsamlı bir tadilatın maliyetini hesaplamak için. Veri tablosu kullanmak için Microsoft'un Excel programının yanısıra OpenOffice yazılım süitindeki hesap çizelgesi programı da kullanılabilir. Bu iki yazılım birbirine son derece benzediğinden burada bahsedeceklerimiz ikisi için de geçerlidir.

Veri tablosu iki boyutlu bir matristir ve birbiriyle ilişkili sayı-



Şekil 6.4: Örnek veri tablosu

ları veya sözcükleri yerleştirmek, bunlar üzerinden bazı hesapları yapmak için kullanılabilir. İki boyutlu matris kullanmak bizlere çok doğal geliyor çünkü kağıt veya bilgisayar ekranı gibi verileri incelediğimiz ortamlar da iki boyutludur.

Bir örnekle başlayalım. Örneğimiz bir restoranda hazırlanacak pilav için gerekli malzeme miktarı ve maliyetinin hesaplanması olacak. Başlangıç tablomuz şekil 6.4'de görülüyor. Başlangıç tablomuzda pilav tarifine dair ham bilgiler var: malzeme adı, kişi başı miktarı, kg fiyatı, ve kaç kişilik pilav yapmak istediğimiz bilgisi. Bu verilerin ilişkisel düzenlenişine dikkatinizi çekerim. Örneğin tüm fiyat bilgileri aynı sütunda. Bir malzemeye ilişkin bütün bilgiler ise aynı satırda. Satır ve sütunları tersyüz etmek te mümkün ama bu yerleştirme bana daha doğal geldiği için bu şekilde yaptım. Sözkonusu veri tablosunun içeriği, örneğin öğrencilerin bir dersten aldıkları notlar, veya bir dükkanın ürün-fiyat listesi olsaydı da buna benzer bir düzen olacaktı. İki boyutlu matrisler bu türden basit ilişkisel verileri yerleştirmek için çok uygundur.

Şu anda tablomuzda verili bazı değerler var. Biz bunlardan yola çıkarak başka değerleri hesaplamak istiyoruz. OpenOffice veya Excel'de matrisin herhangi bir hücresine değer girmek değil de başka hücrelerde varolan değerlerden yeni bir değer hesaplatmak istiyorsak o hücreye eşittir (=) ile başlayan ve hesabın nasıl yapılacağını söyleyen bir aritmetik ifade yazarız. Bu ifadelerde varolan diğer hücrelerdeki değerlere atıfta bulunmamız gerekecektir. Bunu da sözkonusu hücrenin sütun ve satır kodunu yanyana yazarak yaparız. Örneğin C3 kodu bizim tablomuzdaki kişi başı

pirinç miktarına karşılık geliyor. Öncelikle örneğimiz üzerinde her malzemenin toplam miktarını hesaplamalıyızç Pirinç için bu değerin yerleşeceği D3 hücresine şu ifadeyi yazın:

```
=C3*B1/1000
```

Böylece kişi başı gram cinsinden pirinç miktarını kişi sayısı ile çarpıp, bir de kilograma çevirmek için bine bölerek toplam pirinç miktarını kg cinsinden buluyoruz. Siz bu ifadeyi yazıp ENTER'a basarak girince program hemen hesabı yapıp bu formül yerine sonuçta bulunan değeri gösterecektir. Ama formülü kaybetmez ve D3 hücresinin üzerine tekrar giderseniz formülü görüp değiştirebilirsiniz.

Benzer formülleri pilavda kullanacağımız diğer malzemeler için de yazmamız gerekiyor. Pilav gibi basit bir yemekte bu sorun olmayabilir, ama çoğu zaman tablolarımız büyüktür ve benzer formülü tekrar tekrar yazmak çıldırtıcı olur. Bu yüzden veri tablosu yazılımları bir hücreye yazılan formülü o hücrenin sağ alt köşesini fare ile tutup benzer hücrelerin üstüne çekerek bu işlemi bizim için yapar. Bunu yaparken atıfta bulunulan hücre numaralarını da uygun şekilde kaydırarak değiştiri. Fare ile bu işlemi yaparsanız D4 gözüne (toplam tereyağı miktarı) şu ifadenin geldiğini görürsünüz:

```
=C4*B2/1000
```

Burada formül bir alt hücreye uyarlanırken OpenOffice C3 yerine C4, B1 yerine B2 koyarak uyarlama yaptı. C4 uyarlaması doğru, ancan B2 yanlış bir atıf. Kaydırma-uyarlama işlemi sirasında B1 hücresine (toplam kişi sayısı) yapılan atıfın değişmesini istemiyoruz. Bunu programa belirtmek için D3 hücresini şu şekilde değiştirin:

```
=C3*$B$1/1000
```

Şimdi tekrar hücreyi fare ile köşesinden tutup sütun boyunca aşağıya çekin. Bu kez D4 hücresine

```
=C4*$B$2/1000
```

ifadesinin geldiğini görürsünüz ki bu da istediğimiz atıftır. Orijinal atıftaki $ işaretleri satır ve sütun referansının 'mutlak' referans olduğunu belirtiyor. Buna karşılık bu işareti içermeyen C4 gibi referanslar 'göreli' referanslardır. Formül uyarlama işleminde mutlak referanslar uyarlanmaz.

Eğer veri tablonuza ,sütun veya satırlar ekler ya da silerseniz yazılım kullandığınız referansları da ona göre otomatik olarak değiştirecektir.

Şimdi pilavımız için malzeme maliyetlerini hesaplayalım. Bu daha kolay çünkü bütün referanslar göreli. E3 hücresine şunu girin:
`=B3*D3`
böylece toplam pirinç miktarı ile fiyatını çarparak bu malzemenin maliyetini bulmuş oluyoruz. Yine bunu fare ile sütun boyunca aşağı çekerek diğer malzemeler için de toplam maliyeti buluruz.

Son olarak toplam malzeme miktarını ve maliyetini hesaplayacağız. Bunu şu ana kadar öğrendiklerimizi kullanarak D7 hücresinde şu formülle yapabiliriz:
`=D3+D4+D5+D6`
Ancak daha uzun tablolarda böyle bir ifadeyi yazmak çok sıkıntılı olurdu. Bunun için veri tablosu programları bize bazı aritmetik işlevler sağlar, ve bu işlevlerde bir dizi hücreye D3:D6 gibi bir ifadeyle tek hamlede atıf yapmamıza imkan verir. Böylece toplam malzeme miktarını D7 hücresinde şu formülle bulabiliriz:
`=TOPLAM(D3:D6)`
Benze şekilde E7 hücresine
`=TOPLAM(E3:E6)`
girerek toplam maliyeti de buluruz. Sisteminiz İngilizce ise 'TOPLAM' yerine 'SUM' yazmanız gerekebilir. Veri tablosu yazılımları TOPLAM işleminin dışında ortalama ve standart sapma, minimum/mazimum bulmaktan tutun da trigonometrik işlemlere kadar birçok işlev sağlarlar. Bunları programda hücre içeriğini değiştirdiğiniz yerin solunda formül butonuna basınca çıkan listeden inceleyip kullanabilirsiniz.

Bunların dışında veri tablosu yazılımlarında birden fazla tablo kullanmak, hesaplarda başka bir tablonun hücrelerine atıfta bulunmak mümkündür. Ayrıca bir sütundaki verilerden grafikler üretmek te mümkündür (her ne kadar grafik kalitesi pek iyi olmasa da). meraklı okurlarımız bunun için OpenOffice programının web sitesindeki eğitim dökümanlarından yararlanabilirler. Programda yazıtipi veya büyüklüğünü değiştirmek, hücreleri veya sütunları/satırları renklendirmek, vb. işlemler menüden kolayca bulunabilir. Bir blok hücreyi fare ile seçtikten sonra bir stil değişikliği yaparsanız bütün seçili hücrelere uygulanacaktır. Ayrıca sütun ve satır etiketlerinin üzerinde fareye basarak veya basılı tutup gezerek sütun veya satırları bütün olarak seçime dahil edebilirsiniz.

Çok farklı veri tablosu yazılımları kullanıldığı için aynı metinler konusunda PDF formatının sağladığı gibi bir ortak değiş tokuş

formatına ihtiyaç duyulmuştur. Bu format CSV (İng. comma separated values, virgülle ayrılmış değerler) formatı olarak anılır. Hemen her veri tablosu yazılımı tabloları bu formatta dışa ve içe aktarmaya imkan verir. Ancak bu basit format formülleri tutmaz, yanlızc hesaplanmış değerleri alıp-verir. Yine de veri tablosunda tutulan bilgileri örneğin bir istatistik programına aktarıp analiz etmek iççin kullanılabilir. Bunun dışında OpenOffice hesap çizelgesi programı Excel dahil birçok format için dışa/içe aktarma desteği sağlamaktadır.

6.5 Ses, fotoğraf, grafik, ve video

Bu konuda kişisel tecrübem çok kısıtlı, bu yüzden bazı kullanışlı ama az bilinen yazılımlara işaret etmekle yetineceğim.

Ses dosyalarını çalmak için pek çok program bulunuyor. Linux üzerinde RythmBox, QuodLibet gibi programlar hem dijital müzik arşivinizi kolayca kullanmanızı sağlar hemde İnternet radyo yayınlarını bulup çalabilirler. Bunların yanısıra yine Linux üzerinde SoundConverter programı ses dosyalarını farklı formatlara (mp3, ogg, flac) dönüştürmek için işinize yarayabilir. SoundJuicer programı da ses CD'lerini bu formatlardan birine çevirip sabit diskteki arşivinize eklemek için kullanılabilir. Bu ses formatları içinde flac sesi kayıpsız sakladığı için tercih edebileceğiniz bir format.

Bireysel kullanım dışında profesyoneller için XMMS programını önerebilirim (`xmms.org`). Bu program çalma kalitesini kontrol etmek için bir ekolayzer gibi gelişmiş ses kontrolleri sağlıyor. Ayrıca bir ses yayınını ve çalma listelerini birkaç kişinin beraber kontrol etmesine izin veren bir yapısı var.

Fotoğrafları görmek için birçok program mevcut. Ancak bazen bunları kırpmak, format dönüştürmek, parlaklığıyla oynamak gibi işlemlere ihtiyaç duyabilirsiniz. GIMP (`gimp.org`) yazılımı bunları ve çok daha fazlasını yapabilen, ticari bir ürün olan Adobe Photoshop muadili ama kamu lisanslı bir yazılım. Esasen Linux/GNOME için geliştirilen bu yazılım artık Mac ve Windows üzerinde de çalıştırılabiliyor. Çok fazla işlevi olduğundan profesyonel olmayanlar için menülerde aradığınızı bulmak biraz zor olabilir, ama yine de akla ilk gelen kamu lisanslı alternatiftir. GIMP'i deneyen tanıdığım profesyoneller alıştıkları ticari ürünlere kıyasla daha az kullnışlı bulduklarını, ancak yine de işlevsel olarak yeterli olduğunu söylüyorlar.

Eğer broşür tasarımı gibi grafik tasarım işleriyle uğraşıyorsanız Scribus yazılımından da yararlanabilirsiniz (`www.scribus.net`). Bu yazılımı GIMP ile birlikte kullanarak pek çok grafik işlemi yapmak mümkün görünüyor.

Video oynatmak için de birçok alternatif var, ancak VLC yazılımını tavsiye edebilirim (`www.videolan.org/vlc`). Linux, Mac, ve Windows üzerinde çalıştırılabiliyor, ve bu ürünü diğer alternatiflere göre daha kaliteli ve yetenekli buluyorum. Video işleme ise diğer işlere göre oldukça karmaşık bir işlem. kendi videolarınızı işlemden geçirmek (kesip birleştirmek, başlıklar eklemek, vb.) için OpenMovieEditor'ü (`www.openmovieeditor.org`) deneyebilirsiniz. Profesyonel yazılımlarla karşılaştırılamayacak kadar basit ancak bunlara ulaşmanın zorluğu düşünülürse mütevazi projeler için denenebilir.

6.6 İstatistik: veri analizi, grafik, ve modelleme

Günümüzde istatistik hesaplamalar yapmak için kullanılan en yaygın yazılımlar SPSS ve R. SPSS ticari bir ürün, ülkemizde oldukça yaygın, ve herşeyi fare ile yapmaya alışkın okurlara daha sıcak gelen bir yazılım. R is akademik camianın katkılarıyla geliştirilen, kamu lisanslı bir yazılım (Venables and Smith, 2002). R kullanmak fare tıklamaları yerine komutlar yazılarak yapıldığı için çoğu kullanıcıya ulaşılmaz görünebiliyor. Ancak bu duygusal engel bir yana bu türden bir kullanım istatistik gibi karmaşık bir alanda kullanıcıya daha fazla kontrol sağlıyor. Üstelik R son derece zengin istatistiksel analiz, görüntüleme, ve modelleme bileşenleri sağlıyor. Linux'un yanısıra Mac ve Windows'da da kullanılabiliyor. Program `www.r-project.org` adresinden ücretsiz edinilebiliyor. Burada R programının kullanımını istatistik konusunda temel bilgilere sahip okurlara yönelik olarak anlatacağım.

R programını çalıştırıldığında komutlarınızı almak için bekleyen sıkıcı görünüşlü bir komut konsoluyla karşılaşırsınız:

```
R version 2.10.1 (2009-12-14)
Copyright (C) 2009 The R Foundation for Statistical Computing
...
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
```

Bu programla analiz etmek için veriye ihtiyacımız var. Her ne kadar yazılımın içinde eğitim amaçlı hazır veri setleri olsa da biz bir tane oluşturalım. Örneğimizdeki veri setinde bir dersi alan öğrencilerin cinsiyet, yaş, ve not bilgileri var:

```
cinsiyet yas not
       E  21  56
       E  18  27
       K  19  26
       E  19  48
       E  17  35
       K  22  71
       K  21  78
       E  20  59
       K  21  62
       E  20  85
       E  19  34
       K  17  32
       E  18  56
```

Bu veri setini bir veri tablosu yazılımı ile oluşturun, daha sonra CSV formatında dışa aktarın, mesela "notlar.csv" isimli bir dosyaya. Şimdi R konsolunda bu veri setini yükleyelim:
`> verisetim= read.csv("notlar.csv")`
. Bu komutla verisetini dosyadan okutup 'verisetim' isimli bir değişkende sakladık. R'ın 'summary' (İng. özet) komutuyla bunun gibi veriler hakkında kapsamlı bir özet görebiliriz:

```
> summary(verisetim)
 cinsiyet      yas             not
 E:8      Min.   :17.00   Min.   :26.00
 K:5      1st Qu.:18.00   1st Qu.:34.00
          Median :19.00   Median :56.00
          Mean   :19.38   Mean   :51.46
          3rd Qu.:21.00   3rd Qu.:62.00
          Max.   :22.00   Max.   :85.00
> mean(verisetim$not)
[1] 51.46154
```

Buradaki özet verisetindeki üç bilgi sütününun özetlerini veriyor. İlk sütün kategorik bir bilgi (cinsiyet) içerdiğinden sadece kategori sayımları var, diğer sütunlarda skalar bilgi olduğundan

minimum/maximum değerler, ortalama, ve ortanca bilgileri özette yeralmış.

Böyle bir verisetini öncelikle veriyi anlamak amacıyla grafikleştirip inceleyebiliriz. Aşağıdaki komutlar sırayla öğrenci notlarının histogramını, ve not-cinsiyet grafiğini gösterecektir (şekil 6.5).

```
> hist(verisetim$not)
> plot(verisetim$yas,verisetim$not)
```

Burada verisetim ismiyle kaydettiğimiz verisetinin farklı sütunlarına verisetim$not gibi bir sözdizimiyle atıfta bulunabiliyoruz. Eğer isterseniz 'hist' ve 'plot' komutlarıyla ilgili kullanım bilgisini "help(hist)" diyerek görebilir ve bu komutlara ek parametreler vererek grafikleri ihtiyacınıza uygun şekilde ayarlayabilirsiniz (örneğin etiketleri, çizim türü, yazı büyüklüğü, vb.).

Verileri görselleştirme yararlı ve gerekli olsa da veriler arasındaki ilişkiyle ilgili iddialarda bulunmak için yararı sınırlıdır. Bu verisetinde öğrenci başarısının cinsiyet ve/veya yaş ile açıklanıp açıklanamayacağını ancak modelleme yaparak söyleyebiliriz. Burada sadece doğrusal modellerden bahsedeceğim. Varolan yüzlerce R bileşeni içinde farklı modelleme araçları da vardır.

Doğrusal modelleme için R'da 'lm' (İng. linear model kısaltması) komutunu kullanacağız. Oluşturduğum modeli de bir değişkende saklayacağım ki sonradan inceleyebileyim. Önce not başarısı ve cinsiyeti modellemeyi deneyelim:

```
> modelim= lm(verisetim$not   verisetim$cinsiyet)
```

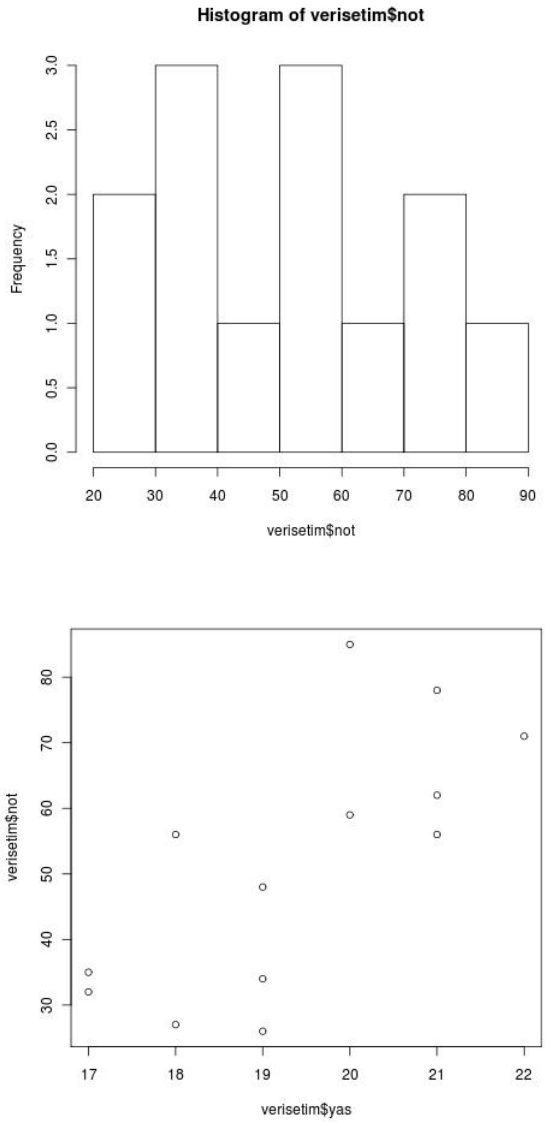
. R'da birçok veri için olduğu gibi doğrusal model nesnesi için de 'summary' komutunu kullanabiliriz:

```
> summary(modelim)

Call:
lm(formula = verisetim$not ~ verisetim$cinsiyet)

Residuals:
   Min     1Q Median     3Q    Max
 -27.8  -16.0    6.0    9.0   35.0

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)         50.000     7.221   6.924 2.51e-05 ***
```



Şekil 6.5: R programıyla çizilmiş bir histogram ve grafik

```
verisetim$cinsiyetK    3.800     11.644   0.326     0.75
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 20.42 on 11 degrees of freedom
Multiple R-squared: 0.00959,Adjusted R-squared: -0.08045
F-statistic: 0.1065 on 1 and 11 DF,  p-value: 0.7503
```

Buradaki çıktıda t-test ölçütü modelin başarısız olduğunu gösteriyor. Yani not düzeyi ile cinsiyet arasında bir ilişki yok. Olsaydı modeldeki cinsiyet değişkeninin yanında bir veya daha fazla * görecektik. Doğal olarak R^2 değeri, yani modelin veriyi açıklama gücü de çok düşük.

Bir de not başarısının yaşa bağlı olup olmadığına bakalım:

```
> modelim = lm(verisetim$not ~ verisetim$yas)
> summary(modelim)

Call:
lm(formula = verisetim$not ~ verisetim$yas)

Residuals:
      Min        1Q    Median        3Q       Max
-22.06188  -9.74010  -0.06188   4.61634  28.09901

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   -119.881     49.390  -2.427  0.03357 *
verisetim$yas    8.839      2.540   3.480  0.00515 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 14.16 on 11 degrees of freedom
Multiple R-squared: 0.524,Adjusted R-squared: 0.4808
F-statistic: 12.11 on 1 and 11 DF,  p-value: 0.005147
>
```

Bu çıktıdaki t-test ölçütü modelin başarılı olduğunu gösteriyor. R^2 değeri de modelin açıklayıcı gücünü 0.48 olarak veriyor. İsterseniz her iki bağımsız değişkeni de içeren bir model kurabilirsiniz:

```
> modelim= lm(verisetim$not    verisetim$cinsiyet) +
verisetim$yas)
```

, ancak bu koşullar altında genişletilmiş model yeni bir şey söylememekle kalmayacak veriyi açıklama gücü de düşecektir.

R'ın becerilerine işaret etmek açısından bu veriyle son bir işlem yapalım. verisetini örneklemdeki kadın ve erkekle için ayrıştıralım, ve not ortalamalarına bakalım. R'daki 'split' komutu bir sütunu kategorik bir kolona karşı ayrıştırarak yeni bir veri seti oluşturur:
`> ayrilmis= split(verisetim`*not*, *verisetim*`cinsiyet)`
. 'ayrilmis' değişkeninde sakladığımız bu ayrıştırma cinsiyet kolonundaki her bir kategorik değer için yeni değişkenimizde bir kolon oluşturur ve orijinal verisetinde ayrıştırılacak sütunun uyan değerlerini o kolona koyar. Yani 'ayrilmis$E' ve 'ayrilmis$K'. Şimdi cinsiyetler için ayrı ortalamalar bulabiliriz:

```
> mean(ayrilmis$E)
[1] 50
> mean(ayrilmis$K)
[1] 53.8
```

R'ın istatistik becerileri üzerine yazılmış birçok kitap olduğu gibi R öğrenmek için gerekli temel dökümanları projenin web sitesinden de bulabilirsiniz. Burada bahsettiklerimiz dışında R ile kümeleme analizi, stokastik modelleme, değişkenlik analizi (anova) gibi birçok istatistik işlemi yapmak mümkündür.

6.7 Kullanacağımız yazılımları nasıl seçmeli?

Bu sorunun cevabı yazılımın kullanım amacının bireysel veya kurumsal oluşuna, hatta ne kadar maceracı olduğunuza göre değişir. Önemli bir kriter yazılımın hatasız olması. Benim kişisel gözlemim açık kaynak (kamu lisanslı) yazılımların bu konuda başarılı olduğu. Açık kaynak yazılımlar ücretsiz olmakla beraber ne önemli özelliği bu değil. Kamus lisanslarının esası yazılımın kaynak kodu herkesin görüşüne açık olmasıdır (bkz. `http://www.belgeler.org/howto/gpl.html`). Bu sayede birçok göz tarafından incelenirler, hatalar daha hızlı bulunur ve giderilir. Ancak ticari yazılımları toptan bir kenara atmak doğru olmaz. Doğru geliştirme pratiklerine sahip birçok firma çok iyi kalitede yazılımlar üretiyor.

Önemli bir başka kriter yazılımın kullanışlı olması. Açık kaynak yazılımların ticari yazılımlar kadar kullanışlı olmadığı söylense de bu genelleme doğru değildir. Çok fazla sayıda olan açık kaynak yazılımlar içerisinde kullanışlılık açısından başarılı olanlar

gibi berbat olanlar da var. Burada açık kaynak ve ticari ürünler arasındaki esas fark, ticari ürünleri yapan firmaların yazılımın kullanımını gereğinden fazla basite indirgeyerek ürünün cazibesini arttırma eğilimidir. Bu ilk bakışta kullanıcıya cazip gelse de esasen yazılımın orta-uzun vadede kullanışlılığını sınırlar. Açık kaynak yazılımlarda ilk bakışta kafa karıştırıcı görünen seçenek kalabalığı aynı zamanda sizin kullanım özgürlüğünüzün sınırlarının geniş olmasıyla ilgilidir.

Bir işi yapmak için seçtiğiniz yazılımdan sonrada vazgeçebilirsiniz. İster kişisel ister kurumsal ihtiyaçlar olsun böyle bir değişikliğin muazzam sıkıntılara yolaçma potansiyeli vardır. Bunu önlemenin yolu seçtiğiniz yazılımların aynı kategorideki diğer yazılımlarla uyumlu olmasıdır. Kişisel bir örnek vermem gerekirse ben OpenOffice'i Microsoft Word'e tercih ediyorum. Bence ikisi de kullanışlılık ve özellik olarak çok benziyor. Ancak OpenOffice pek çok alternatif formatta metin dosyasını içe ve dışa aktarma imkanı sağlıyor. Bu sayede bu dökümanları değiştirmek, alıp-vermek için başka yazılımlar kullanma seçeneğinde sahibim. Oysa Microsoft Word'de bu imkan daha sınırlı. Eğer benim gibi sık sık yeni yazılımları denemeye hevesli biriyseniz bu tür özellikleri gözönüne almakta fayda var. Ayrıca bu konu kurumsal süreklilikle de ilgilidir. Yazılım firmaları kapanabilir, ürünleri artık varolmayabilir. Bu tür yazılımlardan alternatif formatlara dışa aktarım yapamıyorsanız çok çaresiz bir durumda kalırsınız. Geçmiş yıllarda bazı AB kurumları bu sebepten dolayı eskiden beri kullandıkları Word formatı yerine OpenOffice'in kamusal formatını kullanmayı yeğlediler. Öte yandan örneğin Autodesk firmasının Maya isimli bilgisayar animasyon programı birçok formatla veri değiştokuşu yapabiliyor. Böyle bir yazılımı satın alıp kullanmak çok daha az risk taşıyor.

Benzer bir uyumluluk sorunu da farklı işletim sistemleri arasındaki uyumlulukla ilgili. Bir belge üzerinde (metin, veri tablosu, herneyse) bir ekiple beraber çalışıyorsanız ekipteki insanların farklı işletim sistemleri (Mac, Linux, Windows) kullanması çok olasıdır. Böyle bir durumu gözönüne alarak, kullanacağınız yazılımları seçerken olabildiğince farklı sistemleri desteklemesinde dikkat etmelisiniz.

Benzer bir uyumluluk sorunu da tek başına çalışıyorsanız da vardır. Teknolojinin hızla değiştiği günümüzde bilgisayarımızı birkaç yılda bir değiştiririz. Farklı bir donanım denemek istersek

(örneğin PC yerine Mac) herşeyin toptan değişmesi gerekebilir. Özellikle Linux bu konuda bir avantaj sunuyor çünkü 32 veya 64-bit PC mimarisinin yanısıra onlarca farklı donanım mimarisi üzerinde çalışabiliyor. Eğer işletim sisteminiz yeni bilgisayarınızla uyumluysa diğer yazılımlar da aynen çalışmaya devam edecektir.

Son olarak kararsız kaldığınızda diğer bilgisayar kullanıcılarının tecrübelerinden yararlanabilirsiniz. Bazı sektör dergilerinin ve sitelerin (örneğin `PCmag.com`, `cnet.com`) yanısıra programcı sosyal ağları da (örneğin `stackoverflow.com`) bol miktarda yazılım değerlendirmesi ve yorumları içeriyor.

Bölüm 7

Bilgisayarı Yönetmek: Programlama Sanatı

Bilgisayar programı bir hesaplama yönteminin belirsizliğe yer bırakmayacak biçimde, açık ve net adımlar halinde ifade edilmiş halidir. Çok sayıda programlama dili var, ve bir hesaplama yöntemi bu dillerin herbirinde biraz farklı bir ifade bulur. Ancak her zaman esas olan yöntemin kendisidir.

Bir programdaki adımların herbiri çok temel bir işlemden ibarettir. Örneğin programlarımızda "şu ve şu sayıyı çarp", veya "şu sayı diğerinden büyükse şu adıma geç" gibi ifadeler kullanabiliriz. Ancak prensip olarak daha karmaşık, örneğin "şu sayının karekökünü bul", "a sayısının n'inci kuvvetini bul",ya da "şu sayıları sıraya diz" gibi bir komut veremeyiz. Bilgisayar bu tür şeyleri anlayabilseydi zaten matematik kitabını açar okur ve problemleri kendisi çözüverirdi. Program yazarken yapılacak bir hesabı neredeyse eğitimi yetersiz ama çalışkan ve ezberi kuvvetli bir çırağa anlatır gibi ifade etmemiz gerekecektir.

İş sadece bundan ibaret olsaydı programcılık çok zor bir iş olmazdı. Ama yukarıdaki örneklerdeki basit görünen hesapları bile yapmanın birden fazla yolu vardır; ve bu alternatif yollardan bazıları diğerlerinden daha iyidir. Örneğin 5^8 sayısını hesaplamak istediğimizi düşünelim. Bunu 5'i yedi defa kendisiyle çarparak yapabiliriz: 5x5x5x5x5x5x5x5. Ama daha kısa bir yolu var: önce

5'i bir defa kendisiyle çarpıp 5^2'yi bul, sonra bu sayıyı bir defa kendisiyle çarpıp 5^4'ü bul, sonra onu da kendisiyle çarp ve 5^8'i bul. Böylece işlem yedi yerine üç çarpma ile biter. Sözkönusu işlem 5^{2048} gibi daha büyük bir hesap olsaydı iki yöntem arasındaki fark daha da açılacaktı. Bunun uç bir örnek olduğunu düşünebilirsiniz, ama güvenli web sitelerine giriş yaptığınızda kullanılan şifreleme yöntemleri bol bol bu tür hesaplar gerektiriyor ve aradaki fark bilgisayarın çalışma hızını önemli ölçüde değiştiriyor.

Programcılığı asıl ilginç kılan, ve bir matematik branşı olarak kabul edilmesini sağlayan şey bir hesabı yapmanın alternatif yöntemlerinin birbiriyle karşılaştırılması için geliştirdiği tekniklerdir. Ne var ki biz buradaki kısa bahsimizde bu tekniklere değinmeyeceğiz, sadece belirlenmiş bir yöntemin programa dökülmesinde kullanılan temel tekniklerden ve programların çalışmasından bahsedeceğiz. Profesyonel programcılık bu ve daha ileri teknikleri, bir problemi en iyi şekilde çözmek için yaratıcı bir şekilde harmanlamayı gerektirir. Yine de burada bahsedebildiğimiz kadarının bu programların ne olduğu ve nasıl yazıldığına dair temel bir fikir edinmenize yeteceğini umuyorum.

7.1 Bilgisayarın işleyişindeki belirsizlik ve kesinlik: Süreçler, programlar, ve girdiler

Bir bilgisayarın beyninde olup biten şeye hesaplama süreci, ya da kısaca *süreç* (İng. process) adını vereceğiz. Süreç bir *program*ın işe koşulmuş halidir. Aynı programı defalarca işe koşarız, ancak her seferinde yaşanan süreç farklıdır. Yazdığımız program süreci yönlendirir: hangi işin nasıl yapılacağını, hangi durumlarda hangi yöntemlerin tercih edileceğini program belirler. Ancak süreci tek yönlendiren program değildir. Bir insanın her gün aynı işe gitmesi ama her seferinde farklı meselelerle uğraşması gibi aynı programdan işe koşulan süreçler de dış dünyanın önlerine çıkardığı duruma göre farklı yollar izler. Bizi, yani kullanıcıyı da içeren dış dünyanın sürecin önüne koyduğu ve her seferinde farklı olan bu meselelere *girdi* diyeceğiz. Süreç içerisinde bu meselelere üretilen çözümlere de *çıktı* diyoruz.

Bölüm 2.2.2'de bariyer kontrolü problemini çözmeye çalışırken çözümümüzü basit bir programa benzeterek ifade ettik. Bu örnekte girdiler bariyerin önünde ve arkasında araç bulunup bulunmadığını gösteren dedektörlerin verisi ve bariyerin halihazırdaki durumuydu, ve çıktısı da bariyerin açılıp akapanması kararıydı.

Bu programı işe koyup bir süreç başlatabilir, ve sürecin gün içerisinde vereceği açma,kapatma kararlarını kaydedebiliriz. Her gün aynı programı çalıştırsak ta bu kayıtlar birbirinden farklı olacaktır.

Her işe koşmada farklı şeyler yaşansa da bu örnekteki gibi bir sürece *belirlenmiş* (İng. deterministic) süreç diyoruz, çünkü süreci yönlendiren program çok kesindir. Program ne yapılacağına sadece verilere bakarak karar veriyor, mesela zar atarak değil. Kimi zaman belirsizlik de arzu edilebilir. Örneğin tavla uyunu programı yazıyorsanız tam da bunu istersiniz. Böyle bir programın, yani davranışı sadece girdilere göre olmayan, rasgele seçimlere de bağlı olan bir programın işe koşulması ise *belirsiz* (İng. non-deterministic) bir süreçle sonuçlanır.

Rasgelelik bilgisayarın doğasına son derece terstir, ve ancak taklit edilebilir. Tavla programı gibi görece daha basit bir uygulamada bu durumu hissetmeyiz. Ancak örneğin askeri haberleşme veya bankacılık sistemi gibi uygulamalarda kullanılan şifrelemenin gerçekleştirilmesinde ciddi bir sorun olur. Şifrelemeyi hep aynı şekilde yaparsanız –ki bilgisayar sistematik çalışan bir makine olarak şifrelemede kullanılacak rasgele parolaları uydururken de hep aynı yolu izler– eninde sonunda birileri şifreyi kıracaktır, çünkü bahsedilen uygulamalarda hasım çok ve heveslidir. Rasgelelik elektronik bilgisayarın doğasına çok ters olduğundan gelecekte bu tür işlerin doğası itibarıyle olasılıksal olana kuantum bilgisayarlarla yapılması gibi çözümler üzerinde duruluyor.

7.2 Programlama dilleri

Bilgisayarın anladığı dil tektir. 1 ve 0lardan oluşan bu dile makine kodu adını vermiştik (bkz. bölüm 2.3). Teknolojinin emekleme aşamasında olduğu ilk yıllarda programcıların gerçekten de kendilerini bu dilde ifade etmeleri gerekiyordu, ve bu çok sıkıntılı bir iştir. Bu ilk yıllarda bilgisayar bilimcileri bir yandan başka meslekler in nükleer fizik ve astronomi hesapları, savunma sistemlerinde balistik hesaplar gibi problemlerini çözerken bir yandan da kendi işlerini daha iyi yürütmek için bilgisayar programları yazdılar.

Bir bilgisayar programı bir kerede oluşturulup sonra öylece kullanılan bir şey değildir. Bir yandan programın kullanıcıları sürekli olarak programa yeni özellikler kazandırılmasını ister. Bunun için programın tekrar gözden geçirilip belirli yerlerine varolan parçalarla uyumlu eklemeler yapılması gerekecektir. Öte

yandan kullanım sırasında bazı parçaların hatalı veya düşük performansta çalıştığı tespit edilirse bunların düzeltilmesi ve iyileştirilmesi gerekecektir. Üstelik bütün bu işleri programı ilk yaratan ve neresinin nasıl çalıştığını bilen programcı yapacak diye bir koşul da yoktur. Kimi zaman oldukça kalabalık bir programcı ekibi tarafından yürütülen bir iştir bu. Bütün bu sebeplerden ötürübilgisayar programı sadece makinenin anlaması için yapılmaz. Aksine birçok farklı kişinin programı okuması, anlaması ve iyileştirebilmesi esas bir meseledir. Ayrıca yapılacak iyileştirme ve eklemeler sırasında programcıların makul bir sürede işlerini bitirebilmesi, bunun için de programa dökmeye çalıştıkları yöntemi doğallıkla ifade edebilmesi gerekir.

Teknolojinin ilk yıllarından bu yana geliştirilen programlama dilleri bu sorunları çözmeyi amaçlıyordu. Makine kodunun 1 ve 0 kalabalığı yerine bizim yazılı dilimize yakın bir dil kullanılacak, programcılar ellerindeki hesap yöntemlerini bu dilde ifade edecek, kendilerinin ve başkalarının yazdığı programları kolayca okuyup anlayabilecek, düzeltebilecek, öte yandan da bu dilde yazılan program makinenin diline otomatik olarak çevrilebilir olacaktı. Dolayısıyla bir programlama dili ile kasdedilen hem dilin gramer ve yazım kuralları ve sözdağarcığı, hem de o dilde yazılanları makine diline çeviren ve derleyici denilen programdır (İng. compiler), ki sadece bu derleyicinin makine dilinde yazılması gerekecektir. Bu da altından kalkılabilir bir sıkıntıdır.

İlk gerçekleştirilen programlama dili oldukça basittir. Assembler (birleştirici) adı verilen bu dil makinenin ana işlemcisinin (CPU) anladığı komutların 1 ve 0'larla ifade edilen komut numarası yerine komutun yaptığı işi ifade eden ve hatırlaması kolay bir sözcük veya kısaltma kullanılmasına imkan tanıyordu. Bu kadarı bile ilk programcılar için muazzam bir rahatlık sağlamıştı. Üstelik önemli bir avantaj daha sağlıyordu: yeni bir makine yapıldığında, yani yeni ve komut seti eskisinden faklı bir CPU kullanıldığında bütün programları yeniden yazmak gerekmiyordu. Sadece derleyiciyi duruma uygun şekilde değiştirmek yeterli oluyordu. Daha da iyisi yazılan tek bir programı farklı tipte makinelerde çalıştırabilmek için sadece derleyicinin o makinelere uygun çeşitlerini yaratmak yeterli oluyordu.

Ancak bilgisayar programlarıyla yapılmaya çalışılan işlerin çeşidi ve bunların karmaşıklığı kısa sürede artacaktı. Bu ihtiyaç sonucunda sunduğu bütün bu kolaylıklara rağmen Assembler'dan

daha öte arayışlar ve programlama dillerinde çeşitlenme ortaya çıkmıştır. Yeni programlama dilleri tasarlanırken matematik yöntemlerin olabildiğince doğallıkla programa dökülmesi ön plandaydı. Mesela matematikte bir fonksiyonu bir kez tanımlanıp sonra sembolik ismi ile kullanabilirsiniz. Bunun programlama dillerinde de bir karşılığı olmalıydı.

Günümüze kadar geçen sürede yüzlerce programlama dili ortaya çıkmıştır, ve önemli bir kısmı da halen kullanımdadır. Bu durum bir zenginlik gibi görünse de bir yanıyla şaşırtıcıdır. Bilgisayar programları yapılırken aynı bir sanayi makinesindeki gibi bazı hazır parçalar kullanılarak yapılır. Bir motoru veya dişliyi birçok çeşit makinenin yapımında kullanabilirsiniz. Bir programcı da elindeki problemi çözmek için başka programcıların yaptığı parçaları kullanır, bunları bağlayarak, değişiklik ve ekleme yaparak, ve yeni parçalar da üreterek bir program oluşturur. Oysa bir programlama dilinde yazılmış program parçaları, başka bir dilde program yazanlar tarafından kullanılamaz. Bu yüzden varolan programlama dili bolluğu bir yanıyla bu meslek camiası için bir sıkıntı kaynağıdır. Ayrıca işe yeni başlayan gençler için de bir sıkıntıdır. Programlama dili bu mesleğin icrasında temel araçtır, ve bir programlama dilinin kullanımında ustalaşmak, belirli bir zihin kıvraklığına erişmek zaman ve emek ister. Pek çok meslek erbabı bir programlama dilinde ustalaşır ve farklı dilleri kullanmakta sıkıntı çeker. Bu yüzden gençler başlangıçta hangi programlama dilini kullanacakları konusunda ciddi bir panik yaşıyorlar. Herşey bir yana belirli bir dili seçmek demek başka dilleri kullanmalarını gerektiren iş fırsatlarını değerlendirememek demek. Bu yüzden de kullanacakları araca karar verirken haklı olarak piyasa trendlerine bakarak karar veriyorlar. Aynı şekilde bu konuda eğitim veren üniversite veya kurslar da eğitimi kolaylaştıracak olan dil ve teknolojiyi değil piyasa trendleri ve iş imkanları açısından cazip olanı seçmek gibi bir açmazın içindedirler. Biz eğitimcilerin kendi geçmiş ve becerilerie bağlı kalma eğilimimizden doğan tercihler de cabası.

Öte yandan programlama dillerinin çeşitliliği gerekli ve kaçınılmazdır. Konuya teorik olarak baktığımızda bütün programlama dilleri eşdeğerdir. Birinde yazılıp ta diğerinde yazılamayacak bir program yoktur. Ancak bir programlama dili belirli türden işleri yapmak için diğer dillerden daha avantajlı bir gramer ve sözdağarcığı sunabilir. Örneğin bugün halen kullanılan en eski

dil olan FORTRAN dili özellikle matematik problemleri için tasarlanmıştır ve matris işlemlerini kolayca yapmaya uygundur. Bu dil halen MATLAB ve R gibi genel amaçlı matematik ve istatistik yazılımlarında kullanılıyor. Halen kullanılan ikinci en eski dil olan LISP ise yapay zeka çalışmalarında kullanılmak için tasarlanmıştı. Bu dil çalışma hızı yerine kullanma kolaylığını hedefliyordu (örneğin otomatik bellek temizliği yapıyordu). Halen de yapay zeka problemlerinde kullanılmaktadır. Bir başka örnek ise PHP isimli dil. 1990'ların sonunda hızla artan web uygulamalarında kullanılmak için özel olarak tasarlanmış. Amatörce yapılmış tasarımı birçok bilgisayar bilimciyi dehşete düşürecek zaaflar içerse de bu dil web programcıları arasında çok tutulmuş ve halen de çok yaygındır.

Programlama eğitimi veren üniversiteler özellikle son yıllarda öğrencilere bu teknoloji ve dil çeşitliliği içerisinde mesleği icra etmelerini kolaylaştıracak yaklaşımlar üzerinde çalışıyorlar. Biz burada özellikle MIT'de (Massachussets Institute of Technology) olumlu sonuçlar alınmış olan bir yaklaşımı (Felleisen, Findler, Flatt, and Krishnamurthi, Felleisen et al.; Abelson and Sussman, 1996) benimseyeceğiz.

7.3 Klasik bir programlama dili: Scheme

Bu bölümde temel bazı programlama tekniklerine yüzeysel de olsa bir bakacağız, ve birkaç basit problemi bu tekniklerle çözeceğiz. Bunun için çoğu okurun muhtemelen hiç denemediği bir şeyi yapması gerekecek bir programlama dilinin temel gramerini ve birkaç sözcüğünü öğrenmek, ayrıca programlarımızı çalıştırmak için gerekli programı edinip kurmak. Bazı okurların ihtimal ki varlığından ve popülerliğinden haberdar oldukları programlama dilleri var: Java, PHP, veya Python gibi. Ancak tercihimiz başlangıç düzeyi eğitim için grameri basit ve denenmiş bir dilden yana: Scheme (okunuşu 'skim').

Scheme dili en eski dillerden biri olan LISP dilinin eğitim amacıyla basitleştirilmiş bir çeşidi. Bu dil ile pratik yapmak için kullanabileceğimiz güzel bir yazılım da mevcuk: DrRacket. Bu yazılımı `http://racket-lang.org/` web sitesinden ücretsiz olarak indirip kurabilirsiniz. Bilgisayarınıza uygun indirmeyi seçmelisiniz, ve DrRacket'in Windows, Linux, ve Mac üzerinde çalışan çeşitleri mevcuttur. İngilizce bilen ve programlama konusunu bizim bahsedeceğimizden daha derinlemesine merak eden okurlar

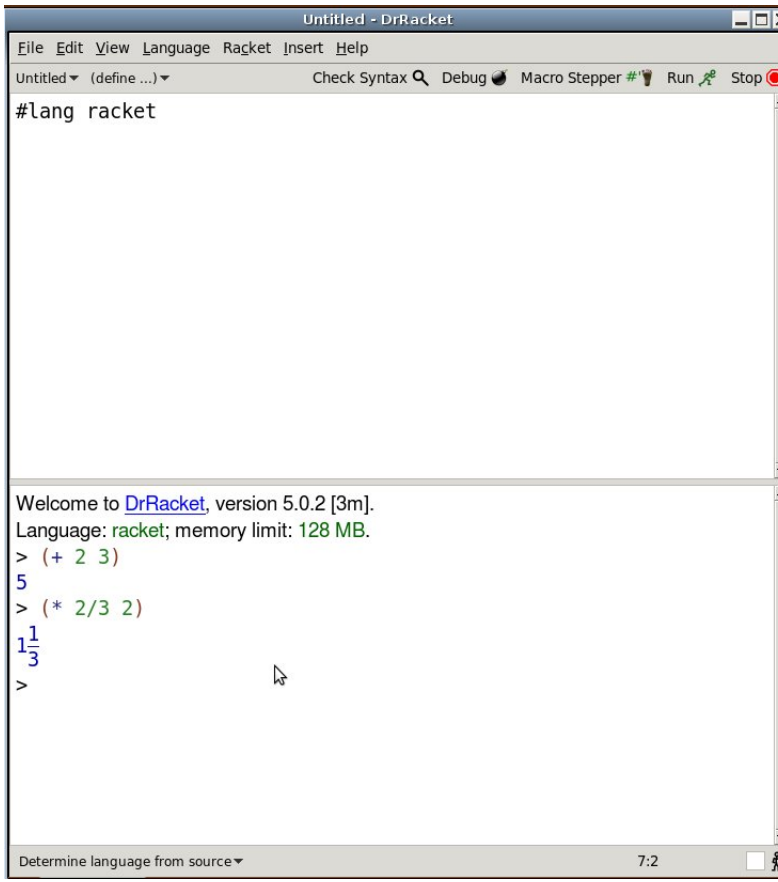
ayrıca bu web sitesindeki eğitim malzemelerini, ve özellikle de ücretsiz bir elektronik kopyasını `http://www.htdp.org/` adresinden edinebilecekleri "How to Design Programs" (Programlar Nasıl Tasarlanır) kitabını inceleyebilirler.

DrRacket programını indirip kurduktan sonra çalıştırdığınızda şekil 7.1'teki program penceresini göreceksiniz. Program penceresinin üst trafındaki büyük alan programlarımızı yazmak için. Ancak önce Scheme dilinin gramerine ve temel sözcüklerine alışmak için biraz egzersiz yapacağız. Bu yüzden önce program penceresinin alt kısmındaki komut bölümünü kullanacağız. DrRacket bu bölüme yazdığımız kısa komutlara hemen tepki verir. Türkçede olduğu gibi Scheme programlama dilinde de anlatım cümlelere bölünmüştür. Bu dildeki her cümle bir parantez içine alınır. Cümle yapısı ise son derece basittir: ilk sözcük ya da işaret bir fiile karşılık gelir, sonra gelene bütün ifadeler ise eylemin konusu olan nesnelerdir. Özne yok, çünkü bütün eylemlerin uygulayıcısı bilgisayar. Sıfat veya zamir de yok. Scheme dilini konuşan DrRacket programının anladığı fiillerden bir kısmı basit aritmetik işlemlerle ilgilidir. Bu fiillerden toplama ve çıkartma bildik + ve - işaretleriyle, bölme ve çarpma ise / ve * işaretleriyle ifade edilir. Örneğin komut bölümüne (+ 2 3) cümlesini yazıp ENTER tuşuna basın. Bu cümleyi verdiğimizde DrRacket bize sonucu yazar ve sonra '>' sembolünü tekrar ekranda göstererek yeni bir komut almaya hazır olduğunu belli eder:

```
> (+ 2 3)
4
>
```

yazdığımız bu cümlede toplama (+) fiilinin konusu olan iki nesne, yani 2 ve 3 birer sayı idi. Programlarda kimi zaman nesnesi harf veya semboller olan fiiller/komutlar da olabilir, ancak biz burada kendimizi sadece nesnesi sayılar olan fiillerle sınırlayacağız. Scheme dilinde 2 ve 3 gibi tamsayıların yanısıra -3 gibi eksi sayıları, 3.14 gibi gerçek sayıları, ve 2/3 gibi kesirli sayıları da kullanabiliyoruz. Örneğin (* 2/3 2) cümlesini veya (+ 1.25 2.25 1.5) cümlesini deneyin.

Nasıl ki Türkçe konuşurken kurduğumuz cümleler konuştuğumuz kişi için bir anlam ifade ediyorsa, Scheme programlama dilinde kurduğumuz cümlelerinde bilgisayar için bir anlamı var. Oldukça sınırlı olan bu anlam cümledeki fiilin uygulanması sonucu



Şekil 7.1: DrRacket program ekranı

ortaya çıkan değerdir, yani bir sayıdır. Başka bir deyişle her cümle bir takım girdileri verilen fiile göre işleyip bir çıktı üreten minik bir programcıktır.

Program yazarken kullanılan temel bir teknik bir cümlenin çıktısının başka bir cümle için girdi olarak kullanılmasıdır. Bu sayede bizim kağıt üzerinde 1+2x3 diye yazdığımız iki eylemden (önce bir çarpma, sonra bir toplama) oluşan işlemi Scheme dilinde (+ 1 (* 2 3)) diye yazarız. Bilgisayar böyle bir cümle içinde cümle durumuyla karşılaştığı zaman, yani bir iç-cümle bir dış-cümlenin girdisi olarak kullanıldığı zaman önce iç-cümleyi işleyip sonucu bulacak, sonra bulduğu değeri dış cümlede yerine koyarak işlemi tamamlayacaktır. Dolayısıyla önce (* 2 3) işlemi yapılıp sonucu, yani 6 sayısı bulunacak, sonra bunu yerine koyunca ortaya çıkan (+ 1 6) işlemi yapılacaktır. Egzersiz olarak günlük dilde yazılmış şu ifadeleri DrRacket'a hesaplatmayı deneyin: 3.14x5x5 ve 1+2x(3+4).

7.3.1 Tanımlamalar

Aritmektik işlemler gibi Scheme'in doğuştan yapmayı bildiği birçok eylem var. Ancak bunlar arasında özel yeri olan bir eylem var: İngilizce olarak kullanılan 'define' yani 'tanımla' eylemi. Bu eylem diğerlerinden farklı çünkü bilgisayar için anlamı bir sayı değil. bu tanımlamanın sonucunda bilgisayar yeni bir şey öğreniyor. Bilgisayara iki tür şey öğretebiliyoruz. Bunlardan biri nesnelerin adı. Örneğin bilgisayara pi sayısını şu tanımlama ile öğretebiliriz:

```
(define pi 3.14)
```

Bu ve diğer tanımlamalarımızı DrRacket program penceresinin program kısmına, yani üst tarafına yazacağız. Çünkü tanımlamalar yapmaya başladığımızda artık gerçekten program yazıyoruz demektir. Yukarıdaki tanımlamayı yerine yazın, daha sonra da 'run' (çalıştır, koştur) düğmesine tıklayarak programı çalıştırın. Görünüşte hiçbirşey olmayacak! Çünkü bu programdaki tek komut bilgisayara birşey öğretiyor, ama öğrettiği şeyi kullanmıyor. Artık bilgisayarımız 'pi' sözcüğünün anlamını bildiğine göre bunu aritmetik işlemlerde kullanabiliriz. Şimdi programdaki tanımın altına (* pi 5 5) yazın ve programı yeniden çalıştırın ('run' düğmesine tıklayın), böylece yarıçapı 5 olan bir dairenin alanını, yani $\pi \cdot r^2 = \pi \cdot r \cdot r$ öğrenmiş olacağız.

Bu yaptığımız tanımlamanın faydası çok sınırlı görünebilir,

çünkü (* 3.14 5 5) yazsak ta pek farklı olmayacaktı. Ama önemli bir faydası vardır. Eğer pi sayısı tanımımızı daha hassas yaparsak, mesela (define pi 3.141521) şiklinde geliştirirsek, programımızda pi sözcüğünü kullandığımız her komut otomatik olarak bu daha hassas değeri kullanacaktır. Aksi takdirde birsürü komutta 3.14 yazdıysak hepsini bulup teker teker daha hassas pi değeriyle değiştirmemiz gerekecekti.

Tanımlama işleminin bilgisayara pi gibi sayıları ezberletmek dışında, ikinci ve daha ilginç bir kullanımı var. Bilgisayara yeni fiiller öğretmek, yani yeni bir işi yapma becerisini kazandırmak. Örneğin yukarıda açık bir aritmetik ifade ile elde ettiğimiz dairenin alanı hesaplamasını bilgisayara öğretelim. Bilgisayara bir fiili öğretmek demek fiilin nesnelerinin ne olduğunu ve bu nesneleri hangi işleme sokarak bir sonuç elde edeceğini söylemek demektir. Öğreteceğimiz eylemin adına 'daire-alanı' diyelim. Bu eylemin tek bir girdisi var: dairenin yarıçapı. Yani ismi 'daire-alan' olan ve tek bir nesnesi (yarıçap) olan bir fiil tanımlıyoruz. Bu fiil bir kere tanımlanınca bilgisayara uygulatmak için `(daire-alan 5)` gibi bir ifade kullanacağız. Ama iş öğretmeye geldiğinde daire yarıçapını bir sembolle ifade ederiz:

```
(define (daire-alan yarıçap)
     (* pi yarıçap yarıçap))
```

Dikkat ederseniz burada da aynı pi sayısı tanımladığımızdaki gibi {mycodeoneline(tanımla neyi-tanımlıyorum değeri-nedir) şeklinde bir ifade kullandık. Yani tanımlama işleminin birinci nesnesi tanımlanan şey1, ikincisi ise o şeyin değeri. Pi sayısı tanımındaki gibi bir isim ezberletme yaparken 'neyi-tanımlıyrorum' kısmına sadece tanımlanan şeyin adını yazmıştık. Oysa tanımlanan şey 'daire-alanı' gibi bir eylem olduğunda 'neyi-tanımlıyorum' kısmına bir eylem ifadesi yazdık: (daire-alanı yarıçap). 'Değeri-nedir' kısmına isa pi tanımındaki gibi bir sabit sayı değil de, eylemin nesnesi olan 'yarıçap' parametresine bağlı bir hesaplama koyduk.

Bu tür tanımlamalara işlev diyeceğiz. Aslına bakılırsa Scheme'in de içinde bulunduğu bazı diller 'işlevsel' programlama dili olarak tabir ediliyor çünkü tamamen bu tür tanımlamalar üzerine kuruluyorlar. İşlev'in en temel özelliği birtakım girdiler alması (örneğin yarıçap) ve birtakım çıktılar üretmesidir (örneğin dairenin alanı).

Artık bilgisayarımız daire-alanı işlevini bildiğine göre progra-

mın kalan kısımlarında bu işlevi, örneğin (daire-alanı 5) diyerek çalıştırabiliriz. Programımızın tamamı şu hale geldi:

```
#lang racket
(define pi 3.141521)
(define (daire-alanı yarıçap)
     (* pi yarıçap yarıçap))
;örnek kullanım
(daire-alanı 5)
```

Programın başında zaten ekranda bulunan "#lang racket" ifadesi var; bunu yerinde bırakıyoruz ki DrRacket hangi dili konuştuğumuzu anlasın (çünkü bizim değinmeyeceğimi başka dilleri de konuşabiliyor). Programdaki ";örnek kullanım" satırı da kendimiz için yazdığımız bir açıklama. DrRacket ';' ile başlayan satırları gözardı ediyor. Racket program penceresinin son hali resim 7.2'de verilmiştir.

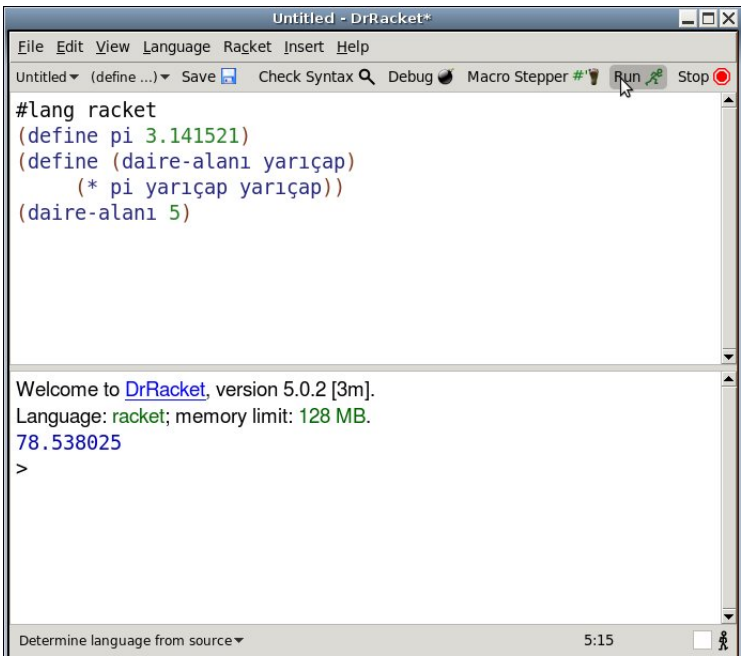
Bu yöntemle istediğimiz eylemi tanımlayabiliriz. Örneğin silindirin hacmini hesaplamak için de bir eylem tanımlayalım. Bu eylemin iki girdisi olacak: silindirin taban yarıçapı (r) ve yüksekliği (h). Hacmi ise $\frac{1}{3}\pi r^2 h$ olarak hesaplayacağız. O halde bu tanımlamayı içeren programımız:

```
#lang racket
(define  pi 3.141521)
(define (koni-hacmi yarıçap yükseklik)
     (/ (* pi yarıçap yarıçap yükseklik) 3))
;örnek kullanım
(koni-hacmi 1 10)
(koni-hacmi 5 7)
```

Yaptığımız programlarla bilgisayara birşeyler öğrettik. Ancak DrRacket programını kapatıp yarın yeniden açtığınızda hepsini unutmuş olacak. Bu yüzden programınızı bir dosya olarak kaydetmelisiniz (File->Save menüsünden). DrRacket'i tekrar açtığınızda aynı dosyayı açıp çalıştırarak bilgisayarı yeniden eğitmiş olursunuz.

7.3.2 Koşullu işlemler

İlk programımızdaki dairenin alanını bulma işlemi yarıçap ne olursa olsun aynı aritmetik işlemle hesaplanıyor. Bu kez biraz



Şekil 7.2: DrRacket program penceresinin bir program girilmiş hali

farklı bir hesaplamayı ele alacağız: bir öğrencinin test sınavından aldığı net puanın hesaplanması. Bu testte iki yanlış cevap bir doğru cevabı götürüyor. O yüzden doğru cevap sayısını d, yanlış cevap sayısını y ile temsil edersek, net puan $d - \frac{y}{2}$ olacak. Ancak bir koşul daha var: öğrenciler eksi puan almayacak. Yani $d < \frac{y}{2}$ ise net puan 0 olacak. Böyle bir hesaplamayı matematikte koşullu bir ifadeyle yazarız:

$$\text{puan} = \begin{cases} 0 & \text{eğer } d < \frac{y}{2} \text{ ise} \\ d - \frac{y}{2} & \text{aksi takdirde} \end{cases}$$

Böyle koşullu bir işlemi Scheme programında ifade etmek için 'if' komutunu kullanacağız ki bu sözcük İngilizce 'eğer' anlamına geliyor. Bir if komutu üç girdi alır:

```
(if koşul doğrusya-değer yanlışsa-değer)
```

Böyle bir komutta koşul girdisi daha önce gördüğümüz sayı girdilerden biraz farklıdır. Burada iki sayının karşılaştırmasının sonucunu koşul olarak kullanacağız. Net-puan hesaplama problemimizde bu koşul $d < \frac{y}{2}$ karşılaştırmasıdır. Böyle bir koşulu Scheme dilinde (< d (/ yanlış 2)) şeklinde ifade edeceğiz. Böylece kurduğumuz cümledeki küçüktür (<) işareti küçük-müdür sorusunu cevaplayan, soru kipinde bir cümle kurmak için kullanılıyor, ki bu tür bir komutun sonucu aritmetik işlemlerden farklı olarak 'doğru' ya da 'yanlış' değerini alır. Örneğin DrRacket komut penceresinde (< 2 3) yazarsanız size cevabı kısaca #t, yani true/doğru olarak verecektir. Buna benzer şekilde >,=, <= gibi semboller Scheme dilinde sırasıyla 'büyük-müdür', 'eşit-midir', 'küçük veya eşit-midir' soruları için kullanılabilir.

Bunları kullanarak öğrencilerin net-puanını hesaplama işlevini aşağıdaki gibi tanımlayabiliriz:

```
1 #lang racket
2 (define (net-puan doğru yanlış)
3   (if (< doğru (/ yanlış 2)) 0
4       (- doğru (/ yanlış 2))))
5 ;örnekler
6 (net-puan 9 1)
7 (net-puan 2 8)
```

Bu programı çalıştırdığınızda iki örnek için çıktılar sırasıyla 8.5 ve 0 olarak hesaplanacaktır. Eğer isterseniz programı bir kere

çalıştırıp DrRacket'i eğittikten sonra komut penceresinde başka örnekler de deneyebilirsiniz.

Program yazarken yukarıdaki örnekten daha karışık koşullu işlemlerle sık sık karşılaşırız. Başka bir örnek olarak bir bankanın kredilere uyguladığı faiz miktarının hesaplanışını ele alalım. Diyelim ki sözkonusu banka üç değişik faiz oranı uyguluyor: 100.000 TL'den küçük borçlar için aylık 1.8%, 100.000-1.000.000 TL arası için 1.7%, ve 1.000.000'dan daha büyük borçlar için 1.6%. Bu hesaplamada borç<100.000 koşulu sağlanıyorsa uygulanacak faiz bellidir: 1.8%. Ancak aksi takdirde ikinci bir koşula daha bakmamız gerekir, yani borç<1.000.000 koşuluna. Dolayısıyla 'if' komutunun aksi-takdirde kısmı da bir koşullu işlem olacaktır. Böyle bir durumu ifade etmek için aynı aritmetik işlemlerde yaptığımız gibi içiçe cümleler kurmamız gerekir:

```
#lang racket
(define (faiz miktar)
  (if (< miktar 100000) 1.8
      (if (< miktar 1000000) 1.7
          1.6)))
;Testler
(faiz 9000)   ;faiz 1.8 olmalı
(faiz 115000) ;faiz 1.7 olmalı
(faiz 1575000);faiz 1.6 olmalı
```

Bu programda dıştaki 'if' komutunun sonucu ya 1.8 ya da içteki 'if' komutudur. İkincisi olduğunda içteki komut çalıştırılır ve onun sonucu tüm işlemin sonucunu belirler.

7.4 Sayıları öğütmek: böl ve fethet tekniği

Bilgisayar programlarından beklentimiz genellikle dairenin alanı hesaplamasından daha karmaşıktır. Çoğu zaman yapılacak işlem birden çok adımı gerektiri ve kaç adımda yapılacağına ancak problemin girdileri belli olunca karar verilebilir.

Bir örnek olarak banka borcunun aylar geçtikçe üzerine faiz binerek ne kadar artacağını bulalım. Bankadan 1.8% aylık faiz ile 70.000 TL borç alındıysa, beşinci ayın sonunda borç ne kadar olur bunu oturup hesaplayabiliriz. Her ayın sonunda borç kendisi artı faiz olacak, yani 100%+1.8%=101.8%'e katlanacaktır. Dolayısıyla beş ayda beş defa 1.018'e katına çıkacaktır. Böylece beş ayın sonunda toplam borç: $70.000x\frac{101.8}{100}^5 = 76531$ TL olur.

Ama bizim burada hedefimiz istenilen ana-para miktarı ve süre verildiğinde borcu hesaplayacak bir program yazmak. Ayrıca bu program bankanın faiz dilimlerindeki değişimi de hesaba katmalı, çünkü banka 100.000'den küçük, 100.000-1.000.000 arası, ve 1.000.000'dan büyük meblağlar için farklı faiz oranları uyguluyor. Yani borç 100.000 TL'yi aştıktan sonra faiz oranı değişecek.

Bu problemde birkaç şeyin birden gözönüne alınması gerekiyor. Bir aylık faiz hesabını yapabiliyoruz, zaten biraz önce bu hesabı yapacak 'faiz' programını yazdık. Ama yeni problemimizde hesabın kaç aylık süreyi kapsayacağını bilmiyoruz. Diyelim ki b miktarda bir borcun a ay sonra ne kadar olacağını hesaplayacağız. Eğer a'nın değeri 1 ise işimiz kolay: yukarıda yazdığımız 'faiz' işlevini bir kez çağırıp bu borç miktarına uygulanan aylık faiz oranını buluruz, sonra bir ayın sonundaki borcu $b((100 + faiz(b))/100)$ olarak hesaplarız. Ya da Scheme ifadesi olarak yazarsak:

```
(* b (/ (+ 100 (faiz b)) 100))
```

Ancak a'nın birden büyük değerleri için önce bir önceki aya kadar biriken borç miktarını bulup sonra bir aylık faiz daha ekleyerek sonucu bulmamız gerekir. Eğer bu yaptığımız açıklamayı *borç* adlı bir işlevi/fonksiyonu matematiksel olarak tanımlamakta kullansaydık şöyle yazardık:

$$bor(b,a) = \begin{cases} b\frac{(1+faiz(b))}{100} & \text{eğer } a = 1 \text{ ise} \\ bor(b,a-1)\frac{(1+faiz(bor(b,a-1)))}{100} & \\ & \text{aksi takdirde} \end{cases}$$

Bu ifadede ilginç bir durum var: *borç()* fonksiyonu tanımın konusu, yani ifadenin sol tarafında, ancak ayrıca ifadenin sağ tarafında da, yani kendi tanımının bir parçası olarak kullanılıyor. Bu duruma özyinelemeli tanım diyoruz, çünkü tanım kendisini yineleyerek yapılıyor. Programlarda özyineleme tekniği uzun bir hesabı daha küçük parçalara bölerek tanımlamanın temel yollarından biridir. Daha küçük, çünkü yukarıdaki *borç()* fonksiyonu her özyinelemede ay sayısı bir eksilterek yineleniyor. Dolayısıyla bu yinelemeler zincirinin sonunda ay sayısı 1'e inecek ve tanım yineleme içermeyen ilk koşulu kullanarak sonlanacaktır. Bu şekilde sonlanmasaydı, yani sonsuz bir yineleme olsaydı böyle bir özyinelemeli tanım tamamen işe yaramaz olurdu.

Şimdi bu özyinelemeli tanımı programa dökebiliriz:

```
#lang racket
(define (faiz miktar)
  (if (< miktar 100000) 1.8
      (if (< miktar 1000000) 1.7
          1.6)))
(define (borç b a)
  (if (= a 1) (* b (/ (+ 100 (faiz b)) 100))
      (* (borç b (- a 1)) (/ (+ 100 (faiz (borç b (- a 1)))) 100))))
;örnekler
(borç 90000 12)
```

Eğer yineleme bir noktada sonlanmazsa özyineleme tekniğinin bütün gücü bir tehlikeye dönüşüverir. Yukarıda yaptığımız işlev tanımı da böyle bir risk içeriyor: eğer kullanıcı yanıp yakılır da ay sayısı olarak sıfır veya eksi bir sayı girerse özyineleme sonsuza kadar devam eder. Çünkü bu programda he özyineleme ay sayısının bir eksiği ile yapılıyor ve yinelemeler bu sayı 1'e dayandığında son buluyor. Oysa ay sayısının başlangıç değeri zaten 1'den küçük olursa onu eksilterek 1'e ulaşamayız, dolayısıyla bu programı `(borç 10000 -2)` diye çalıştırırsanız başınız dertte demektir. Program bir süre bilgisayarınızı yorar ve sonra bir hata mesajı verir.

Böyle bir durumda ne yapabiliriz? Faizin zamanda geriye işletilmesi hiç mantıklı değil, o yüzden negatif bir ay değeri için yapacak hiç bir makul hesap yok. Bu durum hatalı bir durum. Bundan dolayı işlev tanımımızı da şöyle değiştirmeliyiz:

$$bor(b,a) = \begin{cases} hata & \text{eğer } a < 1 \text{ ise} \\ b\frac{(1+faiz(b))}{100} & \text{eğer } a = 1 \text{ ise} \\ bor(b,a-1)\frac{(1+faiz(bor(b,a-1)))}{100} & \\ & \text{aksi takdirde} \end{cases}$$

Bu yeni tanımlamayı programımıza yansıtmak için yeni bir Scheme ifadesine ihtiyacımız olacak. Bunun için de özel bir komut olan 'error' (İng. hata) komutunu kullanacağız. Bu komutun sonucu bir değer değil, programın ne yapacağını bilmediği bir noktaya geldiğini gösteriyor:

```
#lang racket
(define (faiz miktar)
  (if (< miktar 100000) 1.8
      (if (< miktar 1000000) 1.7
          1.6)))
(define (borç b a)
  (if (< a 1) (error "Hatalı girdi")
      (if (= a 1) (* b (/ (+ 100 (faiz b)) 100))
          (* (borç b (- a 1)) (/ (+ 100 (faiz (borç b (- a 1)))) 100)
;örnekler
(borç 100000 1)
(borç 100000 -1)
```

Artık programımız olası her durumda ne yapacağını biliyor.

Özyineleme tekniğini pekiştirmek için başka örnekler yapmakta yarar var. Bunu yaparken küçük bazı yenilikler de göreceğiz. İşte örneklerimiz:

Fibonacci'nin tavşanları Bu kez eski bir problemi ele alalım: Pisa'lı Leonardo, ya a diğer adıyla Fibonacci'nin tavşanları. Avrupa'nın ilk önemli matemetikçilerinden olan Fibonacci (13.yy), bir çift tavşanın üremesiyle bir yıl sonunda çiftlikte kaç çift tavşan olacağını hesaplamaya çalışıyordu. Problemi biraz basitleştirmesi gerekti; şöyle ki:

- İlk başta bir çift yavru tavşan var.
- Yeni yavrular bir ay sonra çiftleşecek olgunluğa erişiyorlar.
- Tavşanlar çiftleştikten bir ay sonra dişi tavşan iki tane (bir erkek bir dişi) yavru doğuruyor.
- Tavşanlar hiç ölmüyor.

Bu kuralların hepsini beraber düşünürsek henhangi bir aydaki tavşan çifti sayısı geçen ayki tavşan çifti sayısına yeni doğanları ekleyerek bulunur. Geçen ay doğan genç tavşanlar hariç, yani bir evvelki ay varolan çiftlerin hepsi doğuracağı için toplam sayı:
`toplam sayı=(geçen ayın nüfusu + evvelki ayın nüfusu)`
olarak bulunabilir. Sadece ilk iki ay için tavşan çifti sayısı 1'dir, çünkü başlangıçtaki tek yavru çiftin büyüyüp, çiftleşip doğurması

için toplam iki ay geçecek. Bu durumda a ay sonunda tavşan sayısını matematiksel olarak yazmamız gerekirse:

$$tavanlar(a) = \begin{cases} \text{hata} & \text{eğer } a \leq 0 \text{ ise} \\ 1 & \text{eğer } 0 < a \leq 1 \text{ ise} \\ \text{tavşanlar(a-1)+tavşanlar(a-2)} & \\ & \text{aksi takdirde} \end{cases}$$

Bu tanımlamayı gerçekleştiren program da şu şekilde olacaktır:

```
#lang racket
(define (tavşanlar ay)
  (if (<= ay 0) (error "Hatalı girdi")
      (if (<= ay 2) 1
          (+ (tavşanlar (- ay 1)) (tavşanlar (- ay 2))))))
;örnekler
(tavşanlar 12)
(tavşanlar 24)
```

Programı çalıştırdığımızda Leonardo'nun aradığı sayıyı, yani bir yılın sonundaki tavşan çifti sayısını 144 olarak hesaplıyor. İkinci yılın sonunda ise 46 bine çıkıyor. Gerçekte işlerin bu kadar iyi gideceğinden şüpheliyim. Ama Leonardo'nun problemi kendisinin soruyu çözmek için geliştirdiği sistematik ve algoritmik yaklaşımdan dolayı hatırlanıyor.

Mortgage hesabı Bu kez ev kredisi (mortgage) için ödeme takvimi çıkartacağız. Bu problemde bir yandan borca sabit bir faiz işliyor, bir yandan ise aylık ödemeler yapılıyor. Aylık ödeme miktarına o diyelim. Hesabımız biraz değişecek. Bu yeni problem için borç işlevini tanımlarsak: f aylık faizle alınan b miktarda ev kredisi, ayda o miktar ödeme yapılıyorsa, a ay sonra:

$$mortgage(b, a, o) = \begin{cases} hata & \text{eğer } a < 1 \text{ ise} \\ b\frac{(1+f)}{100} - o & \text{eğer } a = 1 \text{ ise} \\ mortgage(b, a-1, o)\frac{(1+f)}{100} - o & \\ & \text{aksi takdirde} \end{cases}$$

Bu kez amacımız belli bir süre sonunda borç miktarını bulmak değil, borç bitene kadar olan takvimi çıkartmak. Tabii bunun olabilmesi için borcun küçülüyor olması lazım. Dolaysıyla aylık

ödeme miktarı borcu küçültmeye yetmiyorsa bunu da rapor etmeliyiz.

Bu işi yapmak için hem yukarıdaki mortgage işlevini, hem de borç takvimini gösterecek ikinci bir işlevi tanımlamamız gerekecek, ki bu işlevin adına borç-takvimi diyelim. Bu bizi ilginç bir durumla karşı karşıya getiriyor çünkü borç-takvimi işlevi bir çıktı üretmeyecek. Bütün amacı takvimi ekrana yazdırmak. Yani bu kez sonuçla değil hesaplama sırasında ekrana yazdırılanlarla, bir başka deyişle programın yan etkileriyle ilgileniyoruz.

Borç takvimi işlevini yazmak için yine özyineleme tekniğini, ama bu kez biraz değişik şekilde kullanacağız. Herhangi bir ayın borç durumunu ekrana yazdırmadan önce o aya kadar olan borç durumlarını yazdırmış olmamız gerekecek. Ekrana bilgi yazdırmak için de 'printf' isimli yeni bir komut kullanacağız. Programımız şöyle:

```
#lang racket
(define (mortgage borç faiz ödeme ay)
  (if (< ay 1) (error "hatalı girdi")
      (if (= ay 1) (- (* borç (/ (+ 100 faiz) 100)) ödeme)
          (- (* (mortgage borç faiz ödeme (- ay 1))
                (/ (+ 100 faiz) 100)) ödeme))))
(define (takvim-çıkar borç faiz ödeme)
  (if (> (mortgage borç faiz ödeme 1) borç)
      (error "bu borç bitmez")
      (borç-takvimi borç faiz ödeme 1)))
(define (borç-takvimi borç faiz ödeme ay)
  (printf "ay ~a borç ~a~n" ay (mortgage borç faiz ödeme ay))
  (if (> (mortgage borç faiz ödeme ay) 0)
      (borç-takvimi borç faiz ödeme (+ ay 1))
      ay))
;örnekler
(mortgage 100000 1.8  2000 12)
(takvim-çıkar 100000 1.8 2000)
```

Bu programda borç takvimi yazdırmak takvim-çıkar işlevini çağırarak yapılıyor. Bu işlev önce borcun bu ödemerle bitirilebilir olup olmadığına bakıyor ve gerekirse bir hata mesajı veriyor. Aksi takdirde borç-takvimi işlevini birinci aydan başlayacak şekilde tetikliyor. Borç takvimi işlevinde yeni bir durum görüyoruz. Daha önce programlarımızda yazdığımız işlevlerde tek bir ifade vardı ve onun sonucu işlevin de sonucu oluyordu. Oysa borç-takvimi

işlevinde iki ifade var: 'printf' ve 'if' ifadeleri. Birincisi sadece ekrana bir sonuç yazdırıyor, bir değer üretmiyor. Bu yüzden programımız hala net bir anlama sahip: borç-takvimi işlevi borcu yazdırmak gibi bir yan etkinin yanısıra çıktı değeri olarak borcun kaç ayda biteceğini veriyor (en son satır).

7.5 Sayılardan sıkıldıysan: Çizim egzersizleri ve fraktal desenler

Özyineleme doğada da sık sık görülen bir süreçtir. Bir ağacın ya da brokoli sebzesinin dallarını gözünüzün önüne getirin. Küçük dallar boyutları dışında büyük dallara benzer, çatallanma biçimleri aynıdır. Ağaç dalların oluşumuyla ilgili biyokimyasal süreç - örneğin bir insanın gelişiminden farklı olarak- oldukça basittir, küçük ve büyük dallar için benzer şekilde işler. Bun tür bir öz-benzeşim kar tanelerinin oluşumunda da vardır.

Bu sebplerden dolayı özyineleme kullanan programlarla bir kalemi yönlendirebilirsek bu türden öz-benzeşimli doğal nesnelerin şekillerini çizebiliriz. Sadece doğal şeylerle sınırlı kalmayan bu özbenzeşimli şekillere fraktal diyoruz. Fraktaller iki veya üç boyutlu, renkli olabilir, ancak biz burada kendimiz iki boyutlu ve tek renkli olanlarla sınırlayacağız.

Bu bölümde yapacağımız çizimler için DrRacket'ın sağladığı bir çizim paketinden yararlanacağız. Bu çizim paketini aktif hale getirmek ve sağladığı komutları kullanabilmek için programımızın başına

`(require graphics/turtle)`

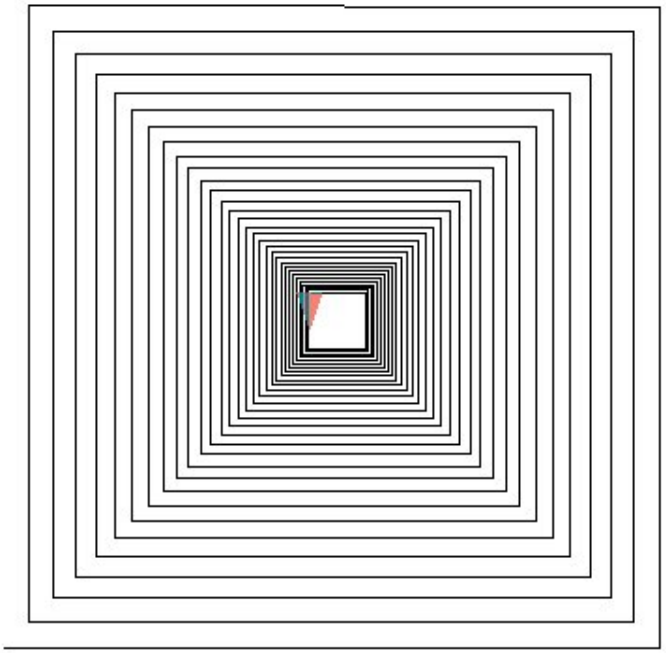
satırını ekleyeceğiz. Bunu yaptıktan sonra istediğimiz zaman

`(turtle)`

komutunu vererek çizim penceresi açıyoruz. Bunu yapınca ekranda bir kalem ucu gözükecektir. Bundan sonra vereceğimiz komutlar bu kalemin çizgi çizmesine, sağ/sol dönüşler yapmasına yarar. Örneğin:

```
(draw 50)
(turn 90)
(draw 100)
```

gibi. Burada 'draw'/çizim komutuna çizilecek çizginin uzunluğunu girdi olarak veriyoruz. 'turn'/dön komutuna ise dönüş açısını.



Şekil 7.3: Kare spiral fraktali

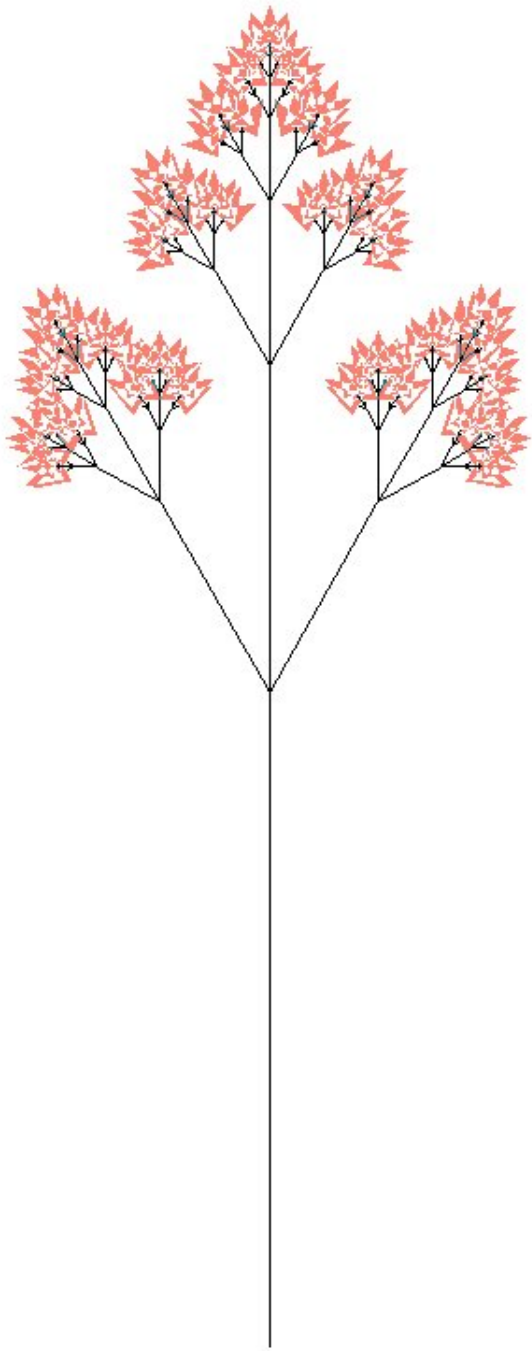
İlk olarak basit bir özyinelemeli çizim ele alalım. Resim 7.3'te gördüğünüz kare spiral tek bir çizim ilkesine dayanıyor: bir çizgi çiz, sağa dön, daha kısa bir çizgi çiz, ve böylece devam et. Tabii her özyinelememizde olduğu gibi yinelemenin nerede sonlanacağını belirlememiz gerekir. Çizgiler kısalarak belirli bir boya, eşik bir değere indiğinde özyinelemeyi sonlandırarak bunu yapabiliriz. Böylece yapacağımız tanıma çizgi uzunluğu ve eşik değeri girdi olarak vermemiz gerekiyor. Ayrıca yinelemeler arasında çizginin ne oranda kısalacağını da bir girdi ile kontrol edeceğiz. Bu durumda programımız şu şekilde:

```
#lang racket
(require graphics/turtles)
(define (kare-spiral uzunluk küçülme-oranı eşik-uzunluk)
         (draw uzunluk)
         (turn 90)
  (if (> uzunluk eşik-uzunluk)
         (kare-spiral (* uzunluk küçülme-oranı) küçülme-oranı eşik-uz
         empty
         ))
;çalıştır
(turtles)
(kare-spiral 350 0.98 30)
```

Bu programın anlaşılır kalması için bazı hata durumlarını kontrol etmeksizin bıraktım (örneğin uzunluk veya küçülme-oranı'nın eksi olması gibi). Bu program bir çıktı üretmiyor, sadece bir yan etkisi var: ekrana istediğimiz şekli çizmek. Programdaki tanımlamaları kullanarak kare-spiral fraktalinin değişik varyantlarını çizdirebilirsiniz. Örneğin:
`(kare-spiral 200 0.8 10)`
veya
`(kare-spiral 500 0.98 10)`
gibi.

Ek bir egzersiz olarak dönüş açısını değiştirmeyi deneyebilirsiniz. Dönüşlerin 90 derece yerine 120 derece olması durumunda bir üçgen spiral, 60 derece için altıgen-spiral, vb., elde edersiniz.

Şimdi biraz daha ilginç bir örnek ele alalım. Bu kez resim 7.4'deki gibi bir ağaç çizeceğiz. Burada ağacı bir özyineleme olarak düşünüyoruz: bir dal biraz uzayıp üç tane yeni dala ayrılıyor. Bu kare-spiral örneğimizden farklı, çünkü ayrılan dallar kalemi kaldırmadan çizilemez. Bunun yerine bilgisayarımızı biraz yoracağız, çünkü ona istediğimiz kadar kalemi aynı anda kullandırtabiliriz. Ağacın dallarının çatallanmasını bu şekilde, 'split' (İng. ayrıl) komutu ile yapacağız. Programımız şu şekilde:



Şekil 7.4: Fraktal bir ağaç

```
#lang racket
(require graphics/turtles)
(define (dal uzunluk açı eşik-uzunluk)
  (turn açı)
  (draw uzunluk)
  (if (> uzunluk eşik-uzunluk)
      (split*
       (dal (/ uzunluk 3) 30 eşik-uzunluk)
       (dal (/ uzunluk 2) 0 eşik-uzunluk)
       (dal (/ uzunluk 3) -30 eşik-uzunluk))
      empty))
;örnek
(turtles)
(turn 90)
(dal 150 0 5)
```

Bu programda bütün ağaç yukarı doru bir dal olarak başlıyor. Dal 30'ar derece sola ve sağa kendisinin üçte biri uzunlukta iki dal, ve bir tane de yukarı kendisinin yarısı kadar ana dal olmak üzere üç dala ayrılıyor. Dalların herbiri de eşik uzunluktan büyük oldukları sürece çatallanmaya devam ediyorlar.

7.6 Programcılığa heveslenenler için bir ufuk turu

Yaptığımız örneklerde programcıların kullandığı temel tekniklerden bir kısmını uyguladık. Temel tekniklerin sayısı sanıldığı kadar çok değildir. Aynı bir ressamın büyük ölçüde fırça ve yağlıboya gibi basit malzemelerle çalışması gibi programcı da bu temel tekniklerde edindiği kıvraklıkla işini yapar. Ancak aynı boyayı karıştırmanın ve renk elde etmenin, veya form ve gölgeleme gibi teknikleri kalem ve fırçayla uygulamanın pratik gerektirmesi gibi programlama teknikleri de pratik ister.

Yine de programcılığın asıl zorluğu bu temel tekniklerin pratiğinden öte bir sorun. Aynı resimdeki gibi burada da yaratıcılık gerekiyor. Bu özellikle gerçek hayata dair herşeyin bilgisayar ortamında da karşılığı olan günümüzde çok geçerli. Yaptığımız örneklerde sadece basit sayılarla çalıştık. Fakat gerçek hayattaki nesne kalabalıklarının temsili bundan çok daha karmaşık veri çeşitleri kullanmayı gerektirir. Facebook gibi bir sitenin çalışması, milyonlarca insanın bilgilerini tutması, ve herbirine saniyeler içinde özelleşmiş kişisel sayfalarını gösterebilmesi oldukça kar-

maşık veri yapılarıyla son derece yüksek performanslı süreçlerin birarada kullanılmasını gerektirir. Bilgisayar programcılığı eğitiminde temel veri yapısı çeşitlerini öğretiyoruz, ancak hemen her gerçek problem yeni bir veri yapısı doğaçlamasına, ve bu verilerden binlerce, hatta milyarlarcasını hızla işleyecek programlara ihtiyaç duyar.

Yine de bunlar kesinlikle bir gözğı değil. Eğer özyineleme tekniğini sevdiyseniz bütün bunlarla uğraşmaktan keyif alabilirsiniz demektir. Programcılık eğitiminde edinilen tekniklerle bizim örneklerimizden çok daha karmaşık işler yaratmak gerekiyor. Ama iş büyük ve detay çok olsa bile tekniklerin uygulamasındaki keyif aynıdır. Bu bölümde bu tekniklerin yüzeysel bir özetini yapacağız. Bunu yaparken de daha çok genç bir disiplin sayılabilecek programcılığın eğitiminde farklı yaklaşımlar olduğunu vurgulamaya çalışacağız. Yazık ki bu yaklaşımların çoğu ne her duruma ne de herkese uygun. Buna rağmen eğitim sırasında kaçınılmaz ve mutlak yöntemler olarak sunuluyorlar. İster bir eğitim kurumunda ister kendi kendinize çalışarak öğrenmek niyetinde olun, bu yaklaşım farklılıklarını bilmek eğitim malzemesi seçiminizi kolaylaştırabilir. Ayrıca farklı yaklaşımları harmanlama ve kendi tarzınızı oluşturma özgürlüğünüzün de farkında olmalı ve onu kullanmalısınız. Öte yandan aşağıdaki bahislerden bazıları -ister kendinizin- ister başkalarının olsun- iyi programla kötü programı ayırdetmenin bazı somut kriterlerini de sunuyor. Bu kriterler herşeye rağmen programcılığın bir bilimsel disiplin olarak gelişmesinde yol gösterici olmuştur, ve profesyonel uygulamalarda da temel kriter olarak görülmelidir.

7.6.1 Alternatif programlama teknikleri: trendler ve gerçekler

Yaptığımız programlar matematiksel bir işlev (fonksiyon) tanımının Scheme dilinde birebir ifadesiydi. Bunların çoğu gerek matematiksel gerek grafik özyineleme kullanıyordu. Bu programlardaki işlev tanımlarında verilen girdi değerleri yerine konulur, özyineleme mekanizması ifadeleri yeniden ve yeniden, sonlanma koşulu gerçekleşene kadar bu girdilerle açılır, ve sonuçta ortaya çıkan (bizim yazmadığımız ama özyineleme ile oluşmasına sebep olduğumuz) uzun aritmetik ifadede girdiler yerine konularak hesap tamamlanır. Özyinelemeyi matematiksel bir ifadeyi birebir yansıtarak oluşturmak bir yana programcı bilgisayarın hafızasını nasıl kullanacağını kontrol etmeye kalkışmaz. Onun yerine bilgisayar

tanımlamaların ve özyinelemelerin gerektirdiği biçimde hafızayı kullanır.

Yaptığımız program örenekleri bu konuda biraz geçmişi olanlar için ihtimal ki şaşırtıcı olmuştur; hem programlama dili seçimi hem de programlama tarzı açısından. İşlevsel programlama denilen bu programlama tarzı daha önceki bölümlerde bahsettiğimiz Church'ün matematiksel yalınlığa dayalı yaklaşımıyla örtüşüyor. Oysa ki hem mesleki pratikte hem de programcılık eğitiminde daha yaygın olan bir başka programlama tarzı var. Ne yazık ki kimi zaman uygun olmayan problemlere uygulanan, ve başlangıç eğitimini de zorlaştıran bu tarz makinenin hafızasını birebir kontrol etmeye ve deyim yerindeyse makineyi ite kaka hesap yapmaya dayanıyor. Emirsel programlama denilen bu tarz Turing'in makine metaforuyla örtüşür. Ayrıca, kanım o ki, mühendislik pratiğinin detaycı ve kontrolcü doğasıyla da uyumlu olduğu için bu kadar yayıgınlaşmıştır. Ne var ki kimi problemleri programa dökmeye uygun olsa da bu emirsel programlama tarzı programların yalın ve anlaşılır olmasını engelliyor. Turing'in tasarımı hesaplamayı makinelere uygun kılmayı amaçlıyordu, insanlara değil.

Bu iki tarzın farkını örneklemek için örnek olarak faktöriyel hesabını ele alalım. Doğal sayılar için tanımlanan faktöriyel 1'den o sayıya kadar olan sayıların çarpımıdır. Mesela

$$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$$

gibi. Verilen bir sayının faktöriyelini hesaplayacak bir program yazacağız, ve bunun için aynı banka borcu örneğindeki gibi verilen girdiye göre değişen sayıda çarpma işlemi yapmak gerekiyor. Faktöriyeli kağıt üstünde yazarken iki farklı yaklaşım kullanabiliriz:

$$n! = \begin{cases} 1 & \text{eğer } n = 1 \text{ ise} \\ n \cdot (n-1) \cdot (n-2) \ldots 2 \cdot 1 & \text{aksi takdirde} \end{cases}$$

veya

$$n! = \begin{cases} 1 & \text{eğer } n = 1 \text{ ise} \\ n \cdot (n-1)! & \text{aksi takdirde} \end{cases}$$

Burada hata durumunu gözardı ettik. Faktöriyelin birinci yazım şekli birçok okura daha tanıdık gelecektir. Bu tanım yapılacak hesabı, yani birsürü sayının çarpılacağını daha açık ediyor ve bu yüzden lise derslerinde gördüğümüz matematiksel ifade tarzına

daha yakın. Öte yandan ikinci tanım yalın bir özyineleme kullanıyor, ancak alışkın olmayanlar için ilk bakışta anlaşılması daha zor görünebiliyor.

Biz banka borcu hesaplama probleminde yukarıdaki ikici ifadeye benzer özyineleme kullanan bir matematiksel tanımdan yola çıkmıştık ve bu yüzden yazdığımız program bu tanımın birebir Scheme'de ifadesinden ibaretti. Burada da ikinci tanımı kullanırsak aynı şeyi yapabiliriz:

```
1  #lang racket
2  (define (faktoriyel n)
3    (if (= n 1) 1
4        (*  n (faktoriyel (- n 1)))))
5  ;örnekler
6  (faktoriyel 5)
```

Oysa birinci faktöriyel tanımını kullanmaya kalkarsak ortaya farklı bir durum çıkar. Bu ikinci yöntemi elimizle hesaplasaydık ara sonuçları bir kağıda yazıp, çarpma işlemlerini yaptıkça karalayıp yeni değerleri yazarak ilerleyecektik. Şimdi kağıdın kenarı yerine bilgisayarın hafızasını karalayıp duracağız. Hafızada içine yazıp/kullanıp/değiştirdiğimiz bu yerlere değişken diyoruz. Adı üstünde, bunların nasıl değişeceğini programımızda açıkça belirteceğiz. Bir değişken kullanmak için önce onu daha önce pi sayısı için yaptığımız gibi tanımlarız ve başlangıç değerini atarız. Daha sonra is 'set!' komutuyla değerini değiştirebiliriz. Faktöriyel hesabını emirsel tarzda yapabilmek için bir yandan şimdiye kadar hangi sayıları çarptığımızı, bir yandan da çarpımın sonucunu hatırlamamız lazım. Programımız aşağıdaki gibidir:

```
1  #lang racket
2  (define (faktoriyel n)
3    (do
4        ([nerdeyim 1] [çarpım 1])
5        ((> nerdeyim n) çarpım)
6        (set! çarpım (* çarpım nerdeyim))
7        (set! nerdeyim (+ nerdeyim 1)))
8    )
9  ;örnekler
10 (faktoriyel 5)
```

Bu ikinci programda birden n'e kadar sayıları çarpmaya başlama-

dan önce 'nerdeyim' ve 'çarpım' değişkenleri başlangıç değerleri ile tanımlanıyor. 'do' (İng. yap) komutu karmaşık bir *döngü* kurma komutudur. Komutun ilk parçası değişkenleri başlangıç değerlerine ayarlamak, ikincisi döngünün sona erme koşulu (nerdeyim n'den büyük olursa) ve sona erdiğinde üreteceği çıktı değeri (çarpım'da biriken değer), sonrakiler ise her dönüşte yapılacak işlemlerdir (çarpımı nerdeyimle çarp, nerdeyim'i arttır). Programın bu hali özyinelemeli olana göre daha karmaşık ve anlaşılmaz görünüyor. Bu yaklaşım bambaşka komutlar gerektiriyor.

Bu örnek problemdeki dezavantajlarına rağmen kimi zaman makine metaforuna dayanan, işlevsel/özyineleme yerine emirsel/döngülerle yazılan programlar gerekli olur. Bu durum özellikle sistemin derinliklerinde çalışan ve gerçekten de görüntü kartı, ağ kartı, disk sürücüsü gibi bir makine parçasını kontrol etmeye yönelik programlarda ortaya çıkar. Çünkü bu durumlarda gerçekleştirilecek işlem bir matematik tanımlamadan yola çıkan hesabı değil bir makinenin ve onunla ilgili hafıza noktalarının kontrolüdür. Ancak genel olarak yalınlık esas olmalı, makine metaforu ve değişkenler başta ne kadar cazip görünürse görünsün temkinli kullanılmalıdır. Öte yandan yalın bir matematiksel tanım aramaktansa bilgisayarı ite kaka hesabın altından kalkmak genelde daha cazip görünüyor gibi. Kitapçıların rafları ve İnternet siteleri bu tuzağa düşen eğitim malzemeleriyle dolu. Makine metaforu sektör çapında bir histeriye dönüşmüş ve daha etkili yöntemleri gölgelemiş görünüyor. Tam da bu yüzden benimseyeceğiniz programlama tekniği üzerine iyi düşünmek ve trendlere aykırı da görünse sağlam temelli ve etkili yöntemleri kullanmak programcılıkta ilerlemek açısından önemli bir unsur olacaktır.

7.6.2 Kolay hesap, zor hesap, doğru hesap: Bilgisayar bilimlerinin temel sorunsalı

Bir problemi çözmek için farklı tarzda programlar yazılabilir. Bunların bazıları diğerlerinden daha 'güzel', daha anlaşılır olabilir. Ancak çözümün kalitesi ile ilgili önemli bazı kriterler vardır: performans ve doğruluk. Google, facebook, sahibinden.com gibi popüler web siteleri binlerce veriyi işleyip saniyeler içerisinde kullanıcıya hatasız tepki vermelidirler. Bu yüzden daha program yazılmadan önce programa dökülecek matematiksel metodun doğruluğunun kontrol edilmesi, ve böyle yüksek işyükü karşısında nasıl bir performans göstereceği düşünülmelidir.

Programların doğruluğu konusunda daha önce yaptığımız örnekler bir fikir verebilir. Örneğin banka borcu hesaplama programında yanlışlıkla 'ay' girdisinin eksi bir sayı olarak verilmesi durumunda olacaklar konusunda bir önlem aldık. Aldığımız önlem yaptığımız matematiksel tanımdan yola çıkıyordu. Oysa faktoriyel hesabı örneğinde yaptığımız teorik öncalışma bu açıdan eksikti, ve programımız da bu eksikliği yansıtıyor. Sonuçta faktoriyel hesabının girdisi bir tamsayı ve eksi değerler de gelebilir, bunun önlemini almak gerekirdi. Bu tür basit hatalar son derece yaygındır. Web sitelerinde gördüğünüz birçok hata mesajının arkasında bu tür basit programcı hataları yatıyor. Programcılık eğitiminin ve ders kitaplarının program yazmadan önce yapılması gereken öncalışmaları tamamen gözardı etmiş olması, ve bu yüzden yetişen programcıların da bu disiplini edinmemiş olması bu hataların esas sebebidir.

Performans ise yönteme göre değişmektedir. Buna örnek olarak bölümün başında sayıların üssünün alınması probleminden bahsetmiştik. Hatırlamak gerekirse şöyle iki yaklaşım mümkündü: 2^8 gibi bir sayıyı $2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2$ olarak yazabileceğimiz gibi $((2^2)^2)^2$ olarak ta yazabiliriz. Birinci ifadeye bakarak bu sayıyı hesaplamak için 7 çarpma işlemi gerekiyor gibi gözüküyor. Ama ikinci ifadeye bakınca, her kare alma işleminin sadece bir çarpım gerektirdiğini düşünürsek, sadece 3 çarpma bu hesaplama için yeterli oluyor. Şimdi şu soruyu soralım: x^y sayısının hesaplanması ne kadar zordur? Burada zorluktan kasıt hesabın kaç adımda yapılabileceği, yani ne kadar süreceğidir. Eğer ikinci ifade şeklini bulmuş olmasaydık bu sorunun cevabı $y-1$ olurdu çünkü birinci ifade şekli bize x^y'yi bulmak için x'i $y-1$ defa kendisiyle çarpmak gerektiğini söylüyor. Ancak ikinci ifade şekli bize bu hesabın çok daha kolay olduğunu söylüyor. Merak edenler için ikinci yöntemle x^y sayısını hesaplamak yaklaşık $\log_2 y$ zaman alacaktır, ki bu sayı $y-1$ sayısına göre daha küçüktür, özellikle y'nin büyük değerleri için çok daha küçüktür.

Buradaki farkın işlemin sonucuna dair bir fark olmadığına dikkatinizi çekerim. Her iki yöntemle de x^y değerinin buluruz. Yani fark sadece hesaplamaya dair bir farktır. Yöntemin yarattığı bu fark bilgisayar bilimlerinin matematiğin bir alt disiplini olarak gelişmesinin altında yatan temel unsurlardan biri olmuştur. Bilgisayar bilimlerinin kısa tarihi bu işle uğraşan bir çok yetenekli insanın bu tür problemleri çözmek için yarattığı zarif çözümlerle

dolu. Matematik bilimi bize doğru hesabın ne olduğunu söylerken, bilgisayar bilimleri de bu hesabın nasıl yapılacağı sorusunu yanıtlamaya çalışıyor. Günümüzde fizik gibi temel bilimlerin veya endüstri mühendisliği uygulamalarının önünde duran ve zor olduğu için hayata geçirilemeyen bazı hesaplamaların bilgisayar bilimleri geliştikçe ve daha etkili yöntemler ortaya koydukça zor olmaktan çıkması mümkündür.

Bu konuda çarpıcı bir örnek çarpma işlemine dair. İki tane herbiri h haneli olan sayıyı çarptığımızı düşünün. Bunu kağıt üzerinde yaparken ikinci sayının her hanesindeki değeri birinci sayının haneleriyle çarpar, kaydırarak altalta yazar ve sonunda toplarız. Bu yüzden $h \cdot h$ tane rakam çarpımı gerekir gibi gözüküyor; en azından 1960'lara kadar dönemin matematikçileri böyle olduğunu düşünüyordu. Oysa Karatsuba isimli genç bir Sovyet matematikçi bu işlemin çok daha az (yaklaşık $3 \cdot n^{1.5}$) rakam çarpımıyla yapılabilmesini sağlayan bir yöntem geliştirdi. Çarpma gibi son derece temel bir işlemin bile nasıl daha iyi yapılabileceği konusunda sadece elli yıl önce bir yenilik ortaya çıkması kayda değerdir.

7.6.3 Karmaşık verilerin temsili ve işlenmesi

Hemen her işte bilgisayardan yararlanılan günümüzde son derece karmaşık verilerin bilgisayara uygun bir sistematikle emsil edilmesi ve hızlıca işlenebilmesi gerekir. Yazı, ses, ve görüntülerin nasıl dijitalleştirildiğinden daha önce sözetmiştik. Burada ise kalabalık verilerin ve bunların arasındaki ilişkilerin temsili, ve yüksek performansla işlenmesinden bahsedeceğiz.

Örnek olarak kullandığımız e-posta programını ele alalım. Eğer istersek program e-postalarımızı gönderici adı veya tarihe göre sıralayabiliyor. Benim posta kutumda on bin e-posta var ve yine de çok kısa sürede sıraya diziliyor. Böyle bir sıraya dizme işini nasıl yaparsınız? İlk akla gelen çözümlerden biri bu işi de aynı bir iskambil oyununda dağıtılan kağıtları elimizde sıraya dizer gibi yapmak. İskambil kağıtlarını sıralarken bir tane alıyoruz, öbür elimizde zaten sıralanmış olan kağıtları inceliyoruz ve yeni kağıdı doğru yere sokuşturuyoruz, sonra yerden yeni bir kağıt alıyoruz, vs., ta ki yerdeki kağıtlar bitene kadar. İlk başlarda doğru yere koyma işlemini bir hamlede yapıveririz. Ama sonlara doğru elimizdeki kağıt destesi kalabalıklaştıkça doğru yeri bulmak için bütün kağıtları şöyle bir süzmemiz gerekir.

Benim on bin tane e-postamı mesela tarih sırasına dizmek için bu yöntemi kullansaydım ne kadar zaman alırdı? On bin defa bir e-posta alacağım, öbür elimde en eskiden en yeniye doğru sıralı tuttuğum e-postaları tek tek inceleyeceğim, elimdeki e-postadan daha yeni bir e-posta bulursam hemen onun önüne koyacağım. Tabi iş ilerlerken elimde sıralı tuttuğum e-postaların sayısı gittikçe artacak ve doğru yeri bulma işlemi gitgide daha fazla zaman alacak. Çok kötümser bir tahminle elimde sıralı duran e-postaların hepsini incelemek zorunda olduğumu varsayalım. İlk e-postayı başlangıçta boş olan elime hemen koyabilirim, ama ikinci adımda 1, üçüncü adımda 2, ..., son adımda 9999 e-posta incelemek zorundayım. Yani toplam 1+2+ ... + 9999 e-posta incelemem gerekecek; yaklaşık $10000^2/2$=50milyon. Sıralanacak şeylerin sayısına n dersek bu yöntemi uygulamak $n^2/2$ zaman alıyor. Emin olun ki bunu bilgisayara yaptırsak bile başa çıkılması zor bir iştir. Kurumsal bilişim sistemlerinde sıralanacak veri sayısı on binden çok daha fazla olabiliyor.

Bilgisayar bilimlerinin uğraştığı problemler arasında sıralama gibi kalabalık veri işleme problemlerini çözecek yöntemlerin geliştirilmesi vardır. Sıralama konusunda yukarıda bahsettiğimizden çok daha iyi yöntemler bulunmuştur. Şunu düşünün: sıralanacak n tane şeyi iki eşit büyüklükte gruba ayıralım, her iki grubu kendi içinde sıralayalım, sonra grupları birleştirelim. Buradaki avantaj şudur: küçük gruplar daha az karşılaştırma işlemiyle sıralanabilirler. Örneğin on bin e-postayı beşer binlik iki gruba ayırırsak gruplardan her biri $5000^2/2$=12.5milyon adımda sıralanabilir. Bunları birleştirmek için ekstradan 10.000 adım daha gerekeceğini düşünürsek, toplam sıralama süresi 12.5milyon+12.5milyon+10.000 olur ki bu yukarıda bulduğumuz 50 milyon sayısının yarısı kadar. Bu yöntemi daha da ileriye götürerek grupları da kendi içinde ikiye, ve onları da yine ikiye bölebiliriz. Bunu devam ettirdiğimizde işlemin süresinin $n \cdot \log_2 n$ olacağı hesaplanmıştır. On bin e-posta için bu rakam yaklaşık 133bin ediyor. Bu ilk yöntemimizdeki 50milyon rakamına göre muazzam bir tasarruftur.

Bununla ilişkili başka bir problemi ele alalım. Nesneleri sıralamaktan kurtulmanın bir yolu onları zaten sıralı olarak saklamaktır. Örneğin bir işletmenin ürün stoğu kayıtları malların ürün koduna göre sıralı olarak, bir okulun öğrenci kayıtları öğrenci numarasına göre, veya bir kütüphanenin kayıtları kitapların kodlarına göre sıralı olarak tutulur. Bu işlemi yaparken sisteme yeni

bir nesne ekleneceği zaman aynı iskambil kağıtlarındaki gibi doğru araya sokulması ve sağındaki/solundaki nesnelerin kaydırılması gerekir. Bilgisayar hafızasında böyle bir kaydırma işlemi oldukça pahalıya patlayabilir (yani uzun sürebilir). Düşünün ki bir kütüphanenin raflarında hiç yer yok ve yen bir kitabı araya eklemek istiyorsunuz. Kitap koduna göre sıralamanın bozulmaması için yeni kitabın önündeki veya arkasındaki bütün kitapları kaydırmak raflar, hatta koridorları kapsayan bir kaydırma işlemi yapmak gerekir. Tabii ki bir kütüphanenin raflarında bu tür durumlara karşı az miktarda boşluk bulunur ve bu durum pek yaşanmaz. Bilgisayar hafızasında dizili nesneler sözkonusu olduğunda ise ya bu şekilde boşluklar bırakmak, ya da başka yöntemler geliştirmek gerekir. Bilgisayar bilimcilerinin uğraştığı problemler arasında bu yöntemlerin geliştirilmesi ve uygulanması da vardır. Yine bu grup problemlerden bir nesnelerin arasındaki ilişkilerin tutulmasıdır. Facebook gibi bir sitedeki arkadaşlık ilişkilerini düşünün. Buradaki nesneleri kütüphanedeki kitaplara benzetirsek nesneler arasındaki ilişkileri göstermek için kütüphane rafları arasında milyonlarca iplik bağlamak gerekirdi. Bunun nasıl bir karışıklık olduğunu düşünün. Neyse ki bilgisayar ortalığı dağıtmadan bu iplikleri takip edebilecek bir makinedir. Tabii bu tür problemlerde de işlemin hızlı gerçekleşmesi gerekir.

Böylece programcılık teorisi ve pratiği sadece girdi-çıktıları tanımlanmış bir işlemi gerçekleştirecek programların yazılmasını değil, gerçek hayata dair karmaşık bilgilerin temsiline ve etkili şekilde işlenmesini mümkün kılan çözümlerin tasarımını da içeriyor. Bu çoğu zaman birikimsel büyüyen bir iştir; yani varolan veri tasarımları ve onları işleyen programlar yeni problemlerin çözümünde kullanılabilir. Ancak işin içinde her zaman varolan çözümlerin yaratıcı bir şekilde biraraya getirilebilmesi veya yeni veri yapılarının tasarlanması ihtiyacı vardır.

7.6.4 Programlama neden mühendislik değil sanattır

Bunun birden fazla sebebi var. Bu sebeplerden biri program yazımın edebi bir iş olması. Burada incelediğimiz basit programlarda komutları satırlara anlaşılır bir şekilde sermeye, girdilere ve işlevlere taktığımız isimlerin amacına uygun olmasına gayret ettik. Programlama dilinin gramer kuralları dışında bu yaptıklarımızın bilgisayar için hiçbir anlamı yok. Tek bir satıra sıkıştırılmış, girdilere anlamsız isimler verilmiş bir program olsaydı bilgisayar

için hiç farketmezdi. Zaten bizim bunları yapmaktaki amacımız da programı kendimiz ve başka programcılar için okunaklı kılmak. Üstelik bir program okunaklı olmaktan oldukça öteye geçebilir. Programdan beklenen işi gerçekleştirmek için izlenen yöntemin özgünlüğü, tekniklerin kullanımındaki yalınlık ile birleştiğinde meslekten biri için okunması 'zevkli' bir program ortaya çıkar. Bu anlamda bir bilgisayar programı bir edebiyat eserine benzer, ancak okurundan özel bir dili bilmesini bekler. Bu açıdan ilk bakışta benzerliğine karşın programcılık mühendislikten oldukça farklıdır.

Bir başka sebep ise programcının işinin bazı tekniklerle desteklenmiş te olsa özünde sezgi ve yaratıcılığa dayalı bir soyutlama pratiği olmasıdır. Birbirinden farklı problemlerle uğraşan programcı elindeki programlama tekniklerinden ve bilinen etkili veri yapılarından yola çıkar. İnsanların yaptığı ama artık bilgisayara yaptırılmak istenen bir dizi operasyonu önce anlamak, sonra elindeki araçlarla bir otomatikleştirme yaratmak zorundadır. Yapılan otomatik sistem çoğu zaman bir şirkette/kurumda süregiden işleyişin bir parçası olacaktır. Bu iş canlı bir yaratığın içine metalden bir mekanizma yerleştirmeye benziyor. Bunu yaparken yaratığı sakatlamamalı, hareketlerini kısıtlamamalı, aksine onun büyümesine ve daha serbest hareket etmesine katkıda bulunmalıdır. Çoğu zaman bir yazılımın kullanışlılığı onun doğru ve hızlı çalışması kadar kullanışlı ve zevkli olmasıyla ilgilidir. Nasıl ki bir otomobilde çekiş gücü kadar ergonomi ve tasarım güzelliği aranıyorsa.

7.6.5 Programlamanın farklı uzmanlık alanları

Programlama pratiğinin uygulandığı pek çok farklı alan var. Ancak bunların önemli bir kısmında farklılık programlama tekniklerinde değil sadece uygulama alanındadır. Öte yandan son derece benzer işler farklı teknikler gerektirebilir. Örneğin fotoğraf görüntüleme programı ile video oynatma programı son derece farklıdır. Video oynatma programından gerçek zamanlı performans beklenir, çünkü videoyu kesik kesik oynatırsa hiçbir anlamı olmayacaktır. Bu yüzden bu amaca uygun programlama teknikleri gerekir. Öte yanda fotoğraf programında böyle bir kriter yoktur. Burada programcılığın içerisindeki kullandıkları teknikler açısından farklılaşan alanlara göz atacağız.

Bu alanlardan biri paralel programlama. Bu bölümün başında süreçlerden ve belirlilik/belirsizlik kavramından sözetmiş-

tik. Süreçlerdeki belirsizliğin bir kaynağı süreçlerin birbiriyle etkileşimidir. Robotlar arasında yapılan bir yarışı düşünün. Bu robotların herbiri belirlenmiş süreçler yaşıyor olsun (yani bir programı takip ediyor olsun). Buna rağmen hepsinin beraber oluşturduğu yarış süreci belirsizlik taşır. Bir robotun paleti yarışın bir noktasında kayıp devrilebilir, bir diğerine yanlışlıkla çarpıp sendeletebilir, vb. Herbiri belirlenmiş, ancak birbirinden bağımsız olan süreçlerin oluşturduğu böyle bir bütüne paralel , veya *eşzamanlı* (İng. concurrent) süreç diyoruz. Robot yarışı gibi bir eğlence etkinliği bir tarafa, örneğin çok uzun süren bir bilimsel hesabın yapılmasında veya Google gibi milyonlarca kullanıcıya hizmet veren bir servisin çalıştırılmasında birsürü bilgisayarı birlikte işe koşarak problemi çözmek için böyle süreçler kullanılır. Böyle eşzamanlı bir sürecin tasarımı ve programlanması da tek bir süreçten oldukça farklıdır.

Başka bir alan yapay zeka alanıdır. Bu alandaki çalışmalar esas olarak insan zekasının bilgisayar programlarıyla taklit edilmesini amaçlar (Russell and Norvig, 2003). Bu en tartışmalı alanlardan biridir çünkü insan zekasının nasıl çalıştığı tam olarak anlaşılmış değil. Genel olarak bu alanda yapılan işlerin başarısını sınırlı bir konuda insanlarla boy ölçüştürerek tartabiliyoruz. Örneğin satranç şampiyonlarını yenen programlar yapılmıştır. Son on yılda geliştirilen programlarla el yazısı tanıma, veya insan yüzlerini ayırdetme gibi işler gerçekleştirilebiliyor. Ne var ki yapay zeka alanında geliştirilen teknikler genel olarak bu alt-alanlara özgü tekniklerdir. Yani oyun programlama ile uğraşanlarla görsel algılama problemleri ile uğraşanlar tamamen başka teknikler kullanıyorlar. Bu alanı birleştiren yaklaşımın ne olduğu hala tartışmalı bir sorudur. Ancak bu sorunlar bir yana yapay zeka alanında geliştirilen teknikler sanayide süreçlerin optimizasyonu gibi önemli problemlere çözüm getiriyorlar.

Bunun dışında kitapçı raflarında görebileceğiniz veritabanı programlama, web uygulamaları, kurumsal programcılık gibi pek çok uzmanlık alanı var. Bunların piyasadaki talep açısından gerçekten birer alan oldukları kesin, ama çoğunun kendine özgü tekniklerinin ne olduğu belirsizdir. Bu tür alanlarda uzmanlaşmanın belirli teknolojileri öğrenmekten ibaret olduğu gibi bir algılama oluşmuştur. Sonuç ise çalışmayan kurumsal web siteleri, ekranlarımıza dökülüveren veritabanı hata mesajlarıdır. Bu tür uzmanlıkların temel programcılık eğitiminin üzerine oturması

gerektiği unutulmamalıdır. Ne yazık ki piyasada yetişmiş programcılara duyulan açlık uzmanlık adı altında altyapısı çürük eğitim yaklaşımlarının benimsenmesine yolaçıyor. Bu da mesleğe yeni atılan pek çoklarının kestirme diye girdikleri yollarda kaybolmasıyla, ve kurumsal bilişim projelerinin son derece verimsiz olmasıyla sonuçlanıyor.

Son olarak gözden kolay kaçan bir alandan bahsedelim: programlama dilleri. Java gibi bir programlama dili ortaya çıkıp yaygınlaştığında pek çoğumuz 'işte bu' deriz, 'artık hepimiz bunu kullanacağız'. Herkesin aynı programlama dilini kullanması büyük bir avantajdır. Böylece problemlerimizi çözerken birbirimizin programlarından parçalar alıp yararlanabiliriz. Sanayi de de ürüm uyumluluğu açısından bu arzu edilen bir durumdur. Oysa geçmişe baktığımızda yeni programlama dillerinin giderek artan sayıda ortaya çıktığınız görüyoruz. Bunun pek çok sebebi var. Bilgisayar programlamanın konusu olan problemlerin artması ve çeşitlenmesi bu sebeplerden biridir. Örneğin son yıllarda ortaya çıkan programlama dillerinin/teknolojilerinin bir kısmı web programcılığına yöneliktir. Web uygulamalarındaki artış bu uygulama alanının kendine özgü ihtiyaçlarına cevap verebilen teknolojileri gerekli kılmıştır. Varolan programlama dilleri de bu ihtiyaçlara göre evrilmekte sıkıntı çekerler. Farklı uygulama alanlarına yönelik bu türden pek çok dil ortaya çıkmaktadır. Sonuçta bu konuya evrimsel açıdan bakarsak bu durum normaldir. Çünkü çevre koşullarının hızlı değiştiği durumlarda evrim süreci de hızlanır. Çeşitlilik artar ve değişim hızı da onunla birlikte artar. Bu yüzden yeni nesil programcılar açısından en önemli becerilerden biri programlama tekniklerini programlama dillerinden bağımsız bir şekilde özümsemek ve gereğinde yeni diller ve teknolojilerle çalışmaya hızla adapte olabilemektir.

Bölüm 8

Geçmişten Geleceğe: Bilgisayarın Evrimine Sosyo-Ekonomik bir Bakış

Popüler medya bilim ve teknolojideki gelişmeleri 'kahraman'larla veya 'dahi'lerle ilişkilendirmeye pek heveslidir. Gerçekten de Arşimed, El-Harezmi, ve Einstein gibi dahilerin bilime katkıları kendi dönemlerindeki bilimin sıçrama yapması için son derece önemli bir rol oynamıştır. Ancak bilim ve teknolojinin yönünü belirleyen sadece yaratıcı zekalar değildir. Onların ortaya attıkları yeniliklerin kabul görüp serpilmesine izin veren bir kültürel ve ekonomik ortam da gerekir. Tarihçilerin nadiren de olsa ortaya çıkardığı örnekler bildiklerimizden çok daha fazla sayıda dehanın ve buluşun bu yüzden kabul görmediğine ve tarihin karanlığına gömüldüğüne işaret ediyor.

Önceki bölümlerde bilgisayar teknolojisinin özelliklerini ve bunların gelişimini kendi bütünlüğü içerisinde ele aldık. Bu bölümde ise bilgisayarın gelişimini onu çevreleyen toplumsal ve ekonomik koşulların değişmesiyle beraber ele alacağız. Kitaba böyle bir bölüm eklemekteki amaç okurların bilgisayar teknolojisinin evrimine nelerin yöne verdiği ile ilgili temel bilgileri edinmesi, ve böylece gelecekte nasıl değişebileceğiyle ilgili sağlıklı değerlendirmeler yapabilmesine katkıda bulunmak. Bilgisayar gibi

herşeyin çok hızlı değiştiği, ve bu değişimin birçok cephede birden şekillendiği bir alanda, özellikle bu işi meslek edinmek veya bu alanda girişimci olmak isteyenler için bu değişimi öngörebilmek son derece önemli olacaktır.

Burada bilgisayar tarihinin sosyo-ekonomik bir özetini bir özetini aktarırken başarı hikayeleri kadar başarısızlık hikayelerine de yer vermeye çalışacağım. Bunlar olmadan olayların nasıl geliştiğini anlamak, ve bilgisayarın evrimine yön veren etkileri kavramak mümkün olmaz. Aşağıdaki ilk kısımda bilgisayarın henüz evlere, okullara, ve ofislere girmediği, 1980'lere kadar olan dönemi anlatılıyor. Dinozorlar çağı olarak adlandıracağımız bu dönem bilgisayarların çok büyük ve hantal olduğu, sınırlı sayıda büyük kurumda sınırlı işler için kullanıldığı bir dönem. Günümüze pek benzemese de bu dönemin tarihi teknolojiyi şekillendiren temel ekonomik ve politik etkileri, IBM gibi büyük oyuncuların ortaya çıkışını, üniversite ve sanayi arasındaki işbölümünün şekillenişini, bilgisayar bir sanayiye dönüşürken küresel ölçekte oyuncular arasındaki rol dağılımının ortaya çıkışını ortaya seriyor.

İkinci kısımda ele alınan ve 1990'lara kadar olan onyıla ise bilgisayarın küçülüp ucuzlaması, ofislere ve az da olsa evlere yayılması damgasını vurmuştu. Bunun sonucu bu farklı kullanım yerlerindeki farklı işleri görecek yazılımların hızla önem kazanması, ve teknolojinin ve piyasanın dönüşümünde donanımın yanısıra belirleyici bir temel unsur olarak yerini almasıydı.

Bilgisayarın ve İnternet'in evlere girmeye başladığı ve web'in ortaya çıktığı 1990'lar ise üçüncü kısımda inceleniyor. Bilgisayar kullanımının çok hızla yaygınlaştığı bir dönem olmasına rağmen bu dönem teknolojinin esaslarının en az değiştiği dönem. Bu dönemde asıl değişim bilgisayar teknolojisinden öte, onun taşıyıcısı olduğu bilginin üretilmesi, paylaşılması ve bilgiye erişimle ilgili bir değişimdi.

Son kısımda ele alınan yeni binyıl ise bilginin kendisiyle ilgili bu değişimin, bir bilgi erişim ve haberleşme makinesi olarak bilgisayrın yaygınlaşmasının ekonomide ve sektörün kendisinde yolaçtığı dönüşümlere sahne oluyor. Bu kısımda ayrıca bilgisayarın artık taşınabilir bir teknoloji olması (cep telefonları), bunun şimdiye kadar yolaçtığı ve önümüzdeki yıllarda yolaçması beklenen değişimler ele alınıyor.

8.1 1980'e kadar: Dinozorlar çağı ve öncesi

Yakın bir geçmişte ortaya çıktığı için bilgisayarın çok iyi kayda geçmiş bir tarihi olduğunu düşünebilirsiniz. Ne var ki işn aslı pek öyle değil. Bunun bir sebebi henüz pek az tarihçinin bu konuyla ilgilenmiş olması. Tarihçilerin aracılığı olmazsa bu konudaki malzemelerin çokluğu bizler için bir kafa karışıklığı sebebi olabilir.

Bunun dışında bilgisayar tarihi üzerine bir yazının oluşumunu engelleyen iki neden daha var. Bunlardan biri bilgisayar teknolojisinin şekillenmesinin en hızlı geliştiği yılların ikinci Dünya savaşına denk gelmesi. Bu yıllarda bilgisayar teknolojisine en büyük ilgi askeri uygulamalardan gelmişti. O yüzden de çalışmalar büyük bir gizlilik içerisinde yürütülüyordu; hala da bu dönemdeki çalışmalarla ilgili pek çok konu açıklığa kavuşmamıştır.Ayrıca bu gizlilik içerisinde önemli teknik problemlerin çözümlerinin nerede (ABD, İngiltere) ortaya çıktığını, kimin kimden esinlendiğini tespit etmek zordur. İkinci bir neden ise bilgisayarın tarihinin kendisinden önce gelen hesaplama teknolojileri ve matematikteki ilerlemelerle sıkı sıkıya ilişkili olması ve bu konulardaki tarih yazımının da yine sınırlı olması.

Bilgisayar adı verilen ilk elektronik makineler 1940'larda ABD ve İngiltere'de yapıldı. Ancak bunun arkaplanında hem matematikteki ilerlemeler, hem de daha önce otomatik hesaplama yapan makinelerin yapımında yaşanmış başarı ve başarısızlıklar vardı. Bilgisayar tarihiyle ilgili varolan yazın tam da bu konuda ikiye ayrılmış gözükmektedir. ABD kökenli bilgisayar tarihi kitapları malzeme ve sayı olarak zengin olmakla beraber çoğunlukla bu tarihi sadece ABD'deki gelişmeler üzerinden anlatmaya eğilimliler ve bu yüzden arkaplan yoksunluğu hissediliyor. ABD özellikle 1950-70 döneminde yarışı önde götürmüş olmasına rağmen bu gruptaki eserler, özellikle bilgisayar teknolojisinin evrimini kavramak açısından eksik, ve dolayısıyla yanıltıcı. Buna karşın İngiltere kökenli tarihçeler tarihsel arkaplana daha çok hakkını verir görünüyor. Ben de bilgisayarın erken dönem tarihçesi için özellikle Campbell-Kelly ve Aspray'in kitabından yararlandım (2004).

8.1.1 Bilgisayarlardan önce hesaplama

17. yüzyıldan başlayarak Avrupa'da, özellikle de Fransa'da herşeyi ölçüp biçme ve sayma çılgınlığı başladı, ve modernizmin gelişimi boyunca bugüne kadar hızını kesmedi (Hacking, 2005). İlk hesap makinesi matematikçi Blaise Pascal tarafından bu dönemde

Fransa'da yapılmıştı ve toplama-çıkarma yapabiliyordu. Ancak çok uzun bir süre boyunca elle hesaplamöa yöntemleri hızla gelişirken mekanik hesaplama yerinde sayacaktır. Zaten o günün metal teknolojisinin yetersizliği yüzünden pek kullanışlı olmayan bu makineler çarpma, bölme, karekök bulma gibi işlemleri yapmaktan çok uzaktı. Oysa matematikte logaritmanın geliştirilmesiyle bu tür işlemler bir ölçüde kolaylaşmıştı (Cajori, 1991). Ciltler dolusu logaritma tablosu üretilmiş ve Avrupa matbaalarında bunların birçok kopyası yapılmıştı. Bir karekök işlemini düşünün: $\log \sqrt(x) = \log x^{1/2} = \frac{1}{2} \log x$. Böylece x sayısının karekökünü bulmak için tablodan logaritmasını bulur, ikiye böler, ve yine tablodan bunun karşılığı sayıyı bulursunuz. Buna karşılık sadece toplama-çıkarma yapabilen ilkel hesap makineleri pratikte yararsız kalmış, sadece kavram olarak ilginç olduklarından birkaç zengin meraklı için sınırlı sayıda üretilmişlerdir.

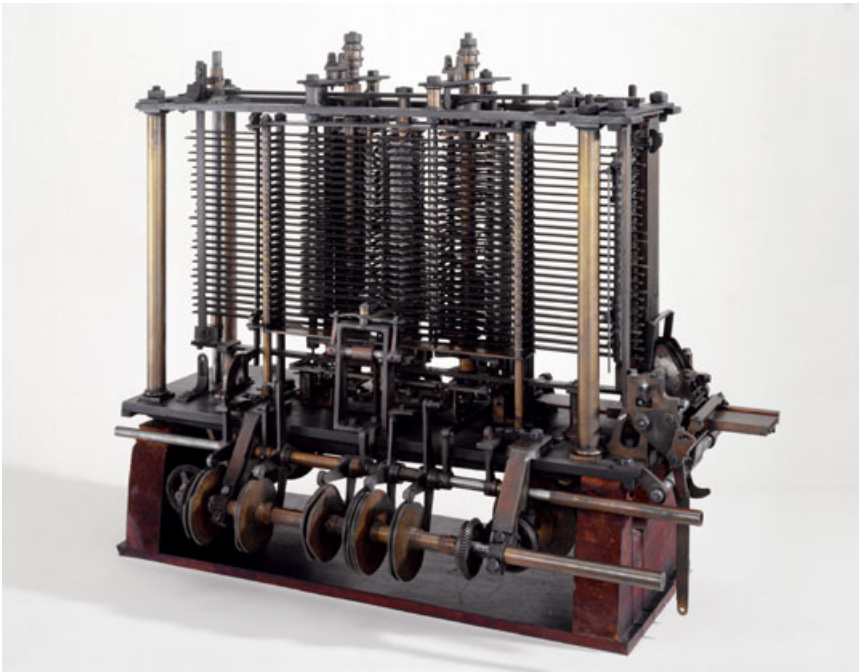
Modernizmin ve endüstrileşmenin sayılara açlığı hızlanarak devam edecekti. Arazi kadastro kayıtları, ticaret ve vergi kayıtları, suç kayıtları, sigorta poliçeleri, liste uzayıp gidiyordu. Devlet ofisleri ve büyük bankalar bu rakamları da aynı fabrikalarda malların işlenmesi gibi, büyük ofislerde işlemek için sistemler geliştirdiler. Memurların rakamlarla dolu bu listeleri toplayıp, tasnif ettikleri bu ofisler yüzlerce kişinin çalıştığı fabrikalardan farklı değildi. Hataları azaltmak için karşılıklı kontrol sistemleri geliştiriliyordu (Campbell-Kelly and Aspray, 2004). Bu rakam yığınlarına anlam verebilmek için istatistik bilimi hızla gelişmeye başlamıştı (Hacking, 2005).

Bu hızlanma içerisinde bazı hesaplar özellikle sorun yaratıyordu. Örneğin gittikçe artan deniz yoluyla ticaret, Dünya denizlerinde güvenli seyahat edebilmek ve yolunu bulabilmek için her yıl ve yılın her günü için yıldızların konumlarını gösteren seyir tablolarına ihtiyaç duyuyordu. Kuzey Avrupa'nın nehir ağzına kurulmuş olan Londra ve Rotterdam gibi şehirlerinde gemiler gelgit yüzünden karaya oturmasın diye Ay'ın hareketlerine göre gelgit tabloları yapmak gekeriyordu. Bu işler logaritma tablosu yapmaktan farklıydı çünkü her yıl yeniden hesaplanmaları gerekirdi. Aynı zamanda yetenekli bir matematikçi olan Charles Babbage bunun makinelerle yapılabileceğine inanıyordu. Heyecanlı ve karizmatik bir karakter olan Babbage 1820'lerin başında bu tür tabloları hem hesaplayacak hem de yazdıracak bir makine tasarlamaya girişti, ve İngiliz hükümet yetkililerini

bu projesine mali destek vermeye ikna etti (Campbell-Kelly and Aspray, 2004). Ancak bu makinenin gerektirdiği hassasiyet zamanın ötesindeydi, ve Babbage düşündüğünden çok daha fazla para (hem kendisinin hem devletin) ve zaman harcamasına rağmen makine asla tamamlanamadı. Babbage'i bilgisayar tarihi açısından asıl ilginç kılan şey bu tasarımı daha da ileriye götürdüğü Analitik Motor projesidir. Babbage bu tasarımda makinenin nasıl hesap yapacağını yönlendirmek için o yıllarda dokuma tezgahlarında kullanılmaya başlamış bir teknolojiden yararlanmayı düşünüyordu. Fransız Jacquard'ın icat ettiği bu dokuma tezgahı teknolojisi, müzik kutularına benzer şekilde, kumaşa dokunacak desenlerin delikli kartlara işlenerek dokuma tezgahında otomatik olarak dokunmasına imkan veriyordu. Kısmen bundan yararlanan Babbage'ın tasarımı esas olarak bugün kullandığımız programlanabilir bilgisayarlara son derece benzemektedir. Ancak heyecanlı ve arayışları dur durak bilmeyen kişiliği aynı zamanda projenin sonu olacaktı, ve makine ancak kısmen yapılabilecekti (resim 8.1). Babbage'ın sürekli hedef değiştirdiğini ve bir türlü pratik bir makine yapamadığını gören hükümet projeden desteğini çekti.

Babbage'ın gözönündeki başarısızlığı aynı zamanda uzun bir süre için bu tür makinelerden umut kesilmesine yol açacaktı. Onun yerine analog hesaplama yapan makineler ilgi görmeye başladı Campbell-Kelly and Aspray (2004). Bu makineler ancak belli bir olguyla analoji/benzeşim kurduklarından genel amaçlı hesaplama mekanizmaları değildi. Ama örneğin gelgit hesapları yapmak için 1876'da İngiltere'de makineler yapıldı ve başarıyla kullanıldılar (1950'lere kadar). Bu tür makineler Güneş ve çevresindeki gezegenlerin dönüşünü gösteren çok gelişmiş maketlere benziyorlardı. 1900'lerin ortalarına kadar farklı amaçlar için bu türden birçok makine yapılacaktı (resim 8.2); hem Avrupa'da hem ABD'de.

Analog makinelerin başarısı uzun bir süre sayısal hesaplama yapma makinelerine ilgi gösterilmesini engellemiştir. Ancak bu süreçte bir ihtiyaç birikmesi oluşuyordu. Örneğin 1910'larda hava tahmini yapma yöntemleri geliştirilmişti. Fakat bunun için o kadar çok hesaplama gerekiyordu ki bir sonraki günün hava durumunu tahmin etmek için aylarca hesap yapmak gerekirdi! Geliştirilen hava tahmini yöntemlerinin uygulanması elli yıla yakin bir süre, uygun hızda bilgisayarların ortaya çıkışını bekleyecekti (Campbell-Kelly and Aspray, 2004).



Şekil 8.1: Babbage'ın 1834'de kısmen tamamladığı Analitik Motor (Londra Bilim Müzesi)

Şekil 8.2: Diferansiyel denklem çözümü için kullanılan bir analog bilgisayar, 1937 Cambridge Üniversitesi Matematik Laboratuarı.

Bu arada ABD'de ortaya çıkan bir teknoloji bizim için ilginçtir. ABD'de her on yılda bir yapılan nüfus sayımı verileri çok önemseniyor, ancak bunların tasnif edilip özet sonuçların yayınlanması için yıllarca, çok sayıda insanın çalışması gerekiyordu. 1980'lerde artık metal sanayi gelişmiş ve toplama-çıkarma yapabilen hesap makineleri yaygınlaşmaya başlamıştı. Herman Hollerith isimli bir girişimci bu teknolojik ilerlemeleri değerlendirerek nüfus sayımı verilerinin delikli kartlara jaydedilmesi ve makinelerle sonuçlarının sayılıp çıkartılmasını sağlayacak bir sistem geliştirdi. İlk kez 1890 ABD nüfus sayımında kullanılan bu sistem büyük başarı sağladı. Resim 8.3'da görülen makineye delikli kartlar teker teker yerleştiriliyor, deliklerin olduğu yerler elektrik akımının geçmesine izin veriyor, ve makinenin panosunda görülen sayaçlardan bazıları bu akıma göre değerini bir arttırıyordu. Böylece nüfus sayımında sayılan bir bireye karşılık gelen kartta kaç tane cevap seçeneği varsa makine bunları sayıyordu. Hollerith'in şirketi bu tür makineleri hem devlet kurumlarına hem büyük şirketlere pazarladı. Daha sonra Thomas Watson'un genel müdürlüğü döneminde, 1924'te bu şirketin adı IBM (Iternational Business Machines, Uluslararası İş Makinaları) olarak değişecekti. IBM'in bu dönemde yaptığı makinelerin günümüz bilgisayarlarıyla hiçbir benzerliği yoktu. Ama bu makineler sayısal verileri karton kartlarda tutmak (yani sayısal bellek) ve bunları saymak gibi önemli teknik ayrıntıları çözüyordu. Bu teknolojik ilerlemeler 1940'larda bilgisayarların ortaya çıktığı dönemde önemli bir pratik katkı sağlayacak, ve IBM de bu dönemin önemli aktörlerinden biri haline gelecektir.

İkinci Dünya savaşına kadar olan bu mekanik hesaplama dönemi teknolojik gelişmelerin doğrudan ekonomik katma değer ve toplumsal ihtiyaçlarla ilişkisini birkez daha gösteriyor. Charles Babbage'ın genel amaçlı bir hesaplama makinesi yapma hedefini dönemin bilim camiası heyecanla karşılamıştı. Ancak hem devletler hem de özel şirketler bunun yerine daha basit ama pratik ve kısa vadede katma değer yaratan teknolojilere yatırım yapıyorlardı. Farklı bir vizyonla, 80 yıl daha önce bilgisayarlar yapılmış olsaydı neler olurdu hiç bilemeyeceğiz.

8.1.2 İkinci Dünya savaşı ve büyük projeler

1800'lerin son yılları ile 1940 arasında matematikte hesaplama konusu ile ilgili önemli teorik çalışmalar yapıldı. Hilbert, Gödel, Church, Turing gibi Avrupa'nın farklı ülkelerinden matematikçiler

Şekil 8.3: Hollerith'in yaptığı ve ABD nüfus sayımında kullanılan makine, 1902 (Kaynak ABD nüfus bürosu, `http://www.census.gov`)

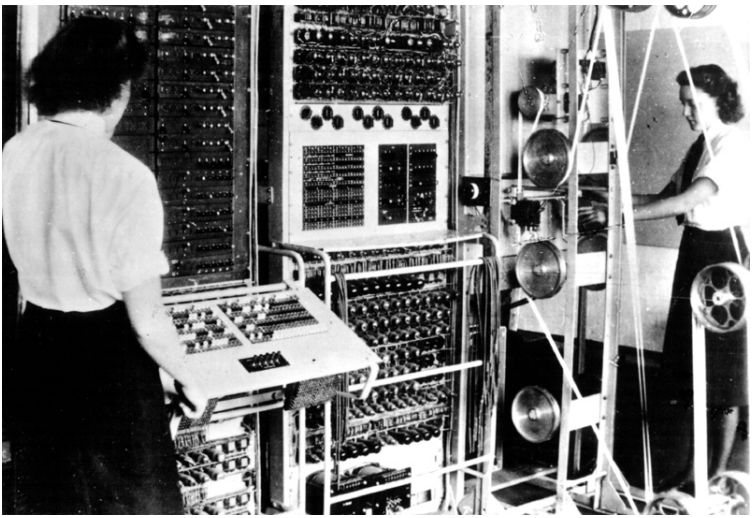
hesaplamanın ne olduğu, neyin hesaplanabileceği, ve bunun nasıl mekanikleştirilebileceği konusunda kafa yordular. Öte yanda analog makineler ve delikli kart teknolojisi iyice gelişmişti, ve belli ki işler böyle tıkırında giderken kimsenin yeni bir Babbage hayaline para yatırmaya niyeti yoktu.

Ancak ikinci Dünya savaşı sırasında ortaya çıkan bir sorun bu durumu değiştirdi. Alman ordusu haberleşme için birinci Dünya savaşından beri giderek geliştirdiği Enigma adlı bir şifreleme makinesi kullanıyordu. Savaş bir yandan tank ve uçaklarla yapılırken bir yandan da bir muhaberat ve teknoloji savaşı yaşanıyordu. Bu teknoloji geliştirme savaşından geriye atom bombasının berbat hatırası kalmıştır, ancak bunun dışında da birçok şey oluyordu. Savaşa sonradan katılacak olan ABD şifre kırmanın yanısıra büyük savaş silahlarının menzil hesaplarını yapmak için hesaplama makineleri geliştirmeye çalışıyordu. Sonuçta savaşa katılan çok sayıda devlet bir anda şifre-kırma (kriptoanaliz) gibi alışılmadık hesap gücü ve hızı gerektiren bir konuya muazzam para ve enerji yatırmaya başlamıştı.

Çalışma esasları itibarıyle bugün kullandıklarımıza benzer bilgisayarların ilk kez yapıldığı bu Dünya savaşı dönemi ayrıntı zamanda bilgisayar tarihinin en karanlıkta kalmş ve tartışmalı dönemidir. Bu tarihin başrol oyuncuları ABD ve İngiltere savaş sırasında sınırlı ölçüde işbirliği yapıyordu. Ancak savaş koşullarının yarattığı gizlilik içerisinde kimin kimden ne öğrendiği ve bazı problemleri kimin ilk çözdüğü tartışmalıdır. Burada ancak sınırlı bir özet vermekle yetineceğiz.

Savaş yılları İngiltere'sinde bu konudaki çalışmaların başlangıcı 1939. O yıl Almanları şifreleme için kullandığı Enigma makinesinin şifrelerini kırmak için başlatılan projeye hesaplama teorisi konusunda önemli katkılar yapmış olan Alan Turing de dahil edilir. 1940'ta yapımı tamamlanan ve Bombe adı verilen şifre kırma makinesi bilgisayara benzememekle beraber sayısal verilerin depolanması ve işlenmesi konusunda İngiltere'de çalışan bilim insanlarının deneyimine önemli bir katkı sağlar. Bu projede Turing ile birlikte çalışan Max Newman daha sonra 1943'te Colossu adlı başka bir şifre kırıcı yapacaktır. Amerika'daki eşdeğer makinelerden iki yıl önce gerçekleşen Colossus (resim 8.4) kimilerince ilk elektronik, programlanabilir bilgisayar sayılıyor (Copeland, 2004).

Turing'in teorik çalışmaları arasında özellikle Evrensel Turing



Şekil 8.4: İlk programlanabilir bilgisayarlardan Colossus, 1943 (Kaynak: Computer History Museum)

Makinesi kavramı bu makinelerin tasarımında önemli bir yer tutuyordu. O zamana kadar yapılan ve sadece belirli bir hesabı yapabilen analog hesaplayıcılar yerine, sorunlar değiştikçe yeniden programlanabilen bir genel amaçlı makine gerekliydi.

Bu arada ABD'de çok daha büyük bütçeli projeler yapılmaktaydı. Bunlardan biri 1943'te tamamlanan ve IBM'in mali desteğiyle Howard Aiken tarafından Harvard üniversitesinde yapılan Mark I isimli makineydi. IBM'in bütün teknolojisini kullanan Aiken Babbage'ın düşündüğü gibi delikli kartlarla yönlendirilen bir hesaplayıcı yapmıştı. Koşullu işlemleri yapamayan, ve bu yüzden tam olarak bilgisayar diyemeyeceğimiz Mark I yine de o dönemde birçok genç araştırmacının bu teknolojilerle tanışmasına vesile oldu ve kamuoyunda bu teknolojilere ilgiyi arttırdı. Ayrıca bu projede oluşan tecrübe IBM'in savaş sonrası bilgisayar sanayinde en önde yer almasının önemli bir sebebi olacaktı.

Aynı sıralarda Pensilvanya'daki Moore okulu başka birçok okul gibi sürekli gelişen ABD ordu silahlarının balistik ve menzil hesaplarının yapımı konusunda görev almıştı. Bu okulda çalışan John Maucly ve John Eckert ayrıca radar cihazlarının

yapımıyla ilgili bir projeye dahildiler. Radar cihazları çevredeki herşeyi gösteriyordu, ve asıl kayda değer olan hareketli düşman araçlarını ekranda seçmek zordu. Çalıştıkları projede bu sorunu çözmek için ekrandaki görüntüyü saklayıp bir sonraki kareden çıkartarak hareketsiz nesneleri görüntüden yoketmek, sadewce hareketli olanları bırakmak gibi bir hedef vardı. Görüntüyü kısa bir süre saklamak için de vakum tüpü gibi bazı parçaların kullanılması deneniyordu. Öten yandan bu ikili menzil hesapları için eski usul analog hesaplayıcıları kullanan ve iyileştirmeye çalışan bir ekibin parçasıydılar. Maculy, cihazda kayda değer bir hızlanma için radar projesinde veri saklamak için kullanılan vakum tüplerini hesaplayıcıda kullanmayı düşünüyordu. Bu proje bir yıl rafta bekledikten sonra, işler çıkmaza girince 1943'te destek gördü. Sayısal bilgiyi mekanik parçalar yerine elektriksel bir cihaz olan vakum tüplerinde tutmak gerçekten kayda değer bir adım olacaktı. ENIAC adı erilen bu yeni makine ancak 1945'te savaşın bitiminden haftalar sonra tamamlanacaktı. Ordu bir anlamda ödediği paranın karşılığını alamamıştı, ama bilgisayar teknolojisinde önemli bir adım daha atılmıştı (Campbell-Kelly and Aspray, 2004).

Dünya savaşı sırasında yapılan bu makinelerin hepsi, Colossus, Mark I, ve ENIAC, easen devasa hesap makineleriydi. O dönemde yaygın olan ofis tipi mekanik hesap makinelerinden farklı olarak çarpma ve daha karmaşık aritmetik işlemleri yapmak için ayarlanabiliyor, elektrikle çalıştıklarından saniyede birkaç yüz aritmetik işlem yapabiliyorlardı, ki bu o zamanın koşullarında kayda değerdi. Ancak bu makinelerin programı pek değiştirilebilir değildi. ENIAC'ı farklı bir işlem için programlamak karmaşık kablo bağlantıları değiştirilerek yapılıyordu. Ev büyüklüğündeki bu makine sadece 1K bilgi saklayabiliyordu. Yine de ENIAC'ta hesaplama ve bellek için mekanik aksamın yanısıra elektriksel parçaların (vakum tüpler) kullanımı önemli bir yenilik ve hesaplama hızında artış getirecek, ve ileriki yıllarda gelişerek kullanılacaktı. Mekanikten elektriğe doğru gidiş, hesap makinesinden bilgisayara doğru gidişten bir adım önde ilerliyordu.

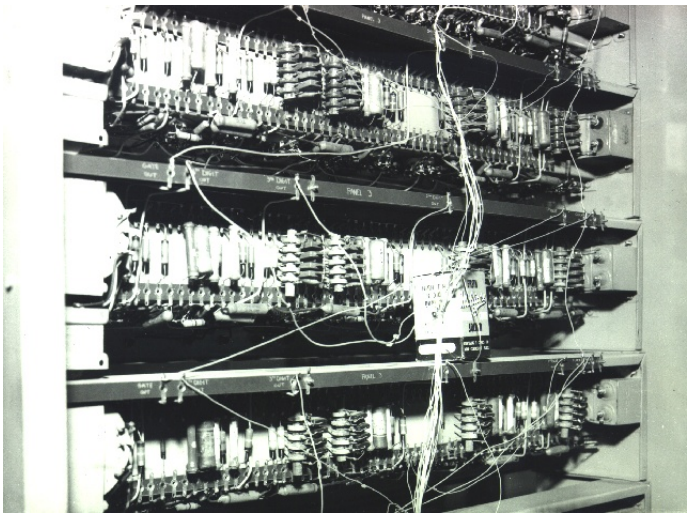
8.1.3 1945-1980: Savaş aletleri ticari makineye dönüşüyor

Savaşın sonlarına doğru, aynı zamanda ABD'nin atom bonbaası projesinde de çalışmış ve Einstein'in mesai arkadaşı olan matematikçi John von Neumann ülkede gerçekleşen bilgisayar

yağımı projeleriyle ilgilenmeye başladı. Pratik sonuçlara ulaşmaya odaklanmış bu projelerdeki ortak sorunları gören von Neumann bu teknolojilerin ilerlemesi için 'program'ların da aynı sayılar gibi bilgisayarın hafızasında tutulması ve makinenin kablolarıyla oynamaya gerek olmadan değişik işler için yeniden programlanabilmesi gerekliliğini görmüştü Ceruzzi (2003). Aynı aylarda Turing'in yazdıkları da buna paraleldi Copeland (2004); ancak bildiğimiz kadarıyla ikisi arasında bir iletişim yoktu. Esasen Turing'in teorik Evrensel Makine kavramına karşılık gelen 'von Neumann Mimarisi' terimi bugün bilgsayar mühendisleri tarafından programlanabilirlik özelliğini taşıyan makineler için kullanılıyor. Bilgisayar bilimcileri ise Turing Makinesi kavramını tercih ediyorlar.

Gelişmiş hesap makinelerinden programlanabilir bilgisayarlara geçişteki bu teorik adımı pratikte gerçekleştiren ise atlantiğin öte yanında, İngiltere'de genç bir matematiksel fizikçi, Maurice Wilkes olacaktı. Devasa bütçeli amerikan projelerinin aksine Wilkes ilkesel sorunları çözen, ama bellek kapasitesi sınırlı bir makine hedeflemişti. Böylece ENIAC'taki 18.000 vakum tüpe göre çok daha az malzeme gerekecekti. Wilkes mütevazi bir bütçesi olan projesine üniversitesi Cambridge'ten destek almakta zorlanmadı. 1946 sonunda başladığı işi 1949 başlarında başarıyla bitirdi (resim 8.5 ve 8.6) ve makineye EDSAC adını verdi (Electronic Delay Storage Automatic Calculator, Elektronik Gecikmeli Bellekli Otomatik Hesaplayıcı). 6 Mayıs 1949'da Wilkes'in öğrencilerinin yazdığı ilk 'program' üzerinde delikler olan bir kağıt şerit halinde makineye yüklendi. Program tamsayıların karelerini hesaplayacaktı. yarım dakika sonra makinenin yazıcısından doğru rakamlar akmaya başladı (Campbell-Kelly and Aspray, 2004). Bu yıl (2010) Kasım ayında yitirdiğimiz Maurice Wilkes'ın yaptığı bu makine böylece ilk gerçekten programlanabilir bilgisayar oldu.

Bilgisayarın buraya kadar olan hikayesinde dikkate değer bazı noktalar var. Bunlardan biri teori ve pratiğin birbirinden kopuk ve farklı yerlerden geliyor olması. Bu dönemde en temel teorik ilerlemeler Avrupa'da ortaya çıkıyor. Öte yanda ABD'deki projeler kısa vadede pratik sonuç almaya odaklanmış ve teorik esasları gözardı etmiş görünüyor. Sonraki yıllarda da bir yandan Avrupa'nın teorik katkıları sürerken bir yandan Amerikalılar bu teknolojiyi sanayi haline dönüştüreceklerdir. Bir başka dikkat çekici nokta ise en önemli yeniliklerin en pahalı projelerden değil daha küçük olanlardan gelmesi. Bu yeniliklerin çoğunun farklı



Şekil 8.5: İlk programlanabilir bilgisayar EDSAC'ın arka yüzü ve bağlantıları, 1949 (kaynak Cambridge Üniversitesi, Bilgisayar laboratuarı)

yerlerde tekrar tekrar icat edilmeleri gerekmişti. Büyük projelerin çoğu genel amaçlı bir hesaplama makinesi değil, belirli bir işe yarayan bir makine arayışındaydılar ve hem devlet kurumları hem büyük şirketler genel amaçlı makinelerin geleceği olduğuna inanmaz bir miyopluk içinde görünüyordu.

Bu teknolojinin ne yönde gelişeceğine dair miyopluk sık sık su yüzüne çıkacaktır. Savaştan sonra 1948'de ENIAC'ın mimarları Eckert ve Maucly bu teknolojiyi ticari bir ürün haline getirmek için devletten yardım istediklerinde, mark I'in yaratıcısı Aiken karşı çıktı. Aiken'e göre bütün ABD'de taş çatlasa 5-6 bilgisayar satılabilecek bir pazar vardı. Bu makine ticari hayatta kullanılamazdı (Ceruzzi, 2003). Yine aynı sıralar IBM yöneticisi Thomas Watson tüm DÜnya bilgisayar pazarının birkaç düzineyi geçmeyeceğini düşünüyordu (Campbell-Kelly and Aspray, 2004). Savaş sonrası yıllarda ABD ve ingiltere'de sınırlı sayıda ve görece küçük şirket bu teknolojiyi ticari hayata sokmak için girişimde bulundu. İlk ticari bilgisayar İngiltere'de yapılmasına rağmen (Feranti Mark I) İngiltere'deki bu hareketlilik çok hızlı sönecekti. Buna karşın 1951'den itibaren IBM'in bu teknolojinin ticari po-

Şekil 8.6: EDSAC önden görünüm ve yanında duran Maurice Wilkes, 1949 (kaynak Cambridge Üniversitesi, Bilgisayar laboratuarı)

tansiyelini daha ciddiye almaya başlamasıyla bilgisayar teknolojisi hemen tamamıyla ABD'de gelişeceği bir döneme giriyordu.

Bu başlangıç yıllarına baktığımızda ne kadar heyecan verici görünürse görünsün yeni teknolojilerin şüphe ve tutuculukla karşılandığını görüyoruz. Büyük şirketler karlı bir piyasa oluştuğunu görene kadar (bazıları diğerlerinden daha erken görse de) bilgisayara yatırım yapmakta tereddüt etmişlerdir. Teknolojinin potansiyel kullanıcısı olan şirketler de bu yeni makineleri kullanmakta tereddüt edecek, ve herkes bir diğerinin ilk adımı atmasını bekleyecektir. Bu süreçte yenilikçi çalışmaların finansmanı büyük ölçüde devletler tarafından karşılanmıştır. Ayrıca devletlerin bunu yaparken nadiren akılcı ve uzun vadeli düşünerek davrandığını görüyoruz. Bu projelerin çoğu hantal, pahalı, ve teorik temelden yoksundu. Wilkes ve von Neumann gibi öngörülü bilim insanlarının doğru yerde ve doğru zamanda bulunup projelerine mali destek almaları genel değil istisnai bir durumdu. Alan Turing'in kaderi ise bunun en talihsiz örneği olacaktı. Belki de bilgisayar teknolojisinin gelmiş geçmiş en önemli dehası olan Turing savaştan sonra eşcinselliği yüzünden devlet kurumlarından destek görmeyecek ve bilgisayarla ilgili yürütülen önemli projelerden dışlanacaktı. 1952'de eşcinsellik suçlamasıyla hüküm giydi ve zorla hormon tedavisine sokuldu. 1954'te intihar ettiğinde 42 yaşındaydı ve tasarladığı bilgisayarla canlıların gelişimini araştırıyordu (Hodges, 2000).

Bilgisayar ile ne yapılabileceğine dair görüşlerin değişmesi zaman alacaktı. Bu aletin askeri savunma dışında iş hayatında da kullanılabileceği düşüncesi ancak 1950'lerin ortalarında yükselmeye başladı. Bu yıllara gelindiğinde ABD'de ordu projeleriyle serpilip büyümüş bir elektronik cihazlar sanayi oluşmuştu. Ayrıca IBM, Remington, NCR gibi ofis makineleri ve delikli kart teknolojisi üreten firmalar vardı Campbell-Kelly and Aspray (2004). IBM bu işin geleceğiyle ilgili tereddütü olmasına rağmen bir yandan araştırmaya ve eski mekanik makinelerini yeni elektronik parçalarla iyileştirmek yönünde denemeler yapmaya devam ediyordu.

IBM'in 1940'lar boyunca yaptığı makineler pek heyecan verici değildi. Ancak 1948-9'da CPC (Card Programmed Calculator, Kartla Programlanan Hesaplayıcı) adı altında yaptıkları cihaz piyasada kayda değer bir ilgi gördü ve firmanın bu teknolojiye yatırımlarını da hızlandırdı. Yine de IBM'in çekingenliği bu ilk yıllarda Eckert ve Maucly'nin kurdukları firma ve ürettikleri

UNIVAC adlı bilgisayarın piyasada hakim olmasıyla sonuçlandı. 1952 ABD başkanlık seçimleri sırasında UNIVAC kullanılarak seçim sonuçları çok yaklaşık bir şekilde tahmin edildi ve bu olayla bilgisayarın savunma sanayi dışında da uygulaması olabileceği gerçeği medyada yankılandı.

Tereddüt ve gecikmelere rağmen IBM kısa zamanda arayı kapatacak ve piyasayı ele geçirecekti. İlk yaptıkları bilgisayarlar çok pahalıya (1 milyon dolar) malolsa da 1953'ten itibaren IBM 702 modeliyle UNIVAC'la rekabet edebilir hale geldi. Bu projede pahalı elektronik parçaların bir kısmı İngiltere'de geliştirilen daha ucuz vakum tüpleriyle değiştirilmişti.

Bu dönemde yapılan makinelerin hepsi devasa büyüklükte ve çok sorunluydular (yanan tüplerin değiştirilmesi için sürekli işe ara vermek gerekiyordu). Bu yüzden onları dinozorlara benzetebiliriz. Oldukça da pahalı olan bu makineleri ancak hızlı hesaplama gücünün maliyetini ödemeye hazır çok büyük şirketler (örneğin askeri uçak veya roket üreticileri) alıyordu. 1950'lerin sonunda IBM ticari bilgisayar piyasasını ele geçirmiş olmasına rağmen bu pazar IBM gelirlerinin çok küçük bir bölümünü oluşturuyordu. Bilgisayarın daha fazla şirket ofisine girmesi için dinozorların biraz evrimleşip küçülmesi ve ucuzlaması gerekecekti.

İlki 1959'da tamamlanan IBM 1201 modeli pazarın hızla büyümeye başlamasına yolaçacaktı. Aylık kirası 2500 dolar gibi makul bir düzeyde olan bu makinelerden 12.000 tane yapıldı. Bu başarının en önemli sebebi makinenin hesaplama gücü değil hızlı çalışan bir yazıcısı olmasıydı (Campbell-Kelly and Aspray, 2004). IBM yeni makinelerin cazibesini arttırmak için kendi parasıyla geliştirdiği programları müşterilerine bedavaya veriyordu. Müşterilerin önceliği işleri fazla değiştirmeden, yeni şeyler öğrenmek ve fazla para harcamak zorunda kalmadan iş süreçlerini iyileştirmekti. Bu durumda fiyat ve kullanım kolaylığı belirleyiciydi. O yüzden sonuçları kendisi yazan bir yazıcıyla donanmıi cihaz ve iş ihtiyaçlarını karşılayan hazır programlar çok cazipti.

Yine bu dönemin önemli bir gelişmesi transistör'ün bulunuşuydu. Stanford'da (Kaliforniya) bir araştırmacı olan Shockley'in 1957'de bulduğu bu elektronik parça 1960'ların başından itibaren sürekli patlayan ve yavaş çalışan vakum tüplerin yerini alacak, bilgisayarın hafıza ve işlem kapasitesinin hızla artmasında önemli rol oynayacaktı Castilla et al. (2000). Shockley'in Californiya'da üniversitenin uçsuz bucaksız arazisinde, bugün Silikon Vadisi

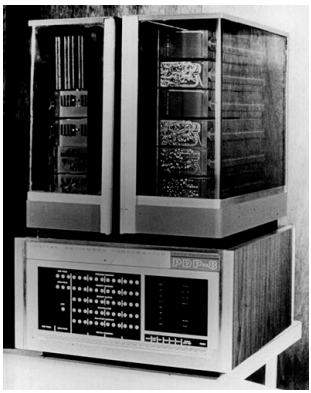
denilen yerde, kurduğu şirket defalarca isim ve kadro değiştirerek şimdi Intel adıyla bildiğimiz mikroelektronik devine dönüşecekti Castilla et al. (2000).

Böylece 1960'lar bilgisayarın ticari pazarının büyümesiyle beraber yavaş yavaş bu sanayide uzmanlaşmaya sahne olacaktı. Pazarın işleyişi tamamına yakını IBM'de üretilen makinelerin yerine elektronik parçaların büyük kısmının Intel gibi şirketlerden tedarik edildiği bir pazar yapısına doğru yavaş yavaş evrilecekti. Çok daha sonraları bu uzmanlaşmaya uzakdoğudaki üreticiler de dahil olacaktır.

1960'lardaki hızlı büyüme ve çeşitlenmeye eşlik eden bir sıkıntı vardı. Bilgisayar modelleri değiştikçe müşterileri memnun etmek için varolan programların hepsinin yeni makine modelleri için yeniden yazılması gerekiyordu. Başlangıçta işin içinde küçük bir detay gibi görünen yazılımla ilgili bu iş çok pahalıya patlamaya başlamıştı. Bu sıkıntının hızla büyüyeceğini gören IBM, modeller arasında program uyumluluğunu sağlamak için Sistem 360 isimli bir projeye girişti. Bu IBM'in muazzam paralar yatırdığı ve çok ter döktüğü b,r proje olacaktı (Campbell-Kelly and Aspray, 2004; Brooks, 1995). İlk sonuçları 1963'te alınan proje 1970'lere kadar sürdü ve esasen bilgisayar yazılımlarında 'işletim sistemi' adı verilen katmanlaşmanın ilk örneklerindendi.

Bu süreçte üniversitelerin rolü tarihçelerde oldukça geri plandadır. Ancak özellikle işletim sistemleri ve programlama dilleri konusunda hem ABD hem de Avrupa'da yapılan bir yandan teorik bir yandan da uygulamalı çalışmalar oldukça önemliydi. Bu çalışmalar piyasa şirketlerinin ön plana çıktığı makinelerin iyileştirilmesi ve geliştirilmesi çalışmalarının önemli bir tamamlayıcısıydı (DiBona et al., 1999). Akademinin katkısı yazılımın ön plana çıktığı 1980'lerde daha da görünür olacaktı.

Bu arada pazarın büyümesi IBM dışındak girişimcileri de harekete geçirmişti. Bunların arasında en başarılılarından biri DEC şirketi ve ürettiği PDP serisi bilgisayarlardı. bu rekabetin sonucunda dönemin dinozorları birazcık olsun küçülecek, ucuzlayacak, ve hızlanacaktı. 1965'te çıkan PDP-8 modeli (resim 8.7) küçük bir odaya rahatça sığıyor ve saniyede 35.000 toplama işlemi yapabiliyordu. Fiyatı da sadece 18.000 dolardı (Ceruzzi, 2003). Piyasada ortaya çıkan büyük IBM ve küçük yenilikçi şirketler dengesi 1980'lere kadar sürecek, ve IBM'in yedi küçük rakibine yedi cüceler ismi takılacaktı (Ceruzzi, 2003; Campbell-



Şekil 8.7: DEC firmasının 1965'te piyasaya sürdüğü PDP-8 modeli bilgisayar (kaynak Computer History Museum)

Kelly and Aspray, 2004). Bu dönemde bilgisayar teknolojisinde hiçbiri esasa dair olmasa da herbiri önemli bir dizi ilerleme olacaktır. Bunlar bellek kapasitesi ve hızının artması, transistör teknolojisinin gelişip hızlanması ve küçülmesi, manyetik bantların ve manyetik disklerin ortaya çıkması gibi yeniliklerdi. Ayrıca artık bu parçalar daha kaliteliydi ve daha az arıza yapıyordu.

1960'ların sonunda ABD uzay programlarının gelişmesi de bilgisayarlara talebi arttıran faktörlerden biri oalcaktı. Ancak 1970'lerin ortalarında kadar değişmeyen şey bilgisayarların hala dinozor büyüklüğünde olması ve ancak orta-büyük ölçekte şirketler, kimi devlet ofisleri ve önde gelen üniversitelerde kullanılmasıydı. Ancak yavaş ta olsa bilgisayar denilen ürün ailesinde evrimsel bir çeşitlilik ortaya çıkıyordu. IBM büyük müşterilere hizmet verirken DEC gibi firmaların daha küçük bilgisayar modelleri orta büyüklükte şirketler, üniversitelerin mühendislik bölümleri gibi görece daha mütevazi bütçeli müşterilere hitap ediyordu. Bu yüzden DEC gibi firmalar maliyetleri azaltmak ve makineleri daha da küçültmek için günümüz entegre devrelerinin (mikroçip) öncülü olan teknikler geliştirmeye başlamışlardı (Ceruzzi, 2003). Zamanla bilgisayar sektöründe uzmanlaşma artacak ve Texas Instruments gibi şirketler entegre devre üretme işini üstleneceklerdi. Bu tür küçük bilgisayar parçaları ayrıca roket ve uydu yapımında da kullanım alanı buluyordu. Bu arada

üniversitelerde bilgisayar bilimleri bölümleri oluşmaya başlamış ve buralarda yapılan çalışmalar çeşitlenen bilgisayar kullanımının ihtiyaç duyduğu yazılım teknolojilerini ve işgücünü karşılamakta önemli bir rol üstlenmeye başlamışlardı.

8.2 1980-1990: Kişisel bilgisayarlar ve yazılımın yükselişi

1970'lerin sonuna doğru bilgisayar tüm dünyada büyük kurumlarda kullanılır olmuştu. Bu dönemde Japonya başta olmak üzere uzakdoğu firmaları da elektronik pazarına girdiler. Elektronik devre üreten Hitachi gibi firmaların yanısıra, örneğin Casio ve Sharp gibi firmalar kısa zamanda hesap makinesi gibi küçük elektronik ofis cihazları sektöründe egemen hale gelmişlerdi. Bilgisayar parçaları o kadar ulaşılır hale geldi ki 1970'lerin sonunda Standord üniversitesinde hevesli öğrenciler kendi bilgisayarlarını yaptıkları bir kulüp bile kurmuşlardı (Ceruzzi, 2003). Küçülen ve becerileri gelişen entegre devreler birçok olasılığın kapısını aralıyordu.

Intel'in 1975'te ürettiği ve neredeyse bütün bilgisayar işlevlerini bir çipe sığdıran 8080 entegre devre çipi ile beraber Altair gibi firmalar 400 dolar civarında fiyatı olan küçük sayılabilecek (bir masanın yarısını kaplasa da) bilgisayarlar üretmeye başladılar. Bu durum dinozorların sonunun habercisiydi. Altair bilgisayarlarında kullanılacak programlama dili olarak Microsoft adlı küçük bir firmanın sağladığı BASIC dilini tercih etmişti ve bu Microsoft'un ilk piyasa başarısı olacaktı. Firma zamanın kimi üniversite hocalarının problemli ve hatalara açık tasarımı yüzünden öğretmeyi reddettikleri BASIC programlama dili gibi birçok tartışmalı yazılım ürününü piyasaya sürecekti (Ceruzzi, 2003).

1980'ler başlarken bilgisayarların evlere ve ofislere gireceği belli olmuştu, ama kişisel bilgisayar adı verilecek olan bu makinelerin tasarımının ne yönde gelişeceği belirsizdi. Bu dönemde ortaya çıkan Apple şirketi görece pahalı kişisel bilgisayarlar satıyordu. Ama bu bilgisayar tasarımının ihtiyaca göre parça ekleyebilmek gibi bir avantajı vardı ve bu yüzden ilgi görüyordu. Commodore şirketinin bilgisayarları çok daha ucuz ama bu özellikten yoksundu; gelişmiş bir hesap makinesi görüntüsündeydi. Tandy markasının bilgisayarları da ucuzdu, ve televizyona bağlanarak kullanılabildiği için maliyetten tasarruf ediyordu. Sonuçta 400-1400 dolar aralığındaki bu bilgisayarlar hemen her büyüklükte

Şekil 8.8: Sinclair ZX Spectrum marka oyun bilgisayarı, 1982 (Kaynak: Bill Bertram)

şirketin, ve konuya meraklı kişilerin alıp kullanabileceği bir maliyete sahipti. O dönemde piyasaya çıkan ve finansal analizlerin bu küçük, kişisel bilgisayarlarda yapılabilmesini sağlayan VisiCalc isimli program birçok küçük-orta büyüklükte işyeri sahibinin bilgisayarlarla ilgilenmesine yolaçtı Campbell-Kelly and Aspray (2004). Kişisel bilgisayar adı altında oluşan makine ve yazılım çeşitliliği üniversitelerden işyerlerine, ve heveslilere kadar herkese hitap eder hale geliyordu. 1982'de babamın bana aldığı Sinclair ZX Spectrum marka oyun bilgisayarı 100 İngiliz Sterlini fiyata sahipti. 16K hafıza ve 8-bitlik bir işlemciye sahip, kitap büyüklüğünde, klavyesi üstünde bir aletti (resim 8.8). Televizyona bağlanarak kullanılabiliyor, oyun programlarını walkman benzeri bir ek cihazla kasete/kasetten yükleyebiliyordu. Üzerinde yüklü

BASIC programlama dilini kullanarak program yazmak benim için oyunlardan çok daha ilgi çekici olmuştu.

Bilgisayarın bu kadar çeşitli kullanım alanına girmesi kısa bir süre içinde yazılımın önem kazanmasına yolaçtı; çünkü aynı makine farklı işleri yapabilir ama bunun için farklı programlar gerekirdi. Önceleri IBM gibi firmalar yazılımı donanımın yanında veriveriyorlardı. Aynı donanım üretiminde firmaların farklı alanlarda uzmanlaşmasında yaşandığı gibi, yazılım konusunda da bir uzmanlaşma kapıdaydı. Ancak herkes yazılımın birdenbire bir problem (ve fırsat) haline gelmesine hazırlıksız yakalanmış gibiydi. Sorunun bir kısmını üniversitelerde yürüyen çalışmalar çözecekti. Kamu kurumları (AT&T gibi) ve üniversitelerde geliştirilen UNIX işletim sistemi, ALGOL programlama dili gibi teknolojiler, en azından yazılan bir programın makine özellikleri ve modeli değişse de, veya başka makineler üzerinde değişiklik gerekmeden çalışmasına izin veriyordu. Bir kez programlama dilinin kendisi yeni model makineye uydurulduğunda o dilde yazılmış bütün programlar da uygun şekilde çalışabiliyordu. IBM'in geliştirdiği FORTRAN, ve devlet finansmanıyla Grace Hopper tarafından geliştirilen COBOL ile birlikte bir de programlama dilleri rekabeti ortaya çıkmıştı.

yazılım sorununun nasıl bir ortak sisteme oturacağı konusunda bir kargaşa sürüp giderken uzun zamandır büyüyen kişisel bilgisayar piyasasında geri planda kalmış olan IBM ilginç bir hamleyle atağa geçecek ve oyunu yeniden şekillendirecekti. IBM o zamana kadar hem üretim hem satış konusunda dikey entegrasyon modeline sahipti. Aynı 1920'lerde Ford araba fabrikalarında olduğu gibi IBM'de de bilgisayarın hemen bütün parçaları IBM'in dev tesislerinde üretiliyordu. Ayrıca yazılım da IBM içinde üretiliyordu. Ayrıca şirket yalnızca kendi pazarlama ağı üzerinden müşterilere ulaşıyordu. Bu model devasa bir iş hacmi olan şirket için uzun yıllar başarıyla işlemişti. Ancak hızla büyüyen, çeşitlenen, ve ABD dışında küresel pazarlara da yayılan bilgisayar pazarı karşısında IBM'in modeli hantal kalıyor ve çöküyordu. 1980'lerin başında şirket yöneticileri son derece radikal bir kararla bütün çalışma modelini değiştirdiler (Campbell-Kelly and Aspray, 2004). IBM kişisel bilgisayar pazarına girecekti. Üstelik bunu diğer şirketlerin yaptığı gibi piyasada varolan, kendisinin üretmediği elektronik devre ve parçaları kullanarak yapacaktı. Bu ürünleri kendi pazarlama ağı dışındaki kanallara, parekendecilere verecekti. Ayrıca yazılımlarının da bir kısmını

dışarıya yaptıracaktı.

IBM daha da ileriye giderek başka firmaların ürettiği parçaların kendi bilgisayarlarına eklenebilmesi için bazı standartlar geliştirdi ve bunları herkesle paylaştı. ISA (Industry Standard Architecture, Endüstri Standardı Mimari) adını verdikleri bu standart sayesinde disk sürücü, modem gibi parçalar bilgisayarın boş girişlerine takılabiliyordu. Bu durum bütün piyasayı etkileyecekti. Uzakdoğu'daki elektronik üreticilerini de içine alan bir IBM PC uyumlu donanımlar piyasası ortaya çıktı (Langlois, 1990; Sanchez and Mahoney, 1996). Bu standardı benimseyen diğer kişisel bilgisayar şirketleri de ürünlerini PC-uyumlu olarak tanıtmaya girişti. IBM'in PC'leri için tercih ettiği entegre devre üreticisi Intel hızla büyümeye başladı. Son olarak IBM'in PC'lerine uygun yazılım üretmesi için anlaştığı Microsoft şirketi bu noktadan sonra hızla muazzam bir yazılım devine dönüşme yoluna girdi.

1980'lerin ortalarında IBM'in çektiği hizaya girmeyen bir tek şirket kalmıştı: Apple. Şirketin yöneticisi Steve Jobbs IBM'ler rekabet etmek için ucuz ve standart donanım kullanmak yerine sistemine daha iyi yazılım üretme ve kullanıcıya daha özgün bir deneyim sunma stratejisini seçti. Apple gelecekte başka krizler yaşamasına rağmen genel olarak bu stratejiyi sürdürecek ve kişisel bilgisayar pazarında özgün bir alternatif olmaya devam edecekti.

1980'lerde yazılım problemi bilgisayar pazarındaki taşların yerinden oynamasına sebep oldu. Böyle dönemlerde ve piyasa koşullarında küçük şirketler için önemli şanslar ortaya çıktığını görüyoruz. Koşulların hızla değiştiği 80'lerde çok sayıda küçük yazılım şirketi ortaya çıktı. Bu şirketler piyasa devlerine göre daha yaratıcı, daha yenilikçiydiler. Ancak böyle dönemlerde hep olduğu gii kısa bir süre sonra bu çeşitlilikte evrimsel bir eleme olacak ve bu oyuncuların pek azı hayatta kalacaktı. Birkaç teknoloji etrafında standartlaşma piyasanın işleyişi için kaçınılmazdır. Bu dönemde IBM'in PC hamlesi belirleyici olmuştur. IBM'in yazılım tedarikçisi olarak Microsoft'u seçmesi kısa bir süre sonra bu şirketin işletim sisteminin (DOS) standart haline gelmesine yolacaktı. Microsoft bu dönemde işletim sistemi dışındaki yazılım pazarlarında da (örneğin kelime işlemci, hesap tablosu yazılımları) birçok ürünle yelpazesini geliştirmeye çalıştı ancak varolan diğer ürünler karşısında başarısız oldu (Campbell-Kelly and Aspray, 2004). bu başarısızlıklara rağmen satılan her IBM PC'den akan para ş,rketin hayatta kalmasına ve hızla büyümesine yetiyordu.

IBM ise PC satışlarıyla iyiye giden işlerin keyfini çıkartıyordu ve ancak yıllar sonra kendi yarattığı bu deve karşı rekabet etmek için yeni stratejiler bulma ihtiyacı hissedecekti. Küçüklerden çoğunun ayıklanmasından sonra PC piyasasında bu şirketler dışında kayda değer olarak sadece müşterilerine daha farklı bir kullanım deneyimi sunan Apple kalmıştı. PC dışında kalan büyük ve orta boy bilgisayarlar ise biraz daha küçüldüler, ve pazardaki payları azalmış olsa da büyük kurumlarda varlıklarını sürdürdüler.

8.3 1990'lar: İnternet/web ve bilgisayarla iletişim

80'lerde bilgisayar birçok işyerine ve bir kısım evlere girdi. Bu arada modem gibi iletişim cihazları gelişmeye ve e-posta kullanımı yayılmaya başlamıştı. E-posta'nın normal posta, fax, ve telex gibi daha eski iletişim teknolojilerinin yerini almaya başlaması kayda değer bir olguydu. Ancak bu durum daha çok sayılı işyeriyle ve üniversitelerle sınırlıydı ve bilgisayar bir iletişim aracından çok metin yazma, tablo tutma gibi işleri kolaylaştıran bir ofis makinesiydi.

İnternet teknolojisinin uzun yıllara yayılan gelişiminden daha önceki bölümlerde bahsetmiştik. Ancak aynı bilgisayarın ilk yıllarında olduğu gibi İnternet te başlarda kısıtlı bir kullanıma sahipti. 1990'ların başında ortaya çıkan bir yenilik bu durumu hızla değiştirecektir: World Wide Web (www). bu kez değişimin konusu ne donanım ne de yazılımdı. Bu değişim insanların ve şirketlerin bilgisayarı birbirleriyle iletişim, bilgiyi paylaşma ve ona erişme aracı olarak kullanmaya başlamasıyla ilgiliydi (Castells, 2005, 2001).

Zürih'te bir fizik araştırma merkezinde çalışan Tim Berners Lee 90'ların başında bugün web sayfası dediğimiz HTML metin standardını ve bu metinleri bilgisayardan bilgisayara İnternet üzerinden aktarmak için gerekli HTTP iletişim standardını geliştirdi. Bunu yapmaktaki amacı fizikçilerin araştırma raporlarını ve yazdıkları makaleleri kolayca paylaşabilmesiydi. Çok kısa bir süre sonra herkesin böyle birşeye ihtiyacı olduğu ortaya çıkacaktı. Microsoft Windows'un yaygınlaşmasıyla beraber PC'lerde grafik arayüz standart haline gelmişti, ve HTML sayfaları bu arayüzde son derece zengin bir okuma deneyimi sunuyordu. Ayrıca fare ile bağlantıları tıklayarak kolayca başka metinlere, hatta dünyanın başka bir yerindeki sunucularda bulunan metinlere geçilebiliyordu. Elektronik ortamdaki metinlerin birbirine bağlanarak oluşturduğu

ağa istinaden bu yeni teknolojiye WWW (World Wide Web, Dünya çapında Ağ) adı verildi.

Web'in ortaya çıkışıyla İnternet artık bilgisayarları değil dünyanın bilgisini birleştiren bir teknolojiye dönüşüyordu. Metinleri heskesin ulaşabileceği bir sunucuya koymak bunları bütün muhataplara teker teker e-posta ile dağıtmaktan, ve sonra metinler güncellenirse aynı şeyleri tekrarlamaktan çok daha pratikti. Şirketler bu şekilde ürün ve hizmetlerinin tanıtımını müşterilerine ulaştırabilirdi. Bilim camiası araştırmalarını bu şekilde paylaşabilirdi. Dernekler, hobi grupları bilgileri bu yoldan paylaşabilirdi. Artık İnternet sadece bir iletişim aracı değil, devasa bir sanal bilgi ağına giriş kapısıydı. Dünya üzerindeki bilgilerin hepsine anında ulaşılabilen bu yeni sanal dünyaya uzaklık kavramını altüst etmesine atıfla siber-uzay adı verilmişti.

90'lı yıllar boyunca önce büyüklü küçüklü işyerleri, okullar, sonra evler İnternet'e bağlandı. Bu dönemde ne donanım ne de yazılım çok köklü bir değişiklik geçirmedi. Ancak web PC'lere talebi arttırdı. Ayrıca büyük çapta İnternet altyapısı yatırımları yapılıyordu. Evlere İnternet bağlantısı ulaştıran şirketler ortaya çıktı. Ayrıca herkesin mütevazi bir ücretle İnternet'te bir site sahibi olmasını sağlayan başka şirketler vardı. İnternet bağlantısının yaygınlaşmasıyla e-posta kullanımı da hızla yaygınlaşacaktı. Yazılım dünyasındaki kaydadeğer bir değişiklik web sayfalarının kullanımını sağlayan browser/tarayıcı yazılımlarına duyulan ihtiyaçtı. Donanım tarafındaki bir değişiklik ise İnternet bağlantısını sağlayan modem gibi parçaların PC'lere eklenmesi, ayrıca harici bağlantı cihazlarının ve küçük-orta kurumlar için altyapı cihazlarının bir pazar haline gelmesiydi. Bu yeni pazar alanlarının hepsinde yine çok sayıda küçük şirket ortaya çıkacak ve kısa bir süre sonra pek azı hayatta kalacaktı. Sonuçta son kullanıcı ürünleri pazarında IBM, Apple, Microsoft üçlemesi devam ediyordu.

Buna karşılık hızla büyüyen bu küresel bilgi alanı yeni ihtiyaçlar ve dolayısıyla yeni fırsatlar ortaya çıkaracaktı. Bu ihtiyaçlardan belki en önemlisi arama motorlarıdır. İlk aram motorlarından biri olan Yahoo 90'ların ortalarında web'deki önemli adreslerin listelendiği bir fihrist olarak başladı. Ancak web'deki içeriğin büyük bir hızla artması arama motoru gibi daha kullanıcı dostu bir teknoloji gerektiriyordu. yahoo'dan birkaç yıl sonra sektöre giren Google arama motoru ise sektörde büyük bir hızla yükselecekti.

Daha önceki değişim dönemlerinde gördüğümüz gibi bu dönemde de yeni ve küçük şirketler birçok yenilikçi çözüm ortaya atmış ve ancak birkaç tanesi ayakta kalıp yeni oluşan pazarlarda pay sahibi olmuştur. 90'lar ilerlerken arama motorları herkesin web'e giriş kapısı haline gelivermişti ve müşterilerine ulaşmak için yarışan şirketlerin verdiği reklamlarla çok hızlı bir şekilde büyüyen bir pazar olmuştu. Web sitelerinin sayısı 1993'te 130 iken 1997'de 650.000'e çıkmıştı[12] ve kullanıcı sayısı da buna paralel bir artış gösteriyordu. Ayrıca ticari amaçlı özel şirket sitelerinin bu rakamdaki payı hızla artıyordu. 90'ların sonuna doğru web bilgi paylaşım ve bilgiye erişim aracı olma yolunda hızla ilerlemişti ve bir alışveriş kanalına dönüşmeye başlıyordu.

Bu dönemin kolay gözden kaçan gelişmelerinden biri İnternet kullanımının bilgisayar programcıları camiasındaki etkileridir. Bilgisayar programları dijital şeylerdir ve İnternet bağlantısı üzerinden kolayca bir yerden başka bir yere kopyalanabilirler. Böylece örneğin biyolog veya mimarlardan farklı olarak bilgisayar programcıları üzerinde çalıştıkları şeyleri kolayca İnternet üzerinden paylaşabilir ve uzaktan birlikte çalışabilitlerdi. Bu açılık hem meslek camiasında hem de yazılım sektörünün yapısında ciddi değişikliklere sebep olma potansiyeli taşıyordu. Örneğin web sitelerinin çoğunun çalışması için kullanılan Apache adlı web sunucusu programı bu yöntemle, dünyanın farklı yerlerinden programcıların katılımıyla geliştiriliyordu. Böyle bir durumda yazılımın kimin malı olduğunu söylemek çok zordu. Bu yüzden de bu yöntemi kullanan programcılar yazılımlarına ticari programların tam tersi olarak kamu malı olduğuna dair bir ibare (lisans) koyuyorlardı. Bu şekilde bir yazılım kamu malı oluyordu ve onu satmak sözkonusu olamazdı. Öte yandan başarılı bir yazılım projesinde yeralan programcılar bu şöhreti iş imkanına dönüştürebiliyorlardı (Kogut and Metiu, 2001).

Yazılım üreten şirketler ilk yıllarda bu tür kamu lisansı taşıyan yazılımlara mesafeliydiler. Bunlardan nasıl para kazanılacağı belli değildi. Ancak 90'ların sonuna doğru meslek camiası ile özel şirketlerin böyle kamusal lisanslar çevresinde inovasyon işbirliği yapmaları alışıldık birşey haline gelmeye başlayacak ve yazılım inovasyonunda ciddi bir verimlilik yaratacaktı (Gençer, 2010; West, 2003). Öyle ki IBM gibi şirketler kendi malları olmayan bu tür teknolojilere milyarlarca dolar yardım yapar hale geldiler (Capek et al., 2005). Sonuç olarak herkes gibi bu şirketler de

o yazılımlardan yararlanabiliyordu. Bunları doğrudan satmak mümkün olmasa da işbirliğiyle ortaya çıkan yenilikçilik hızı tüm katılımcılar için bir pazar avantajı sağlıyordu.

Açık kaynak olarak ta adlandırılan bu yazılımların 90'lı yıllarda en ünlüsü linux işletim sistemi oldu. 80'lerde yazılımın ticarileşmesine bir tepki olarak ABD'de ortaya çıkan GNU hareketi aslında kamu kurumlarında üretildiği halde şirketlerce sahiplenen UNIX işletim sistemini kamusal alanda tutmak için kamu lisansı kavramını ortaya atmıştı. 91'de Finlandiya'lı bir bilgisayar öğrencisi olan Linus Torvalds'sın başlattığı Linux projesi bu açık kaynak yazılım birikiminin üzerinde duruyordu ve kısa zamanda yüzlerce başka öğrenci ve daha sonraları şirketlerin desteğiyle, tamamen sanal ortamda hızla gelişecektir. İlk kez PC işletim sistemi sektöründe Microsoft'a bir rakip ortya çıkıyordu. Üstelik bir ofisi bile yoktu! 2000'lerin başında gelindiğinde Linux, Microsoft'un işletim sisteminin kullanılmasının zor olduğu PC dışındaki cihazlarda (İnternet altyapı cihazları, ADSL modemler, sunucular) ciddi bir paya sahip olacaktı. IBM ve HP gibi şirketler PC dışı ürünlerinde bu yazılımı kullanarak yavaş ta olsa Microsoft tekelinin ağırlığını üstlerinden atmayı deniyorlardı (Economides and Katsamakas, 2006).

8.4 2000'lerden bugüne: e-ticaret ve mobil cihazlar

Daha 90'ların sonuna doğru web sadece bilgi alışverişi aracı olmaktan öte bir noktaya gelmiş, ve yaygınlığı sayesinde her türlü şeyin alınıp satıldığı bir ticaret kanalına dönüşmeye başlamıştı. Bu yeni fırsat dönemi öyle hızlı gelişti ki ABD'de adının başına e- veya sonuna .com koyan her şirketin kapısına yatırımcılar kuyruk oluyordu. Bu dönem gerçek dünyada yapılan herşeyin bir de sanal alemde denendiği bir dönemdi: sanal iletişim, sanal ticaret, hatta sanal seks. Ancak yine bu dönemin onbinlerce e-ticaret şirketi arasından sayılı şirket yaşamına devam edebilecekti.

Bunlardan en kayda değer bir tanesi Amazon'dur. 90'ların sonunda web üzerinden kitap satışı ile işe başlayan şirket daha sonra müzik ve video satışı işine de girdi. Şirketin başarısının bir yanı kitap ve CD gibi malların uzaktan satışa uygun olmasıydı. örneğin ayakkabı ve giyecek sözkonusu olduğunda tüketiciler haklı olarak deneyerek almaya alışık oldukları bu tür malları İnternet üzerinden sipariş etmeye –en azından o yıllarda– mesafeliydiler. Ancak Amazon'un başarısındaki en önemli faktör farklı pazarlama

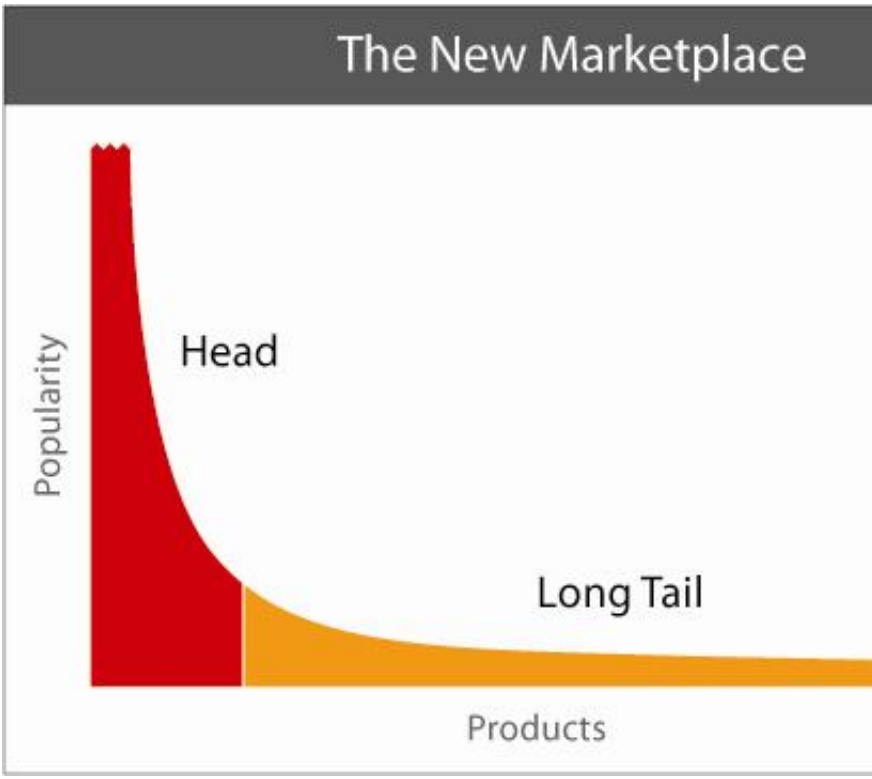
yaklaşımıydı. En fazla 30-40 bşn farklı kitap bulundurabilen büyük kitabevlerinden farklı olarak Amazon yüzbinlerce (zamanla milyonlarca) ürün sunabiliyordu. Bunu da kendi ürün stoklarını tutmak yerine çok sayıda üretici ile müşteriler arasında bir aracı gibi durarak yapıyordu.

Chris Anderson 'Uzun Kuyruk' adlı kitabında e-ticaretin normal ticaretten farklılıklarını canlı bir dille anlatır (Anderson, 2006). Kitap ya da müzik ağazaları yer ve stok sorunu yüzünden sadece en çok aranan ürünleri bulundurabiliyorlar. Öte yandan herbirinin az müşterisi olan çok fazla sayıda başka ürün var. herbirinin iş potansiyeli az da olsa bu ürünlerin karşılığı olan iş hacmi toplamda popüler ürünlere yakın, hatta daha fazla olabilir. Yani şekil 8.9'de Anderson'dan aktardığımız ürün/popülerlik grafiğinin küçülerek devam eden 'kuyruk' kısmı baştaki az sayıda popüler ürünün iş hacmiyle karşılaştırılabilir bir alan barındırmaktadır. Amazon iş modeli bu potansiyeli gerçeğe çevirmeye çok uygundu.

2000'ler boyunca değişik sektörlerde başarılı e-ticaret modelleri ortaya çıktı. Bunların hemen hepsi Amazon gibi ürün çeşitliliği esasına dayanarak e- olmayan yerleşik rakiplerinden farklılaşıyorlar. Örneğin e-bay ikinci el ürünlerde, Expedia seyahat ve tatil ürünlerinde aynı modeli kullanmaktadır. Türkiye'de de HepsiBurada, Sahibinden, Markafoni gibi e-ticaret girişimleri oldukça başarılı olmuştur.

E-ticaret'in yükselişiyle ortaya çıkan değişim daha öncekilerden farklıdır. Bilgisayarın ticarileşmesi (60-70'ler), kişisel bilgisayarların çıkışı (70-80'ler), yazılımın yükselişi (80'ler), ve web'in yaygınlaşması (90'lar) dönemleri bilgi ve iletişim teknolojisinin temel taşlarının teker teker yerine oturduğu dönemlerdi. Ancak yeni binyıldaki değişim daha kalıcı ve monoton seyreder görünüyor. 90'ların sonundaki aşırı hareketli girişimcilik dönemi sönmesine rağmen e-ticaret önceki dönemlerden çok daha fazla girişimi istihdam etmektedir ve edecektir. Örneğin herkes aynı PC'yi kullanabiliyorken her sektör ve her yerelde farklı e-ticaret girişimlerine yer var gibi görünüyor. Ayrıca geçen on yıla rağmen bu alan yaratıcılığa son derece açık olmaya devam ediyor. Bir yandan konsolidasyon süreçleri devam etse ve şirketler küresel ölçekte birleşse de bu fırsat zenginliği sürüyor.

Yeni binyılın ilk yılları boyunca aslında bilgisayar teknolojisinden bağımsız gibi görünen başka bir teknoloji yaygınlaşıyor



Şekil 8.9: E-ticarette popüler olmayan ürünlerin oluşturduğu uzun kuyruktaki iş hacmi popüler ürünlere yakındır (Grafik: Chris Anderson)

ve ciddi bir değişimin motoru olmaya hazırlanıyordu: mobil cihazlar, ve özellikle cep telefonları. Bu teknoloji bir süre telefonun taşınabilir halı olarak kendi yolunda ilerledi. Bu süre boyunca mobil iletişim şebekeleri güçlensi ve kapasitesi arttı. Ayrıca küçük cihazlarda kullanılanilen mikroçipler güçlendi. Böylece yeni olasılıklar ortaya çıkıyordu. Artık bu cihazlar sayesinde her yerde her zaman İnternet'e ve web'e bağlanabiliyoruz. Dolayısıyla web'in sağladığı bilgi deryasına ve uzaktan alışverişe açılan kapılar artık sadece ev ve ofislerdeki PC'lerde değil, aynı zamanda cebimizde.

Kırk yılı aşkın bir süre önce Licklider (1968) insanların yakın bir gelecekte bilgisayarlar aracılığıyla iletişim kuracağı bir dünyayı müjdelemişti. Bugün bunu en uç noktada yaşıyoruz. Cep telefonlarımızdan neredeyse sadece uyurken ayrılıyoruz. Sesli ve görüntülü iletişim kurabiliyoruz. Daha da öte navigasyon ve coğrafi bilgi sistemleri gibi teknolojiler bize yolda yürürken ya da araba kullanırken çevremizde neler olduğu (kefeler, restoranlar, sinemalar) hakkında bilgi veriyor. Yani sadece uzaktaki insanlarla değil çevremizdeki, gözümüzün önündeki şeylerle de bu cep telefonları aracılığıyla iletişime geçiyoruz. Bugün geldiğimiz noktada doğal olarak cep/mobil teknolojiler yeni alışveriş ve bilgiye erişim aracı olarak kişisel bilgisayarların yerini almaya adaydır. Son dönemde Google gibi büyük firmaların yaptığı yatırımlar da piyasadaki beklentinin bu yönde olduğunun altını çiziyor.

Mobil teknolojilerin yükselişinin yarattığı bu etki piyasanın diğer alanlarında da bir tepki doğurmaktadır. Artık çoğumuz beğendiğimiz filmleri, dizileri dilediğimiz zaman cep telefonumuzdan veya ev ya da ofisteysek bilgisayardan izleyebiliyoruz. Hal böyle olunca özellikle genç nesiller için televizyon ve radyo oldukça yetersiz teknolojiler olarak gözükmektedir. Dolayısıyla bu ilgi kayması uzun yıllardır yerleşik bu teknolojiler çevresinde oluşmuş pazarda taşları yerinden oynatmaya başlamıştır. Buna gazete de dahildir çünkü artık haberlere İnternet veya cepten, anında ve üstelik görüntülü ve sesli olarak erişebiliyoruz. Bu yüzden geleneksel medya yeniden ilgi çekmek için yenilikler peşine düşüyor. Bu yeniliklerden biri İnternet TV (IPTV). Bu teknoloji TV'nin de etkileşimli bir mecra haline gelmesini hedefliyor.

Birbirinden bağımsız gibi görünse de son on yıldaki değişimler ortak iki dinamik çevresinde şekillenmektedir. Bunlardan biri bireyselleşme, özellikle de tüketimin bireyselleşmesi. Sadece popüler ürünleri sunabilen eski usul kitabevlerinin aksine Amazon.com

en küçük ilgi gruplarına bile istediklerini sunabiliyor. Filmleri TV'de yayınlandıkları saatte değil canımız istediği zaman, istediğimiz yerde seyrediyoruz/tüketiyoruz. Üstelik gitgide daha fazla bireysellik talep ediyoruz. Doğal olarak cebimizde bizimle her yere gelebilen mobil cihazlar bu talebe çok iyi karşılık geliyor.

İkinci bir dinamik ise kısmen üretimin de bireyselleşmesidir. Eskiden ürünlerini satacak yer bulamayan, özel ilgi gruplarına hitabeden küçük yayınevleri veya müzik prodüktörleri Amazon gibi siteler sayesinde alıcılarına ulaşabiliyorlar. hepimiz ikinci el eşyalarımızı, hatta orijinal ürünlerimizi e-Bay, Sahibinden, Gitti-Gidiyor gibi sanal pazaryerlerinde satabiliyoruz. Eğer ilginç bir videonuz varsa büyük TV kanallarını sollayıp youTube sayesinde milyonlarca kişiye, hem de kısa bir sürede ulaşabilirsiniz. İnternete koyduğunuz ev kaydı bir şakı bir anda hit olabilir. Bir programcıysanız yaptığınız programı İnternetten paylaşabilir, ve belki başka yerlerde ilgilenen meslektaşlarınızla yardımlaşarak çok popüler bir yazılıma dönüştürebilirsiniz. Tüketimdeki bireyselleşmeden gelen çeşitlilik talebi üretimdeki çeşitlenmeyle karşılanabilir. Ve Bilgisayarın bütün tarihinde tekrar tekrar gördüğümüz gibi pazarın küçük ve yeni oyuncuları her zaman büyüklerden daha fazla yeniliklere gebedir.

Notes

[1]Yakın zamanda Google firmasının ortaya attığı bir teknoloji programcılığa yabancı olanların da cep telefonları için basit programlar yazmasını sağlamayı amaçlıyor. Bu tür gelişmeler programcılığın kısıtlı bir meslek camiasının uğraşı olmaktan öteye geçip, geniş kesimleri ilgilendiren bir uğraş olmaya başladığının göstergesi.

[2]Bkz. http://www.infoq.com/news/2010/06/career-labor-statistics

[3]DeepBlue teknolojisi ve hikayesi için bkz. http://researchweb.watson.ibm.com/deepbl

[4]Matematiğin bu dalı benim becerilerimi aşıyor. Buradaki bilgilerin bir kısmını Stanford Encyclopedia of Philosophy'den derledim.

[5]bkz. http://www.seas.upenn.edu/about-seas/eniac/

[6]ENIAC projesindeki kadın programcılara ilişkin bir güncel yazı için bkz. http://www.wired.com/culture/lifestyle/news/1997/05/3711

[7]Bu kelimenin İngilizce 'ısırık' karşılığı 'bite'tan türediği sanılıyor, ancak kesin bir tarihsel not bulamadım.

[8]Orjinal tasarım notu için bkz. http://www.faqs.org/rfcs/rfc20.html

[9]Bkz. PXE teknolojisi.

[10]http://internetanniversary.cs.ucla.edu/Speakers_Bios.html

[11]Bkz. `http://tools.ietf.org/html/rfc791`

[12]Kaynak: Matthew Gray, `http://www.mit.edu/~mkgray`

Kaynakça

Abelson, H. and G. J. Sussman (1996, July). *Structure and interpretation of computer programs* (second ed.). The MIT Press.

Adleman, L. M. (1994, November). Molecular computation of solutions to combinatorial problems. *Science (New York, N.Y.) 266*(5187), 1021–1024.

Aloisio, M. (2004). The calculation of easter day, and the origin and use of the word computer. *IEEE Annals of the History of Computing 26*, 42–49.

Anderson, C. (2006, July). *The Long Tail: Why the Future of Business is Selling Less of More.* Hyperion.

Benenson, Y., B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro (2004, April). An autonomous molecular computer for logical control of gene expression. *Nature 429*(6990), 423–429.

Brooks, F. P. (1995). *The mythical man-month.* Addison Wesley Longman.

Cajori, F. (1991). *A History of Mathematics.* Chelsea, NY.

Campbell-Kelly, M. and W. Aspray (2004, July). *Computer: A History of the Information Machine (The Sloan Technology Series).* Westview Press.

Capek, P. G., S. P. Frank, S. Gerdt, and D. Shields (2005). A history of ibm's open-source involvement and strategy. *IBM Systems Journal 44*(2), 249–257.

Castells, M. (2001). *The Internet Galaxy: Reflections on the Internet, Business, and Society*. Oxford University Press.

Castells, M. (2005). *Enformasyon Çağı (The Information Age: Economy, Society andCulture)*. Bilgi Üniversitesi Yayınları.

Castilla, E. J., H. Hwang, E. Granovetter, and M. Granovetter (2000). *The Silicon Valley Edge : a habitat for innovation and entrepreneurship*, Chapter Social Networks in Silicon Valley, pp. 218–424. Stanford University Press.

Ceruzzi, P. E. (2003). *A History of Modern Computing*. MIT Press.

Copeland, B. J. (2004). Colossus: its origins and originators. *IEEE Annals of the History of Computing 26*(4), 38–45.

Crick, F. (1990). *Şaşırtan Varsayım*. Tübitak.

DiBona, C., S. Ockman, and M. Stone (1999). *Open Sources: Voices from the Open Source Revolution*, Chapter Open Sources: Voices from the Open Source Revolution. O'Reilly.

Economides, N. and E. Katsamakas (2006). Two-sided competition of proprietary vs. open source technology platforms and the implications for the software industry. *Management Science 52*(7), 1057–1071.

Felleisen, M., R. Findler, M. Flatt, and S. Krishnamurthi. The structure and interpretation of the computer science curriculum.

Gençer, M., B. Oba, B. Özel, and V. S. Tunalıoğlu (2006). *Open Source Systems*, Chapter Organization of Internet Standards. IFIP Working Group 2.13 Foundation on Open Source Software 2006. Springer.

Gençer, M. (2010). Bilgi tarlasında mayın temizliği: Bilişim innovasyonunda demokratikleşme ve açık innovasyonun gelişimi. In *XV. Türkiye'de İnternet Konferansı*.

Hacking, I. (2005). *Şansın Terbiye Edilişi*. Metis.

Head, T., X. Chen, M. Yamamura, and S. Gal (2002). Aqueous computing: a survey with an invitation to participate. *J. Comput. Sci. Technol. 17*(6), 672–681.

Hodges, A. (2000, October). *Alan Turing: The Enigma*. Walker & Company.

Ifrah, G. (1995). *Çakıl Taşlarından Babil Kulesine II: Rakamların Evrensel Tarihi*. TÜBİTAK.

Knuth, D. E. (1996). *Selected Papers on Computer Science*. Stanford University Press.

Kogut, B. and A. Metiu (2001). Open-source software development and distributed innovation. *Oxford Review of Economic Policy 17*(2), 248–264.

Langlois, R. N. (1990). Creating external capabilities: Innovation and vertical disintegration in the microcomputer industry. *Business and Economic History 19*, 93–102.

Licklider, J. and R. Taylor (1968). The computer as communication device.

McKusick, M. K. (1999). *Open Sources: Voices from the Open Source Revolution*, Chapter Twenty years of Berkeley Unix: From AT&T-owned to Freely Redistributable. O'Reilly.

Newell, A. (1982, January). The knowledge level. *Artificial Intelligence 18*(1), 87–127.

Penrose, R. (1989). *Kralın Yeni Usu*. Tübitak.

Russell, S. J. and P. Norvig (2003). *Artificial intelligence : a modern approach*. Prentice Hall series in artificial intelligence. Prentice Hall/Pearson Education.

Sanchez, R. and J. T. Mahoney (1996). Modularity, flexibility, and knowledge management in product and organization design. *Strategic Management Journal 17*, 63–76.

Stallings, W. (2006, August). *Data and Computer Communications (8th Edition)* (8 ed.). Prentice Hall.

Turing, A. M. (1936). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society 42*, 230–265.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind 54*, 433–460.

Venables, W. N. and D. M. Smith (2002, May). *An Introduction to R* (1st ed.). Network Theory Ltd.

Watson, J. D. (2001, June). *The Double Helix: A Personal Account of the Discovery of the Structure of DNA*. Touchstone.

West, J. (2003). How open is open enough?melding proprietary and open source platform strategies. *Research Policy 32*, 1259–1285.

Yazar Dizini

Konu Dizini