CENG 218 Programlama Dilleri

Bölüm 2: Dil Tasarım Kriterleri

Öğr.Gör. Şevket Umut Çakır

Pamukkale Üniversitesi

Hafta 2

Hedefler

- Programlama dili tasarım kriterlerinin tarihini açıklamak
- Programlama dillerinde verimliliği(efficiency) anlamak
- Programlama dillerinde düzenliliği(regularity) anlamak
- Programlama dillerinde güvenliği(security) anlamak
- Programlama dillerinde genişletilebilirliği(extensibility) anlamak
- C++ 'ın tasarım hedeflerini anlamak
- Python'un tasarım hedeflerini anlamak

Arka Plan

- İyi programlama dili tasarımı nedir?
- Bir dili yargılamak için hangi kriterler kullanılmalıdır?
- Bir dilin başarısı veya başarısızlığı nasıl tanımlanmalıdır?
- Bu kriterlerden herhangi birini veya tümünü karşılayan bir dili başarılı olarak tanımlayacağız:
  - Tasarımcılarının hedeflerine ulaşır
  - Bir uygulama alanında yaygın kullanıma ulaşır
  - Başarılı olan diğer diller için bir model görevi görür

Arka Plan

- Yeni bir dil oluştururken, genel bir hedefe karar verilir ve bu tasarım süreci boyunca akılda tutulur
- Bu özellikle özel amaçlı diller için önemlidir
  - Hedef uygulama alanı için soyutlamalar dil tasarımına dahil edilmelidir
- Bu bölüm, bazı genel tasarım kriterlerini tanıtır ve tasarımcıya potansiyel yardımcılar olarak bir dizi ayrıntılı ilkeyi sunar

Tarihsel Genel Bakış

- İlk zamanlarda makineler son derece yavaştı ve bellek yetersizdi
  - Program hızı ve bellek kullanımı başlıca endişelerdi
- **Uygulama verimliliği(Efficiency of execution)**: birincil tasarım kriteri
  - İlk FORTRAN kodları aşağı yukarı doğrudan makine koduyla eşleştirilerek derleyicinin ihtiyaç duyduğu çeviri miktarını en aza indirir
- **Yazılabilirlik(Writability)**: bir programcının onu hesaplamayı net, doğru, kısaca ve hızlı bir şekilde ifade etmek için kullanmasını sağlayan bir dilin kalitesi

Tarihsel Genel Bakış

- İlk günlerde yazılabilirlik verimlilikten daha az önemliydi
- Algol60, algoritmaları mantıksal olarak açık ve öz bir şekilde ifade etmek için tasarlanmıştır
  - Birleştirilmiş blok yapısı, yapılandırılmış kontrol ifadeleri, daha yapılandırılmış bir dizi türü ve özyineleme
- COBOL, sıradan İngilizce gibi görünmelerini sağlamaya çalışarak programların okunabilirliğini artırmaya çalıştı
  - Ancak, bu kodları uzun ve ayrıntılı yaptı

Tarihsel Genel Bakış

- 1970'lerde ve 1980'lerin başında vurgu, güvenilirliğin yanı sıra basitlik ve soyutlama üzerineydi
  - Bir çevirmenin çeviriden önce bir programın doğruluğunu kısmen kanıtlamasına olanak tanıyan mekanizmalarla birlikte dil yapıları için matematiksel tanımlar tanıtıldı
  - Bu, güçlü veri tiplerine(strong data typing) yol açtı
- 1980'lerde ve 1990'larda mantıksal veya matematiksel kesinliğe vurgu yapıldı
  - Bu, fonksiyonel dillere olan ilginin yenilenmesine yol açtı

Tarihsel Genel Bakış

- Son 25 yılın en etkili tasarım kriterleri, soyutlamaya yönelik nesne odaklı yaklaşımdır
  - Mevcut kodun yeniden kullanılabilirliğini artırmak için kütüphanelerin ve diğer nesne yönelimli tekniklerin kullanılmasına yol açtı
- Verimliliğin erken hedeflerine ek olarak, neredeyse her tasarım kararı hala okunabilirlik, soyutlama ve karmaşıklık kontrolünü dikkate alır

Verimlilik(Efficiency)

- **Verimlilik(Efficiency)**: genellikle hedef kodun verimliliği olarak düşünülür
- Örnek: derleme zamanında zorunlu kılınan güçlü veri türü(strong data typing), çalışma zamanının işlemleri yürütmeden önce veri türlerini kontrol etmesine gerek olmadığı anlamına gelir
- Örnek: ilk FORTRAN sürümleri, bellek alanının yürütmenin başlangıcında bir kez tahsis edilmesine izin vermek için tüm veri bildirimlerinin ve alt rutin çağrılarının derleme zamanında bilinmesini gerektirdi.

Verimlilik(Efficiency)

- **Programcı verimliliği**: Bir kişi programlama dilinde ne kadar hızlı ve kolay bir şekilde okuyup yazabilir?
- **İfade edicilik(expressiveness)**: Karmaşık süreçleri ve yapıları ifade etmek ne kadar kolay?
- Sözdiziminin kısa olması da programcı verimliliğine katkıda bulunur
  - Örnek: Python süslü parantez veya noktalı virgül gerektirmez, yalnızca girinti ve iki nokta üst üste (:)

Verimlilik(Efficiency)

- Bir programın güvenilirliği bir verimlilik sorunu olarak görülebilir
  - Güvenilir olmayan programlar, programcının teşhis etmesi ve düzeltmesi için zaman gerektirir
- Programcı verimliliği, hataların bulunma ve düzeltilme kolaylığından da etkilenir
- Zamanın kabaca %90'ı programların hata ayıklaması ve bakımına harcandığından, sürdürülebilirlik(maintainability) programlama dili verimliliğinin en önemli göstergesi olabilir

Düzenlilik(Regularity)

- **Düzenlilik(Regularity)**: bir dilin özelliklerinin ne kadar iyi entegre edildiğini ifade eder
- Daha fazla düzenlilik şu anlama gelir:
  - Belirli yapıların kullanımına ilişkin daha az kısıtlama
  - Yapılar arasında daha az garip etkileşim
  - Dil özelliklerinin davranış biçiminde genel olarak daha az sürpriz
- Düzenlilik kriterini karşılayan dillerin en az şaşkınlık(astonishment) ilkesine bağlı olduğu söylenir

Düzenlilik(Regularity)

- Düzenlilik üç kavrama ayrılabilir:
  - Genellik(Generality)
  - Ortogonal tasarım(Orthogonal design)
  - Tekdüzelik(Uniformity)
- **Genellik(Generality)**: Yapıların mevcudiyetinde veya kullanımında özel durumlardan kaçınarak ve yakından ilgili yapıları tek ve daha genel bir şekilde birleştirerek elde edilir
- **Ortogonal tasarım(Orthogonal design)**: yapılar, beklenmedik kısıtlamalar veya davranışlar olmaksızın anlamlı herhangi bir şekilde birleştirilebilir

Düzenlilik(Regularity)

- **Tekdüzelik(Uniformity)**: Farklı şeylerin farklı görünürken benzer şeylerin benzer göründüğü ve benzer anlamlara sahip olduğu bir tasarım
- Bu üç nitelikten birine sahip değilse bir özellik veya yapı düzensiz olarak sınıflandırabilir

Genellik(Generality)

- **Genellik**: Bu özelliğe sahip bir dil, mümkün olduğunca özel durumlardan kaçınır
- Örnek: prosedürler ve fonskiyonlar
  - Pascal, fonksiyonların ve prosedürlerin iç içe oluşturulmasına; fonksiyonların ve prosedürlerin parametreler olarak diğer fonksiyonlara ve prosedürlere aktarılmasına izin verir, ancak bunların değişkenlere atanmasına veya veri yapılarında depolanmasına izin vermez
- Örnek: operatörler
  - C'de, iki yapıyı(struct) == ile doğrudan karşılaştıramazsınız; bu nedenle, bu operatör genellikten yoksundur

Genellik(Generality)

- Örnek: sabitler(constants)
  - Pascal, sabitlere atanan değerin ifadelerle hesaplanmasına izin vermezken Ada, tamamen genel bir sabit bildirim özelliğine sahiptir

Ortogonalite(Orthogonality)

- Gerçekten ortogonal olan bir dilde, yapılar farklı bağlamlarda farklı davranmazlar.
  - Bağlama bağlı kısıtlamalar ortogonal değildir, bağlamdan bağımsız olarak uygulanan kısıtlamalar genellik eksikliği sergiler
- Örnek: fonksiyon dönüş türleri
  - Pascal, dönüş değerleri olarak yalnızca skaler(scalar) veya işaretçi türlerine izin verir
  - C ve C ++, dizi türleri dışındaki tüm veri türlerinin değerlerine izin verir
  - Ada ve Python tüm veri türlerine izin verir

Ortogonalite(Orthogonality)

- Örnek: değişken bildirimlerinin yerleştirilmesi:
  - C, yerel değişkenlerin yalnızca bir bloğun başlangıcında tanımlanmasını gerektirir
  - C++, kullanımdan önce bir blok içindeki herhangi bir noktada değişken tanımlara izin verir
- Örnek: **basit(primitve)** ve **referans(reference)** türleri
  - Java'da, basit türler **değer semantiğini(value semantics)** kullanır (değerler atama sırasında kopyalanır), nesne türleri (veya referans türleri) **referans semantiğini(reference semantics)** kullanır (atama aynı nesneye iki referans üretir)

Ortogonalite(Orthogonality)

- Ortogonallik, Algol68'in önemli bir tasarım hedefiydi
  - Yine de yapıların tüm anlamlı şekillerde birleştirilebildiği bir dilin en iyi örneğidir

Tekdüzelik(Uniformity)

- **Tekdüzelik**: dil yapılarının görünüm ve davranışlarının tutarlılığını ifade eder
- Örnek: fazladan noktalı virgül
  - C++, bir sınıf tanımından sonra noktalı virgül gerektirir, ancak bir işlev tanımından sonra kullanılmasını yasaklar
- Örnek: bir değer döndürmek için atama kullanma
  - Pascal, işlevin değerini döndürmek için bir atama deyiminde işlev adını kullanır
    - Kafa karıştırıcı bir şekilde standart bir atama ifadesine benziyor
  - Diğer diller bir dönüş ifadesi(return statement) kullanır

Düzensizliklerin Nedenleri

- Birçok düzensizlik, dil tasarımının zorluklarına ilişkin örnek olay incelemeleridir
- Örnek: C++ 'da ekstra noktalı virgül problemi, C ile uyumlu olma ihtiyacının bir yan ürünüydü
- Örnek: Java'daki basit türlerin ve referans türlerinin düzensizliği, tasarımcının verimlilik konusundaki endişesinin bir sonucudur
- Belirli bir hedefe çok fazla odaklanmak mümkündür
- Örnek: Algol68, genellik ve ortogonalite hedeflerine ulaştı, ancak bu biraz belirsiz ve karmaşık bir dile yol açtı

Güvenlik(Security)

- Belirli özelliklere kısıtlamalar getirilmezse güvenilirlik etkilenebilir
  - Pascal: güvenlik sorunlarını azaltmak için işaretçiler sınırlandırılmıştır
  - C: işaretçiler çok daha az kısıtlıdır ve bu nedenle kötüye kullanıma ve hataya daha yatkındır
  - Java: işaretçiler tamamen ortadan kaldırıldı (nesne tahsisinde örtülüdürler), ancak Java daha karmaşık bir çalışma zamanı ortamı gerektirir
- **Güvenlik(Security)**: güvenilirlikle yakından ilgilidir

Güvenlik(Security)

- Güvenlik göz önünde bulundurularak tasarlanmış bir dil:
  - Programlama hatalarını zorlaştırır
  - Hataların keşfedilmesine ve raporlanmasına izin verir
- Güvenlik endişesinden kaynaklanarak türler, tür denetimi ve değişken bildirimleri ortaya çıktı
- Güvenliğe özel odaklanma, bir dilin açıklayıcılığını(expressiveness) ve özlülüğünü(conciseness) tehlikeye atabilir
  - Tipik olarak programcıyı, kodda mümkün olduğunca çok şeyi zahmetli bir şekilde belirtmeye zorlar

Güvenlik(Secuirty)

- ML ve Haskell, güvenli olmaya çalışan ancak maksimum ifade ve genellik sağlayan işlevsel dillerdir.
  - Çok türlü nesnelere izin verirler, bildirim gerektirmezler ve yine de statik tip denetimi gerçekleştirirler
- **Anlamsal olarak güvenli(semantically safe)**: bir programcının dil tanımını ihlal eden herhangi bir ifade(expression) veya komutu(statement) derlemesini veya yürütmesini engelleyen diller
  - Örnekler: Python, Lisp, Java

Genişletilebilirlik(Extensibility)

- **Genişletilebilir dil**: Kullanıcının kendisine özellikler eklemesine izin veren bir dil
- Örnek: yeni veri türleri ve yeni işlemler (fonksiyon veya prosedürler) tanımlama yeteneği
- Örnek: dilin yerleşik özelliklerini genişleten yeni sürümler
- Çok az dil söz dizimi ve semantik eklemelerine izin verir
  - Lisp, bir makro aracılığıyla yeni söz dizimi ve semantik sağlar
- **Makro(Macro)**: derlendiğinde diğer standart koda genişleyen bir kod parçasının sözdizimini belirtir

C++: C'nin Nesne Tabanlı Uzantısı

- C++: Bjarne Stroustrup tarafından 1979-80'de Bell Laboratuvarlarında oluşturuldu
- Yeni dilini şu nedenlerden dolayı C'ye dayandırmayı seçti:
  - Esneklik(Flexibility)
  - Verimlilik(Efficiency)
  - Ulaşılabilirlik(Availability)
  - Taşınabilirlik(Portability)
- Simula67 dilinden sınıf yapısını eklemeyi seçti

C++: C'nin Nesne Tabanlı Uzantısı

- C ++ için tasarım hedefleri:
  - Sınıflar, miras ve güçlü tip denetimi şeklinde iyi program geliştirme desteği
  - C ve BCPL seviyesinde verimli çalışma
  - Son derece taşınabilir, kolay uygulanabilir ve diğer araçlarla kolayca arayüzlenebilir(etkileşime girebilir)

C++: İlk Gerçekleştirmeler(Sürümler)

- Sıradan C kodunu oluşturan Cpre adlı bir önişlemci biçiminde 1979-80'de ilk gerçekleştirme
- 1985: Önişlemciyi daha gelişmiş bir derleyici ile değiştirdi (taşınabilirlik için hala C kodu üretiyordu)
  - Derleyicinin adı **Cfront**'du
  - Dil artık C ++ olarak adlandırılıyordu
  - Metotların, tür parametrelerinin ve genel aşırı yüklemenin dinamik bağlanması eklendi

C++: İlk Gerçekleştirmeler(Sürümler)

- C++ için tasarım hedefleri:
  - Mümkün olduğunca C uyumluluğunu korumalı
  - Sıkı bir şekilde pratik deneyime dayanan artan bir gelişime uğramalı
  - Eklenen herhangi bir özellik, çalışma zamanı verimliliğini düşürmemeli veya mevcut programları olumsuz yönde etkilememelidir
  - Herhangi bir programlama stilini zorlamamalı
  - Tip kontrolünü sürdürmeli ve güçlendirmelidir
  - Aşamalar halinde öğrenilebilir olmalı
  - Diğer sistemler ve dillerle uyumluluğu korumalı

C++: Büyüme

- Cpre ve Cfront eğitim amaçlı olarak ücretsiz olarak dağıtıldı ve dile ilgi uyandırdı
- 1986: ilk ticari uygulama
- Dilin başarısı, standart bir dil oluşturmak için uyumlu bir çabanın gerekli olduğunu gösterdi.

C++: Standardizasyon

- C ++ kullanımı hızla arttığı, gelişmeye devam ettiği ve birkaç farklı uygulamaya sahip olduğu için, standardizasyon bir sorundu
- 1989: Stroustrup bir referans el kitabı(reference manual) üretti
- 1990-1991: ANSI ve ISO standartları komiteleri, kılavuzu standardizasyon çabası için temel belge olarak kabul etti
- 1994: standart bir kapsayıcı(container) ve algoritma kitaplığının eklenmesi
- 1998: Önerilen standartlar gerçek ANSI / ISO standardı oldu

C++: Geriye Dönük

- C ++ neden başarılı oldu?
  - Nesne yönelimli tekniklere olan ilginin patlamasıyla aynı zamanda tanıtıldı
  - Herhangi bir işletim ortamına bağlı olmayan basit sözdizimi
  - Semantiği performans cezasına maruz kalmadı
  - Esnekliği, karma yapısı ve tasarımcısının özelliklerini genişletme isteği popülerdi
- Kötüleyenler, C++ 'nın çok fazla özelliğe ve benzer şeyler yapmanın birçok yoluna sahip olduğunu düşünüyor

Python: Genel Amaçlı Bir Betik[1] Dili

- Guido van Rossum, 1986'da Python adlı bir betik dili için bir tercüman(translator) ve sanal makine geliştirdi
- Hedeflerinden biri, Python'un C ve Shell gibi sistem dilleri veya Perl gibi betik dilleri arasında bir köprü görevi görmesine izin vermekti
- Konum yerine içerik veya ilişkilendirmeye göre düzenlenmiş nesne koleksiyonlarını temsil etmek için yararlı olan, özetleme(hashing) yoluyla uygulanan bir dizi anahtar/değer çifti içeren bir sözlük dahil edildi

[1] Betik ve programlama dilleri arasındaki farklar

Python: Basitlik, Düzenlilik ve Genişletilebilirlik

- Tasarım hedefleri şunları içerir:
  - Basit bir normal sözdizimi
  - Bir dizi güçlü veri türü ve kitaplığı
  - Acemiler tarafından kullanımı kolay

Python: Etkileşim ve Taşınabilirlik

- Python, genellikle büyük sistemler yazmayan, bunun yerine kısa programlar yazan kullanıcılar için tasarlanmıştır
  - Geliştirme döngüsü, G/Ç işlemleri için minimum ek yük ile anında geri bildirim sağlar
- Python iki modda çalıştırılabilir:
  - Maksimum etkileşim için ifadeler veya komutlar bir Python kabuğunda çalıştırılabilir
  - Dosyalara kaydedilmiş daha uzun betikler halinde oluşturulabilir ve bir terminal komut isteminden çalıştırılabilir

Python: Etkileşim ve Taşınabilirlik

- Taşınabilirliğin tasarım amacına iki şekilde ulaşıldı:
  - Python derleyici, kaynak kodunu bir Python sanal makinesinde(PVM) çalıştırılan makineden bağımsız bayt koduna(byte code) çevirir
  - Uygulamaya özel kitaplıklar, veritabanlarına, ağlara, Web'e, GUI'ye ve diğer kaynaklara ve teknolojilere erişmesi gereken programları destekler

Python: Dinamik Tür ve Parmak Türü

- Python, Lisp ve Smalltalk'ta bulunan dinamik tür mekanizmasını içerir
  - Tüm değişkenler türsüzdür
  - Herhangi bir değişken herhangi bir şeyi adlandırabilir, ancak her şeyin veya değerlerin bir türü vardır
  - Tür denetimi çalışma zamanında gerçekleşir
- Bu, programcı için daha az ek yük ile sonuçlanır
  - Daha az "parmak türü"(finger typing)
  - Programcı, bir kod segmentini çok daha hızlı kurup çalıştırabilir

Python: Geriye Dönük

- Python, büyük veya zaman açısından kritik sistemler için C veya C++ 'ın yerini alması amaçlanmamıştır
- Çalışma zamanı tip denetimi, zaman açısından kritik uygulamalar için uygun değildir
- Statik tip kontrolünün olmaması, büyük bir yazılım sisteminin test edilmesi ve doğrulanmasında bir sorumluluk olabilir
- Acemi veya programcı olmayanlar için kullanım kolaylığı tasarım hedefine büyük ölçüde ulaşılmıştır