**BLM 426 YAZILIM MÜHENDİSLİĞİ**

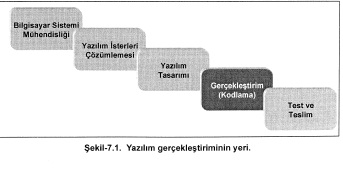
**BAHAR 2016**

**Yrd. Doç. Dr. Nesrin AYDIN ATASOY**

**9. HAFTA: YAZILIM GERÇEKLEŞTİRİMİ**

Bu bölümde, gerçekleştirim, yani kodlama aşamasının genel özellikleri üzerinde durulacak, kodlama ve biçimlerine değinilecektir.

Şekil 9.1’ de görülen gerçekleştirim aşamasında, tasarım, bir geliştirme ortamı üzerinde önce programlama diline ve sonra da makine koduna dönüştürülerek yürütülebilir kod (executable) oluşturulur.



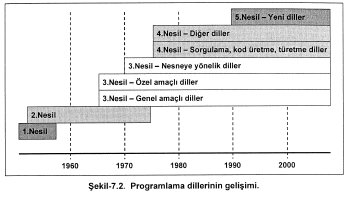
Şekil 9.1. Yazılım gerçekleştirim yeri.

# Programlama Dilleri

Programcılığa yaklaşımlar bilgisayarın icadından bu yana pek çok kez değişikliğe uğramıştır. Bunun en önemli nedenlerinden biri gittikçe karmaşıklaşan program isteklerine cevap verebilmektir. İlk bilgisayar programları Assembly dili ile yazılmaktaydı. Program boyutları büyüdükçe daha da üst düzey diller ortaya çıktı. Yaygın olarak kullanılan ilk üst düzey programlama dili FORTRAN'dır (Formula Translator). Ancak çeşitli eksiklik ve kısıtlamalar nedeniyle FORTRAN ile açık, anlaşılması kolay, uzun ve kapsamlı programlar yazmak zordu.

# Dillerin Gelişimi

Tarihçesine göre diller beş ana nesil içinde toplanabilirler. Şekil 9.2’ de bir zaman çizgisi üzerinde gösterilen dillerin gelişimini şimdi kısaca inceleyelim:



Şekil 9.2. Programlama dillerinin gelişimi.

# Birinci Nesil DilIer

İlk ortaya çıkan programlama dilleri makine düzeyinde kodlamaya, yani 1 ve 0'lardan oluşan sayı gruplarına dayanmaktadır. Daha sonradan Assembly denen insanların daha rahat okuyabileceği ilk dil ortaya çıkmıştır. Komut ve işlenen birimden oluşan kısa ifadeler halinde olan Assembly dili bilgisayarların sahip olduğu işlemci türüne göre farklılık gösterebilir.

# İkinci Nesil Diller

Bu diller 1950'lerden sonra geliştirilmeye başlanmış, 1960'ların sonunda ortaya çıkan modern dillerin temelini oluşturmuşlardır. İkinci nesil diller **zengin kütüphaneleri, geniş kullanım alanları ile oldukça yaygın hale gelmişlerdir.** Bu diller arasında FORTRAN, COBOL, ALGOL ve BASIC sayılabilir.

FORTRAN-66 ilk standart sürüm iken FORTRAN-77 standart sürüm (ANSI) olmuştur. COBOL, ticari alanlarda veri işleme amaçlı olarak kullanılan belirli standartlara sahip bir dil haline gelmiştir. ALGOL, üçüncü nesil dillere öncülük eden zengin veri tipleri ve yordamsal yapıya sahip bir dildir. BASIC, özellikle kişisel bilgisayarlarda kolay programlama olanağı sağlamıştır. Günümüzde ise bu dilin grafıksel gösterim amaçlı yeni bir sürümü VISUAL BASIC adıyla halen kullanılmaktadır.

# Üçüncü Nesil Diller

Bu nesil diller genellikle **modern ve yapısal diller olarak adlandırılır ve üç temel bölüme** ayrılırlar.

Genel amaçlı ilk dil olarak ALGOL görülmektedir. Bundan sonra onu model olarak alan PL/1, PASCAL, MODULA-2, C ve ADA izlemiştir. Bu diller, mühendislik, bilim, gömülü sistemler ve kontrol sistemleri geliştirilmesinde yaygınlıkla kullanılmaktadır.

C programlama dili aslında işletim sistemleri için geliştirilmiştir. Ünlü Unix işletim sistemi de C ile geliştirilmiştir. Günümüzde de en yaygın ve en güçlü diller arasındadır.

ADA, Amerikan Savunma Bakanlığı'nın yazılım krizine çözüm olması amacıyla gömülü gerçek-zamanlı sistemler için geliştirilmiştir.

Nesneye yönelik yüksek düzey diller, nesneye yönelik çözümleme ve tasarımı desteklemek üzere geliştirilmiş dillerin sayısı çok artmıştır. Ancak içlerinden yalnızca C++, Objective-C, SMALLTALK, PASCAL (nesneye yönelik sürümü), JAVA dilleri uygulama alanı bulmuşlardır. Bunlardan da C++, C# ve JAVA günümüzün en yaygın dilleri arasındadır. Bu dillerin ortak özellikleri sınıf tanımlamaları, kapsama ve kalıtım ile çeşitli nesneye yönelim özelliklerini desteklemeleridir.

Özel amaçlı diller, belirli bir uygulama tipi için tasarlanmış, alışılmışın dışında yapılara ve söz dizimlerine sahiptirler. Bu tür diller dar bir uygulama alanına sahiptirler. Ancak, belirli bir uygulama tipinin gereksinimlerini karşılamak üzere geliştirildiklerinden, isterleri tasarıma ve tasarımı koda dönüştürme işlemi daha başarılı olmaktadır. En önemli diller arasında LISP, PROLOG ve APL sayılabilir. LISP, simge ve listelerle ardışık bilgi işleme sağlamaktadır. Özellikle yapay zeka uygulamalarında, ağaç yapıları yardımıyla problem çözme amaçlı olarak kullanılır. PROLOG uzman sistemlerin geliştirilmesinde önemli yeri olan bir dildir. APL, diziler ve vektörlerle işlem yapmak için iyi bir gerçekleştirim aracıdır.

# Dördüncü Nesil Diller

Dördüncü nesil diller, yordamsal olan ve olmayan yapıları birleştirici özelliklere sahiptirler.  
Bu nesil dilleri birkaç grupta incelemek mümkündür:

# Sorgulama Dilleri

Veritabanı yönetimi çok geniş bir uygulama alanında kullanılmaktadır. Özellikle veri tabanlarına erişmek, sorgulama ve raporlama yapmak üzere çeşitli diller geliştirilmiştir. DBASE III/IV, CLIPPER, DELPHI PARADOX bu diller arasında sayılabilir. Bazı sorgulama dilleri de kullanıcının var olan veri tabanları üzerinde işlem yapabilmesini sağlarlar. SQL (Structured Query Language) en yaygın olan dillerden biridir. Bu diller de İngilizce konuşma diline çok yakındır bir syntax yapısına sahiptir.

# Kod Üreticiler

Çoğu zaman program kodunu oluşturmak üzere üçüncü nesil bir dili kullanarak uzun satırlar yazmak gerekebilir. Kod üreticiler, kullanıcının daha üst düzey yapı ve anahtar sözcüklerle daha az miktarda kod yazarak sözdizimi tam doğru olan üçüncü nesil bir dilde kod üretilmesini sağlarlar. Günümüzde yaygın bir şekilde kullanılan bilgisayar destekli yazılım mühendisliği (Computer Aided Software Engineering - CASE) araçları (UML, STP, SDL, MDA araçları gibi), grafik tabanlı kullanıcı arayüzü geliştirme araçları, iş dünyası için çeşitli araçlar C, C++, ADA gibi dillerde kod üretmektedirler. Üretilen bu kod uygun bir derleyiciden geçirilerek yürütülebilir kod elde edilmektedir.

# Türetme Diller

Özellikle belirli alanların sorunlarını daha kolay ve etkin bir şekilde çözebilmek için bazı dillere eklemeler ve uzantılar yapılarak yeni diller türetilmektedir.

Örneğin, "Real-Time C++" (RTC++), C++ diline bir uzantı olarak gerçek zamanlılığı desteklemektedir. "Mentat", yine C++ diline uzantı olarak paralel ve dağıtık programlama olanağı sunmaktadır.

# Diğer Diller

Karar destek, prototiplerine ve belirtim gibi çeşitli amaçlara hizmet eden diller de bulunmaktadır. Arayüz belirtim (CORBA-IDL), toplu iş tanımlama (script, batch fıle) gibi işlevleri yerine getirmede kullanılan diller olduğu gibi kişisel bilgisayarlarda yaygın bir şekilde kullanılan çalışma tabloları, veritabanı sistemleri de ayrı diller kullanmaktadırlar. XML (Extensible Markup Language) daha çok internet dünyasında kullanılan HTML (Hypertext Markup Language) için bir standart getirmek üzere tanımlanmıştır. Bu diller insanların okuyabileceği türdeki verileri biçimlendirilmiş bir şekle getirmek için kullanılır.

# Beşinci Nesil Diller

Geleceğe hükmedecek bu son nesil diller çeşitli araştırmalar halindedirler. Bu tür diller yeni oluşumları kapsayacaklardır. Örneğin, nesneye yönelik programlamadan sonra gelişen yeni bir yaklaşım da konuya yönelik (subject-oriented) programlamadır. Yeni programlama dilleri, daha fazla kapsama oluşturacak, belki de bizi ardışık düşünmekten kurtaracaktır.

# Dillerin Özellikleri

# Genel Özellikler

Programlama dillerinin genel özelliklerini şu şekilde özetleyebiliriz:

# Tasarımdan koda geçiş kolaylığı

Bu işlemler, veri tanımlamaları, giriş/çıkış düzenekleri, zaman belirtimi, hata kotarımı, aritmetik, mantık ve bit işlemleri, nesneye yönelim özellikleri, soyut veri tipleri, modülerlik sayılabilir.

# Amaca uygunluk

Uygulama alanının ve geliştirilecek yazılımın işlevsel ve başarım özelliklerine göre programlama dili seçimi yapılmalıdır. Örnek olarak, C++ dilini seçip tasarımı nesneye yönelik olarak yapmayı, ADA dilini seçip task özelliğini kullanarak paralel programlama yapmayı verebiliriz.

# Dilin etkinliği

Yaygın programlama dilleri İngilizce üzerine kurulmuş olup konuşma diline yaklaştıkça hem yazılması hem de okunması kolaylaşır, düşüncelerin makineye anlatılması daha etkin olur. Bu da kodlamayı yapan insanların verimliliğini artırır. Yapıları zayıf, okuması, yazması ve anlaması güç bir dil ile bilgisayara istenileni yaptırmak son derece güç olacağından emek ve zaman kaybı kaçınılmaz olur.

# Derleyici etkinliği

Programlama dilinden üretilen makine kodunun etkinliği yürütme sırasında ortaya çıkar. Derleyiciler kaynak kodu çözümlemeden önce ne kadar iyi denetlerler ve üretilen makine kodunu ne kadar iyileştirirlerse o kadar başarılı sayılırlar.

# Taşınabilirlik

Bir programlama dili ile yazılan kaynak kodun değişik derleyicilerle, işletim sisteminin yeni sürümleriyle ve hatta başka makineler üzerinde sorunsuzca çalışması arzu edilir.   
Bu da ancak belirli standartlara sahip dillerin kullanılmasıyla gerçekleşebilir. Bu standartlar genellikle ISO (International Standardization Organization), ANSI (American National Standardization Institute) ve OMG (Object Management Group) tarafından sağlanmaktadır.

# Bakım Kolaylığı

Yazılım bakımının kolay olmasındaki en büyük etkenlerden biri, esnek bir tasarımın yanında, kaynak kodun niteliğidir.

# Tip kontrolü

Programlamanın temelinde veri işleme yatar. Veriler bilgisayar donanımının kullandığı temel tipler (karakter, tamsayı ve kayan nokta sayısı) haline dönüştürülerek kullanılırlar. Programlama dilinin tip kontrolü yapma yeteneği güvenilir yazılım geliştirme için son derece gerekli bir özelliktir.

# Denetim yapıları

Tüm programlama dilleri kodlayıcıya ardışık komutlardan, koşul testine göre dallanmalardan ve tekrarlardan oluşan mantıksal yapılar sunarlar. Klasik dillerin bazılarında hata yakalama düzenekleri, paralel programlama yapıları, zaman denetimli yapılar bulunmaktadır.

# Nesneye Yönelik Dillerin Özellikleri

**Modülerlik:**Yazılım, kendi üzerinde işlemler yapılmasına izin veren basit ve tutarlı arayüzlere sahip ayrı modüller halinde geliştirilir.

**Veri soyutlama (abstraction):** Nesneler hem verileri hem de onlar üzerindeki işlemleri kapsayan soyut veri tipleri şeklinde tanımlanırlar.

**Otomatik bellek yönetimi:** Sistem öz kaynaklarını kullanan nesneler işleri bitince onları otomatik olarak serbest bırakırlar.

**Sınıflar:** Sınıf, kendisinden nesnelerin oluşturulan, fakat diğer basit tiplerden olmayan özel yapıda bir tiptir.

**Kalıtım (inheritance):** Bir sınıf, temel sınıfın veri ve işlemlerinin bir kısmını ya da tamamını devralarak türetilir. Bu türetimi çoklu ve tekrarlanan bir şekilde yapmak mümkündür.

**Çokşekillilik (polymorphism):** Bir nesne, yürütme anında, çeşitli sınıflardan oluşturulmuş başka nesneleri işaret edebilir, yürütme anında dinamik olarak başka nesnelere bağlanabilir.

Bütün koşulları karşılayan dillere nesneye yönelik, yalnızca ilk dördünü karşılayan dillere ise nesneye dayalı adı verilmektedir.

# Dil Seçimi

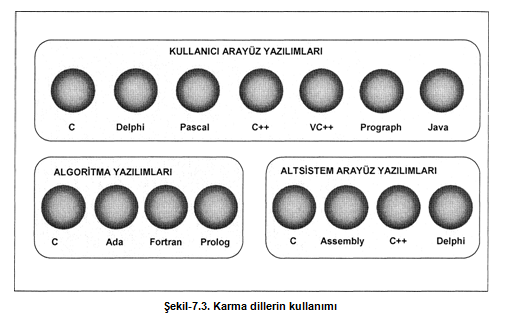
Yazılımın kullanılacağı bilgisayar ortamı, donanımın ve işletim sisteminin özellikleri dikkate alınmalıdır. Seçilecek dilin yetenekleri, algoritmik ve işlemsel karmaşıklık derecesini karşılayabilir olmalıdır. Veri yapılarının özellikleri ve karmaşıklık dereceleri seçilecek dilin yetenekleri arasında olmalıdır. Seçilecek dil, yazılımın bakım aşamasında son derece önemli olan okunabilirlik ve anlaşılabilirlik özelliklerini desteklemelidir.

# Dillerin Uygulama Alanları

Uzun yıllardır en güçlü ve en yaygın diller arasında C dili gösterilmektedir. Sistem yazılımlarında (işletim sistemi, ara katman, aygıt sürücü gibi) tercih edilen C'nin nesneye yönelik uyarlaması olan C++ da onun kullanıldığı pek çok alanda yaygın olarak kullanılmaktadır.

Eğitim dili olarak yapılmış PASCAL; görsel programlama dili Visual BASIC, Visual C++, DELPHI; gerçek zamanlı sistemlerde ADA (83 ve 95), C, MODULA-2; iş dünyası yazılımlarında COBOL,LINK; yapay zeka uygulamaları ve uzman sistemlerde PROLOG, LISP.

Nesneye yönelik çözümleme ve tasarım kullanılması durumunda, C#, Objective-C, C++ ve JAVA ön plana çıkmaktadır. Bu deneyimlere göre, bazen bir bilgisayarlı sistemin tamamında tek bir dil kullanmak mümkün olmayabilir. Örneğin, bir kontrol sistemi yazılımında, C ile yazılmış aygıt sürücüler ADA veya C++ diliyle denetlenirken, grafıksel kullanıcı arayüzü için C veya C++ kullanılabilmektedir. Şekil-7.3 çeşitli dillerin beraberce kullanıldığı karma bir yapıyı göstermektedir:



Şekil 9.3. Karma dillerin kullanımı.

# Kod Çevrim İşlemi

Programlama dillerinin en önemli özelliklerinden biri de insan ile makine arasında ne tür bir geçiş sağladığıdır. İnsanlar normal karakterlerle gösterilen yazı ve sayıları anlayabilirler. Ancak bir bilgisayar yalnızca 0 ve 1 değerlerini alabilen ikili düzenden, yani "bit'lerden anlamaktadır. Dolayısıyla arada iletişimi sağlayacak bir çevirmene gereksinim vardır.

Normal olarak bir program ASCII biçimindedir. ASCII, insanların kullandıkları simge, harf ve rakamlar yerine onların sayısal karşılıklarını kullanır. Bu kodlar 0 ile 255 arasındadır. Her bir karakter bir sekizli (byte) ile gösterilir. Günümüzün gereksinimleri değerlendirildiğinde bu kadar karakterin yetmediği görülmüş ve iki sekizli ile gösterilen yeni ve daha evrensel bir kod sistemi de oluşturulmuştur. Halen birçok sistem, standart ASCII kullanmaktadır.

# Derleyiciler

Normal metin halinde yazılmış program içeren dosyalara kaynak kod adı verilmektedir. Derleyici (compiler) bu dosyada bulunan metni oluşturan ASCII karakterlerini, dilin sözdizim kurallarına uygun olarak okur, hata yoksa bunları bilgisayarın anlayabileceği komutlardan oluşan makine diline (Assembly) çevirir. Bu komutlar da bilgisayarın işlemcisinin tanıyabilmesi için nesne koduna (object code) çevirir.  
Derleyicilerle birlikte kullanılan önemli terimlerden biri derleme anı diğeri yürütme anıdır. Derleme anı, kaynak kodun derlenmesi işlemi sırasında meydana gelen olaylar için, yürütme anı da yürütülebilir durumdaki kodun işlemci üzerinde çalıştırılması sırasında meydana gelen olaylar için kullanılır.

# Yorumlayıcılar

Bazı yazılımlar yorumlayıcı (interpreter) adı verilen özel altyapılarla birlikte kullanılırlar. Standart BASIC, LISP, PROLOG ve JAVA birer örnektir. Bir yazılım çalıştırılmadan önce yorumlayıcının bilgisayara yüklenmesi ve onun içinden de kaynak kodun yüklenerek çalıştırılması gerekir. Yorumlayıcı kaynak kodun satırlarını birer birer okuyarak işlemci komutları haline dönüştürür ve bu komutları uygular. Bu nedenle derlenmiş koda göre daha yavaştır.

# Geliştirme Ortamı

Yazılım geliştirmek için ilk sahip olunması gereken, uygun bir geliştirme ortamıdır. Ana sistem olarak da adlandırılan bu sistemin yanında geliştirilen yazılımın test edilmesi için bir test sistemi ve gerçek sistem yani hedef sistem bulunmalıdır.

# Yürütme Ortamı

Tek bilgisayardaki yürütme ortamı genellikle işletim sistemidir. Bazen, bunun üzerinde bir başka katman daha konarak daha etkin çoklu programlama ortamı oluşturulur (Java Virtual Machine bunlardan biridir). Dağıtık bilgi işlemeyi destekleyen çeşitli bilgisayar yazılım mimarileri vardır. Özellikle bir yerel alan ağı üzerindeki uygulama yazılımlarının bir arada ve uyum içinde çalışmasını sağlayan, onları alttaki bilgisayar ve ağ yapısından ayıran ara katman yazılımlarına daha önce değinmiştik.

# Kodlama Biçimleri

Tasarım ne kadar iyi yapılırsa yapılsın onu hayata geçiren işlem kodlamadır. Günümüzde kodlama, yazılım geliştirme sürecinin hala büyük emek gerektiren, önemli, fakat bazıları için biraz can sıkıcı bir evresidir. Yazılım geliştirmede Türkçe ya da bir başka dil kullanımı çoğu zaman tartışma konusudur. Belgelendirme ve kodlamada hangi dilin kullanılacağını bazen müşteri belirler bazen de yönetimin kararı uygulanır. Derleyicilerin çoğu da yalnızca İngiliz alfabesini desteklemektedirler. Türkçenin bozulmasını engellemek amacıyla kaynak kodun ingilizce olarak yazılması daha tutarlı olmaktadır. Türkçe harfleri destekleyen dil veya derleyici kullanmak başka ortamlara taşınabilirliği büyük ölçüde azaltmaktadır. Hangi aşamada hangi dilin kullanılması gerektiği yönetim tarafından alınması gerekli önemli bir karardır. Bu kararın verilmesi tamamen teknik amaçla olmalı, yalnızca anadilimizin zarar görmesini engellemeye çalışmak hedeflenmemelidir.

# Kod Belgelemesi

Bu belgeleme işlemi içinde, tanımlayıcı isimlerinin seçilmesi, açıklamalar ve görsel yapı yer alır. Şimdi bu özellikleri içerecek şekilde kod belgeleme kurallarına değinelim:

# İsimlendirme

* Bir dilde tanımlamalarda anahtar sözcük veya onun yakın bir benzeri isim olarak kullanılmamaldır.
* Değişken ve nesne isimlerinde birden fazla sözcük kullanılacaksa aralarına alt çizgi işareti (\_) konmalıdır.
* Yazılımı küçük tutmak amacıyla kısa isimler kullanımı doğru değildir. İsimlerin uzun veya kısa olması çalışmayı ve boyutu etkilemez.
* Proje boyunca aynı isimlendirme yöntemi kullanılmalıdır.
* Bazı dillerde büyük ve küçük harf ayrımı olduğu unutulmamalı, buna göre bir isimlendirme yöntemi seçilmelidir.
* Karmaşıklığa yol açabilecek veya başkası tarafından okunduğunda anlaşılmayı güçleştirecek isimler kullanılmamalıdır.
* Evrensel bir değişken veya sınıf ismi aynı zamanda yerel olarak kullanılmamalıdır.
* Program başında tanımlanan sabitlerin veya makroların isimlendirilmesinde büyük harfler kullanılması daha uygundur.
* İsimlendirmede Türkçe ve İngilizce sözcükleri karıştırarak kullanılmaması, kodun tamamında projenin başında seçilen dilin kullanılması önerilir.

# Açıklamalar

* Her türlü yazılımın mutlaka bir iç belgelendirmesi olmalıdır. Bunun yollarından biri de açıklama satırlarıdır.
* Açıklama satırlarının değiştirilmesi, sonradan bakımı biraz zor olabileceği için aşırıya kaçılmamalıdır
* Çok sıkı denetlenen nitelik güvence yöntemleri uygulandığı takdirde, açıklama satırları arasında, hangi yazılım isterinin karşılandığı yazılır.

# Hata Ayıklama

* Hata ayıklama ilk kodlama sırasında kodlayıcının en çok yaptığı iştir.   
  Özellikle hata ayıklama gereksinimi ortaya çıktığında geliştiricinin en klasik yardımcısı standart çıkış ekranına kod içinden iletiler yazdırmaktır (cout, printf,put\_line gibi komutlarla).
* Ekran çıktılarının her zaman değil de yalnızca istendiği zaman ve istenen miktarlarda kullanabilmek için mutlaka bir düzenek kullanılmalıdır. Bazı programlama dilleri ve derleyiciler koşullu derleme özelliği ile bu tür kod parçalarının tamamen gizlenmesini sağlayabilirler (C ve C++ için #ifdef - #endif yapısı gibi). Şekil-7.5, C++ için iki adet düzenek örneği göstermektedir:

|  |  |
| --- | --- |
| #define DEBUGPRINT |  |
| #ifdef DEBUGPRINT |  |
| cout « “X = *" <<* xval << endl; |  |
| #endif |  |
| #ifdef DEBUGPRINT |  |
| #define PRINTl(A) cout « A « endl |  |
| #define PRINT2(A,B) cout « A « B « | endl |
| #define PRINT3(A,B,C)cout « A « B « | C « endl |
| #else |  |
| #define PRINTl(A) /\*PRINT1(A)\*/ |  |
| #define PRINT2(A,B) /\*PRINT2(A,B)\*/ |  |
| #define PRINT3(A,B,C)/\*PRINT3(A,B,C)\*/ |  |
| #endif |  |

# Veri Bildirimi

* Bir tek program içinde ortak olarak kullanılan veri tipleri aynı dizinde bulunan ve beden dosyaları tarafından içerilen başlık dosyalarına konmalıdır.
* Evrensel veri yapıları ve değişkenler, görülebilirlik kurallarına uygun olarak belirli bir dosyada toplanmalıdırlar.
* Veri tiplerinden değişken bildirimi yapılırken, tek sırada değil de alt alta ve alfabetik bir sıra izlenmesi aranan bir değişken isminin kolay bulunmasını sağlar.

# Deyim Yapıları

* Her deyim bir satırda yer almalı, gereksiz karmaşıklıklar içermemelidir.
* Karmaşık koşul testleri (if yapıları) kullanmaktan kaçınılmalıdır.
* İç içe girmiş fazla sayıda döngü ve koşullu dallanmalar kullanılmamalıdır.
* Koşullarda gereksiz ve tekrarlanan testler yapılmamalıdır.
* Deyimler, yazan kişiden başkasının da okuyup anlayabilmesi için olabildiğince basit, ancak işlevini yapabilecek şekilde yazılmalıdır.

# Etkinlik

Mühendisliğin en önemli özelliklerinden biri de kaynakların etkin ve verimli bir şekilde kullanılarak amaçlara ulaşılmasıdır. Bilgisayar sistemlerinin etkinliği de tanımlanan işi amaca uygun hız ve güvenlikle en ekonomik şekilde karşılamasıyla değerlendirilir.

Bunun için de kaynakların iyi bir şekilde kullanılması gerekir. Bilgisayarlı sistemlerin özkaynakları, bilgi işleme kapasitesine sahip makineler, bunların işlemcilerinin zamanları, bellekleri, saklama alanlarıdır.

Etkinlikle ilgili dikkat edilmesi gereken ve önceden de değinilen birkaç özellik şunlardır:

* Kod Etkinliği
* Bellek Etkinliği
* Giriş/Çıkış Etkinliği

Bir bilgisayar sisteminin genellikle iki tür giriş/çıkışı vardır. Bunlardan birincisi, insanla olan etkileşim, yani kullanıcı arayüzü, diğeri de başka aygıtlarla ve çevre birimleriyle olan etkileşim, yani veri arayüzüdür.

En önemli kaynaklardan biri ana ve yardımcı bellektir. Özellikle dinamik sistemlerde bellek kullanım gereksinimi zaman içinde farklılık gösterdiğinden, uzun süreli çalışmalarda bellek sıkıntısına düşmemek için kullanımı sonra eren nesne ve veri yapılarının silinmesi gereklidir. Silme işlemi geliştiricinin kendi algoritmalarıyla yapılabileceği gibi, bir ara katman, bir yürütme sistemi veya işletim sistemi, hatta özel bir donanım tarafından da yapılabilir. Bu işleme atık toplama (garbage collection) denir.

Ana ve yardımcı bellek yanında, bir yazılım birimi, dosya tutamacı (fıle handle), giriş/ çıkış aygıtı, paylaşılır bellek, semafor, soket gibi kaynakları da kullanabilir. İşletim sistemine göre değişiklik göstermesine rağmen, bu tür kaynakların kullanımı sınırlıdır. Normal olmayan bir şekilde çalışması duran bir yazılım birimi kullanmakta olduğu kaynakları serbest bırakabilmelidir. Bazı işletim sistemleri bunu güvenli bir şekilde sağlarken, bazıları çöken yazılımla birlikte tüm sistemin de çökmesine neden olmaktadır.

Günümüzde bazı programlama dilleri dinamik bellek yönetimini kendi içinde yapmakta, hatta bazıları dinamik yapılanmaya izin vermemektedir. JAVA dili işaretçi kullanmayan ve kendi atık toplama düzeneğine sahip bu dillerden biridir.

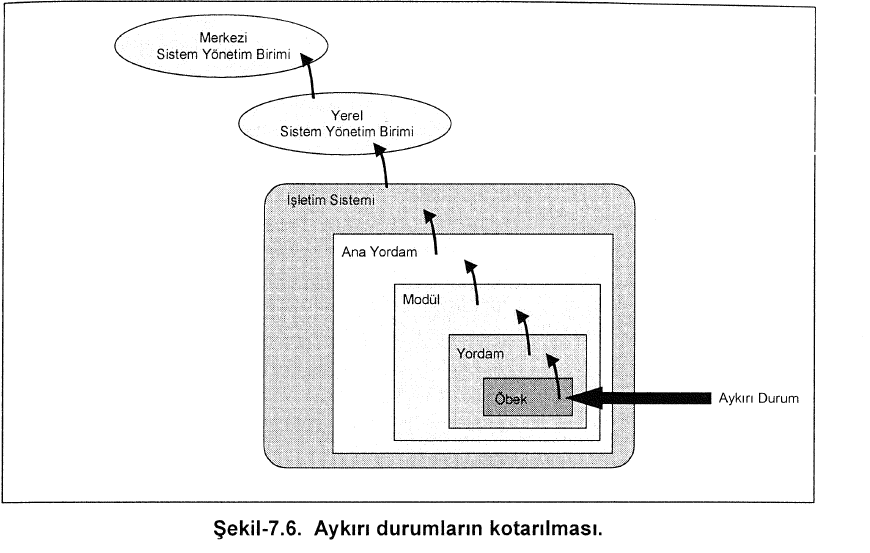
# Aykırı Durumların Kotarılması

Aykırı durumlar (exception), bir programın çalışmasını geçersiz bir komut yürütme, yanlış veri işleme ya da başka nedenlerle istenmeyen bir şekilde ve denetim dışında sonlanmasına neden olan durumlardır.

Günümüzdeki programlama dillerinin bir kısmı aykırı durumları yakalayabilen düzeneklere sahiptir (C++ için try-catch, ADA için exception öbekleri gibi).

Yürütme sırasında herhangi bir olağan dışı durum oluştuğunda Şekil 9.4 ‘te gösterilen sıraya göre çeşitli geri kazanma işlemleri yapılabilir. Aşağıda belirttiğimiz herbir basamakta durum düzeltilemezse hatanın cinsi ve yeri tanımlanarak üste doğru taşınmalıdır.

* Kod öbeği içinde kotarma
* Yordam içinde kotarma
* Modül içinde kotarma
* Ana yordamda kotarma
* Durdurma



Şekil 9.4. Aykırı durumların kotarılması.

# Modül Oluşturma

Dosya halinde modül oluşturmada dikkate alınması gereken noktalara değinmekte yarar vardır:

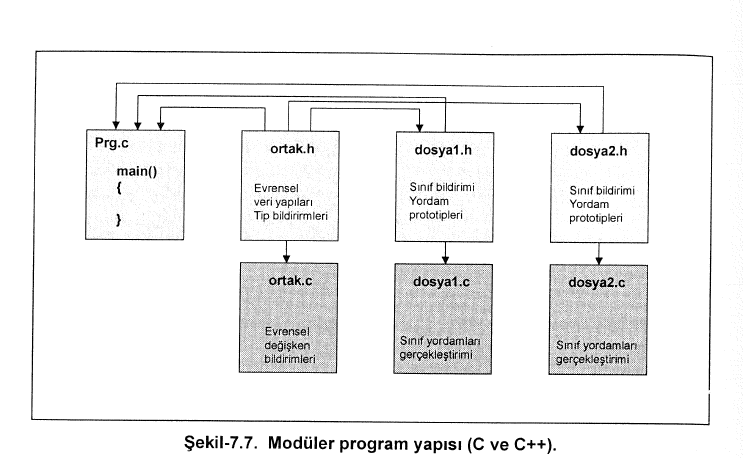
Yazılımın büyüklüğü arttıkça yordamları ve verileri birbirlerine olan bağlılıkları düşünülerek birimlere ayırmak gereklidir.

Birbirleriyle ilişkili yordamlarla bunlara ait verileri bir dosya içine koyarak bir modül oluşturulabilir. Nesneye yönelik programlamada modül olarak sınıflar da kullanılabilir  
Yaratılan herbir dosyanın baş kısmına yeterli bir açıklama konmalıdır.   
Her tür program geliştirmede örnek olacak C++ dili için bir dosya yapısı Şekil 9.5’ te görülmektedir.

Bir yazılım birimi, yani program için evrensel olan tipler, veri yapıları, sabitler ve makrolar ayrı bir dosya içinde toplanmalı ve bu dosya diğerleri tarafından erişilmelidir.

Aynı tür işlevlere sahip yordamlar belirlenmeli ve bir modülde toplanmalıdır.

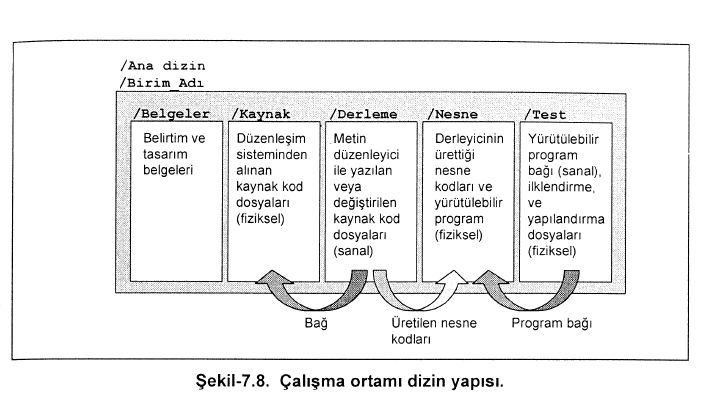
Modülün açık, anlaşılır bir arayüzü olmalıdır. Bu da genellikle bir başlık dosyası (header) veya paket belirtimidir.



Şekil 9.5. Modüler program yapısı (C - C++).

# Kod Yazımı

Kod yazmak, her ne kadar bir sanat niteliğinde olsa da, bir takım kurallara uymak ve disiplinli olmak gereklidir. Bir programlama dili bilen herkes "program" yazabilir, fakat herkes "yazılım" geliştiremez. Tüm yazılım geliştirme aşamalarının gerçekleşmesi ve belirli bir mühendislik bilgisinin denetiminde kod yazılması nitelikli yazılım için temel oluşturur. Kodlama, ana sistem üzerinde, uygun bir dizin yapısı kullanılarak oluşturulan proje ortamında yapılır. İdeal olarak, her çalışma gününün sonunda geliştirilen kodlar birleştirilerek yeni bir sürüm üretilmeli ve çalışma alanının yedeklenmesi sağlanmalıdır. İdeal bir çalışma ortamı dizin yapısı Şekil 9.6’ da görülmektedir:



Şekil 9.6. Çalışma ortamı dizin yapısı.

# Belgelendirme

Yazılım gerçekleştirim aşamasında mutlaka bir belge üretilmesine gerek yoktur. Ancak, kodlama sırasında elde edilen bazı bilgileri (deneyim, test sonuçları, önemli noktalar gibi) gelecekte kullanabilmek üzere düzenleşim yönetim sisteminde saklamak gerekebilir. Gerçekleştirime yönelik belgelendirme çalışmalarını şu şekilde özetleyebiliriz:

* Her bir kaynak kod dosyasının başında, bir başlık kısmı bulunmalıdır.
* Kodlamada çalışmış kişilere daha sonradan yararlı olacağı düşünülen önemli noktaları, derleme sırasında karşılaşılan sorunları ve bunların nasıl çözüldüklerini anlatan kayıtlar bir dosyada tutulmalıdır. (Bazen bu dosyalara "okubeni.txt" şeklinde isim verilir).
* Küçük programlar için her türlü derleme bilgisi ana fonksiyonun bulunduğu dosyanın başına, açıklama halinde yazılmalıdır.
* Ana ve test sistemi üzerinde yapılan testlerde kullanılan her türlü yazılım, ayarlama ve yapılandırma gerekli açıklamalarla beraber kaynak kod yanında düzenleşim yönetim sistemine girmelidir.
* Kod yapısını, tasarım belgelerinde yer almayan çeşitli bilgileri içeren, açıklayıcı özellikte her türlü belge dosyası kaynak kod ile beraber düzenleşim yönetim sisteminde saklanmalıdır.

**KAYNAKLAR**

1. Yazılım Mühendisliği; M. Erhan Sarıdoğan.