PROGRAMLAMA DİLLERİ MODÜLÜ (C++)

EMEL OKKALI

BİLİŞİM TEKNOLOJİLERİ VE YAZILIM DERSİ AMAÇLARI

Bu dersin amacı, öğrencilere bilişim teknolojileri ve yazılım alanında temel yeterlilikler ve beceriler kazandırarak, onları teknoloji ile etkili bir şekilde etkileşim kurabilecek, sorumluluk sahibi, üretken ve dijital vatandaşlar olarak yetiştirmektir.

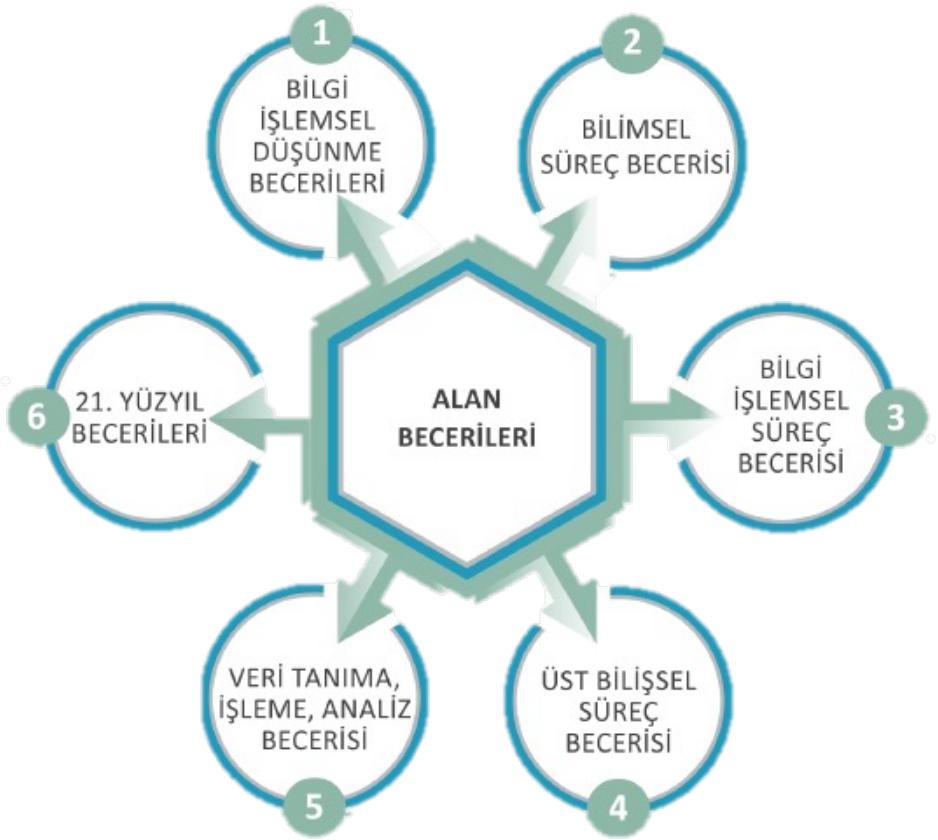
BİLİŞİM TEKNOLOJİLERİ VE YAZILIM DERSİ AMAÇLARI

Bu ders kapsamında öğrencilere;

- Problem çözme,
- Analitik düşünme,
- Proje geliştirme,
- Ekip çalışması

Gibi becerilerin edindirilmesi hedeflenmiştir. Bunun yanı sıra, yerli teknoloji üretiminin önemine dair farkındalık oluşturmak ve geleceğin teknolojilerine adapte olabilecek yetkinlikler kazandırmakta bu dersin amaçlarından biridir.

BİLİŞİM TEKNOLOJİLERİ VE YAZILIM DERSİ ALAN BECERİLERİ



Bilgiyi İşlemsel Düşünme Becerisi

Bilgi işlemsel düşünme becerisi, problemleri çözme, sistemleri tasarlama ve insan davranışını anlamada kullanılan bir yaklaşımdır.

Bilgi işlemsel düşünme, aşağıdaki temel unsurlardan oluşur:

- **ALGORİTMİK DÜŞÜNME**: Bir problemi çözmek için adım adım talimatlar (algoritmalar) geliştirebilme yeteneği. Bu, problemi daha küçük parçalara ayırma (dekompozisyon), benzer problemleri tanıma (desen tanıma), ve genel çözümler geliştirme (soyutlama) süreçlerini içerir.
- **SOYUTLAMA**: Karmaşık bir sistemi veya problemi, üzerinde çalışılabilir hale getirmek için daha basit veya daha genel bir formda ifade etme yeteneği. Bu, önemsiz detaylardan arındırarak ve temel prensiplere odaklanarak problemleri genellemeyi içerir.

Bilgiyi İşlemsel Düşünme Becerisi

- **OTOMASYON**: Problemleri çözmek veya görevleri tamamlamak için bilgisayar veya başka araçları kullanma. Bu, algoritmaları ve soyutlamaları kullanarak görevleri otomatikleştirmeyi ve böylece verimliliği artırmayı amaçlar.
- **HATA AYIKLAMA VE TEST ETME**: Bir sistemin veya algoritmanın istenilen şekilde çalışıp çalışmadığını kontrol etme ve hataları (bugluar) bulup düzeltme süreci. Bu, sistemli bir yaklaşımla sorunların kökenini belirleyip çözüm yolları geliştirmeyi içerir.
- **YİNELEME**: Bir çözümü veya süreci sürekli olarak geliştirme. Bu, bir problem üzerinde çalışırken, elde edilen sonuçları değerlendirme ve bu sonuçları kullanarak süreci veya ürünü iyileştirme çabasını ifade eder.

Bilgi işlemsel düşünme becerisi, öğrencilere ve profesyonellere yalnızca teknolojiyle ilgili sorunları değil, aynı zamanda günlük yaşamın çeşitli yönleriyle ilgili problemleri de çözme konusunda yardımcı olabilir. Eğitimde, bu becerinin öğrencilere öğretilmesi, onların analitik düşünme, problem çözme ve yaratıcı düşünme yeteneklerini geliştirmelerine yardımcı olur.

Bilimsel Süreç Becerisi

Bilimsel süreç becerisi, bilimsel yöntemleri kullanarak soruları yanıtlama ve problemleri çözme yeteneğidir. bu beceriler, bilim insanlarının doğayı ve evreni anlamak için kullandıkları süreçlerin bir parçasıdır ve öğrencilerin bilimsel düşünme yeteneklerini geliştirmek için eğitimde de önemlidir. bilimsel süreç becerileri şunları içerir:

- **GÖZLEM YAPMA**: Doğal dünyayı dikkatli bir şekilde inceleme ve hakkında notlar alma.
- **SINIFLANDIRMA**: Nesneleri veya olayları gruplara ayırma ve düzenleme.
- **ÖLÇME**: Nesnelerin veya olayların miktarını belirleme ve karşılaştırma.
- **TAHMİN YAPMA**: Gelecekteki olaylar hakkında bilgiye dayalı tahminlerde bulunma.

Bilimsel Süreç Becerisi

- **SORU SORMA**: Bilimsel araştırma yönünü belirleyecek meraklı sorular geliştirme.
- **HİPOTEZ OLUŞTURMA**: Gözlemlere dayalı olarak test edilebilir tahminler yapma.
- **DENEY YAPMA**: Hipotezleri test etmek için kontrollü deneyler tasarlama ve uygulama.
- **VERİ TOPLAMA**: Deneylerden veya gözlemlerden elde edilen bilgileri toplama.
- **SONUÇ ÇIKARMA**: Toplanan verileri analiz etme ve hipotezlerin doğruluğunu değerlendirme.
- **İLETİŞİM**: Bulgu ve sonuçları başkalarıyla paylaşma.

ÖRNEK

- **SORUNUN TANIMLANMASI**

Öğrencilere, okul içinde sık karşılaşılan bir problemi çözmek için bir yazılım uygulaması geliştirmeleri istenir. örneğin, kütüphane kitaplarının takibini kolaylaştıracak bir sistem geliştirmek.

- **HİPOTEZ OLUŞTURMA**

Öğrenciler, sorunu çözmek için çeşitli çözüm yollarını tartışır ve hangi teknolojinin (örneğin, bir veritabanı uygulaması) bu problemi en etkili şekilde çözebileceği konusunda hipotezler oluşturur.

- **DENEYLERİN YAPILMASI VE VERİ TOPLANMASI**

Öğrenciler, seçtikleri çözümü uygulamaya başlarlar. Bu, programlama dillerini kullanarak bir kütüphane yönetim sistemi yazmayı içerebilir. Geliştirme sürecinde, uygulamanın kullanıcı arayüzü, veritabanı işlemleri ve kullanıcı etkileşimleri gibi çeşitli bileşenler üzerinde çalışırlar.

- **SONUÇLARIN ANALİZİ**

Geliştirilen uygulamanın test edilmesi, burada öğrenciler yazılımlarını gerçek dünya verileriyle deneyerek ve potansiyel hataları (bugları) ayıklayarak uygulamanın beklentileri karşılayıp karşılamadığını değerlendirirler.

- **SONUÇLARA DAYANARAK SONUÇ ÇIKARMA**

Test sonuçlarına dayanarak, öğrenciler uygulamanın etkinliğini değerlendirir ve ilk hipotezlerini test ederler. Örneğin, uygulama kütüphane kitap takibini gerçekten kolaylaştırıyor mu? Eğer uygulama etkiliyse, nasıl iyileştirilebileceği veya genişletilebileceği üzerine öneriler geliştirilir. Eğer uygulama yetersizse, sorunların nedenlerini araştırırlar ve çözüm yolları önerirler.

Bilgi İşlemsel Süreç Becerisi

Bilgi işlemsel süreç becerisi, bireylerin teknoloji ve bilgisayar sistemlerini kullanarak bilgiyi toplama, analiz etme ve sorun çözme süreçlerinde uyguladıkları bir dizi yetenek ve metodolojiyi ifade eder.

Bilgi işlemsel süreç becerisine örnek olarak, bir sosyal medya analiz projesi ele alınabilir. Bu proje, özellikle sosyal medya platformlarında kullanıcı davranışlarını, trendleri ve duyarlılıkları analiz etmeyi amaçlayabilir.

Üst Bilişsel Süreç Becerisi

Üst bilişsel süreç becerisi, kendi düşünme süreçlerini bilinçli olarak izleme, değerlendirme ve düzenleme yeteneğidir. Bu beceri, bireylerin öğrenme süreçlerini daha etkili hale getirmelerine, problem çözme yeteneklerini geliştirmelerine ve karar verme süreçlerini iyileştirmelerine yardımcı olur. Üst bilişsel süreç becerisi üç ana bileşeni içerir:

- **ÖZ-DÜZENLEME**: Öğrenme sürecinin aktif bir şekilde yönetilmesi, hedef belirleme, strateji geliştirme, zamanı yönetme ve motivasyonu sürdürme yeteneğidir. Bu, aynı zamanda öğrenme sırasında dikkati kontrol etme ve dağıtıcı unsurlardan kaçınma becerisini de kapsar.
- **ÖZ-DÜŞÜNME (ÖZ-MONİTÖRLEME):** Kendi düşünme, öğrenme ve anlama süreçlerini sürekli olarak izleme ve değerlendirme yeteneğidir. Bireyler, bu süreçte kendi anlayışlarını ve kavrayışlarını sorgular, hataları ve eksiklikleri tanır ve gerektiğinde düzeltici önlemler alır.
- **ÖZ-DEĞERLENDİRME:** Bireyin kendi performansını ve öğrenme sürecinin sonuçlarını değerlendirme yeteneğidir.

Bu, hedeflere ulaşma derecesini değerlendirme, kendi çalışmalarını geriye dönük olarak analiz etme ve gelecekteki öğrenme ve problem çözme çabalarını yönlendirecek içgörüler elde etme sürecini içerir.

ÖRNEK

Bir matematik problemi üzerinde çalışan bir öğrenciyi düşünelim:

- **ÖZ-DÜZENLEME**: Öğrenci, problemi çözmek için bir plan yapar ve hangi matematiksel yöntemlerin kullanılacağını belirler. Çalışma süresini planlar ve tüm dikkatini problemin çözümüne odaklar.
- **ÖZ-DÜŞÜNME**: Problem çözümü sırasında öğrenci, kendi anlama sürecini izler ve problemi doğru anlayıp anlamadığını kontrol eder. Bir adımda takılıp kalmışsa, nedenini düşünür ve farklı bir yaklaşım denemeye karar verir.
- **ÖZ-DEĞERLENDİRME**: Problemi çözdükten sonra, öğrenci çözüm sürecini ve sonucunu değerlendirir. Doğru çözüme ulaşıp ulaşmadığını kontrol eder ve süreçte neyin iyi neyin kötü gittiğini düşünür. Aynı türden problemleri gelecekte daha etkili bir şekilde çözebilmek için elde ettiği içgörüleri değerlendirir.

Veri Tanıma, İşleme, Analiz Becerisi

Veri tanıma, işleme ve analiz becerisi, büyük veri setlerinden anlamlı bilgiler çıkarmak için gereken teknik ve analitik yeteneklerin bir kombinasyonunu ifade eder. Bu beceri seti, verilerin toplanması, temizlenmesi, düzenlenmesi, analiz edilmesi ve sonuçların yorumlanması süreçlerini kapsar.

VERİ TANIMA: Veri setlerinin içeriğini, yapısını ve potansiyel kalite sorunlarını anlama sürecidir.

- Veri türlerini (sayısal, kategorik) ve yapılarını (zaman serileri, çapraz kesit) anlamak.
- Eksik veriler, aykırı değerler ve tutarsızlıklar gibi kalite sorunlarını tanımlamak.

VERİ İŞLEME: Verileri analize hazır hale getirmek için gerekli ön işlemleri yapma sürecidir.

- **Verileri temizleme:** Eksik verileri doldurma, aykırı değerleri düzeltme veya kaldırma.
- **Veri entegrasyonu:** Farklı kaynaklardan gelen verileri birleştirme.
- **Veri dönüşümü:** Verileri analize uygun forma dönüştürme (örneğin, log dönüşümü).

Veri Tanıma, İşleme, Analiz Becerisi

VERİ ANALİZİ: Veriler üzerinde çeşitli istatistiksel, matematiksel veya makine öğrenmesi tekniklerini uygulayarak anlamlı bilgiler, trendler ve desenler çıkarma sürecidir.

- **Keşifsel veri analizi:** Veri setlerinin temel özelliklerini anlamak için görselleştirme ve basit istatistikler kullanma.
- **İstatistiksel testler ve modelleme:** Hipotez testleri, regresyon analizi gibi yöntemlerle veriler arasındaki ilişkileri test etme.
- **Makine öğrenmesi:** Sınıflandırma, kümeleme ve tahmin gibi görevler için algoritmalar kullanma.

ÖRNEK

PROJE TANIMI: OKUL KÜTÜPHANESİ KİTAP ÖNERİ SİSTEMİ

AMAÇ: Okul kütüphanesindeki kitapların ödünç alınma kayıtlarını analiz ederek, öğrencilere ilgi alanlarına uygun kitap önerileri sunan bir sistem geliştirmek.

VERİ TANIMA

- **Veri seti**: Kitapların ödünç alınma kayıtları, kitapların kategorileri, yazar bilgileri ve öğrencilerin önceki ödünç alma geçmişi.
- **Görev**: Öğrenciler, veri setinin yapısını ve içeriğini anlarlar. Hangi verilerin önemli olduğunu, eksik veya hatalı verilerin olup olmadığını belirlerler.

ÖRNEK

VERİ İŞLEME

- **Veri temizleme**: Eksik verilerin doldurulması, hatalı kayıtların düzeltilmesi.
- **Öznitelik mühendisliği**: Kitapların popülerliğini, öğrencilerin kitap tercih trendlerini analiz edecek yeni veri sütunlarının (özniteliklerin) oluşturulması.

VERİ ANALİZİ

- **Keşifsel veri analizi (KVA)**: Kitap ödünç alma trendlerinin, popüler kategorilerin görselleştirilmesi.
- **Makine öğrenmesi**: Öğrenci profillerine ve geçmiş ödünç alma verilerine dayanarak kitap önerileri yapacak basit bir makine öğrenmesi modelinin eğitilmesi.

ÖRNEK

PROJE UYGULAMASI

Veri Setinin Hazırlanması: Öğrenciler, kütüphane veri tabanından kayıtları çeker ve bir veri çerçevesine dönüştürürler.

Veri İşleme: Python kullanarak veri temizleme ve öznitelik mühendisliği işlemleri yapılır.

Analiz ve Modelleme: Python'daki pandas, matplotlib ve scikit-learn gibi kütüphaneler kullanılarak KVA yapılır ve kitap öneri modeli eğitilir.

Sonuçların Değerlendirilmesi: Modelin öneri performansı test edilir ve öğrencilere kitap önerileri sunulur.

PROJENİN EĞİTİMSEL FAYDALARI

Bu proje, öğrencilere veri bilimi ve makine öğrenmesi projelerinde gereken temel becerileri kazandırırken, aynı zamanda bilgi işlemsel düşünme, problem çözme, algoritma tasarımı ve kodlama becerilerini de geliştirir. Öğrenciler, gerçek dünya verileriyle çalışmanın yanı sıra, elde ettikleri bilgileri anlamlı ve pratik bir çözüme dönüştürmenin tatminini yaşarlar.

21.Yüzyıl Becerileri

21.Yüzyıl becerileri, bireylerin hızla değişen, teknoloji odaklı ve küreselleşen dünyada başarılı olabilmeleri için gereken yetenekler ve yeterlilikler topluluğunu ifade eder. Bu beceriler, öğrencilerin, işgücünün ve genel olarak toplumun, mevcut ve gelecekteki zorluklarla başa çıkabilmesi ve fırsatlardan yararlanabilmesi için kritik öneme sahiptir. 21. Yüzyıl becerileri genellikle üç ana kategori altında toplanır:

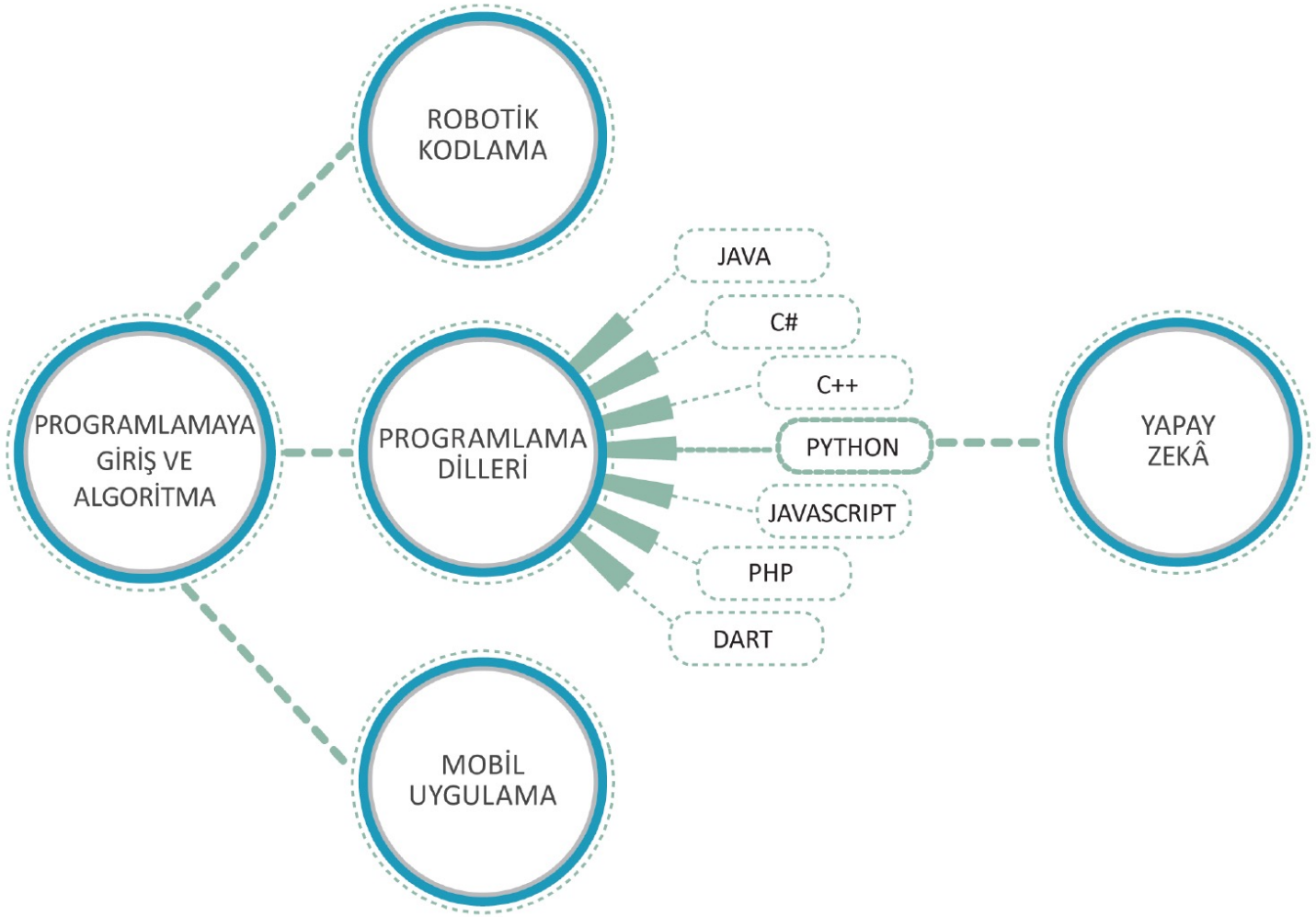
- Öğrenme ve inovasyon becerileri,
- Dijital okuryazarlık becerileri ,
- Yaşam ve kariyer becerileri.

DERS SEÇİM KRİTERİ

Programlamaya giriş ve algoritma modülü alınmadan; robotik kodlama, mobil uygulama geliştirme ve programlama dilleri modülleri seçilemez. Programlama dilleri modülü alınmadan yapay zekâ uygulamaları modülü seçilemez.

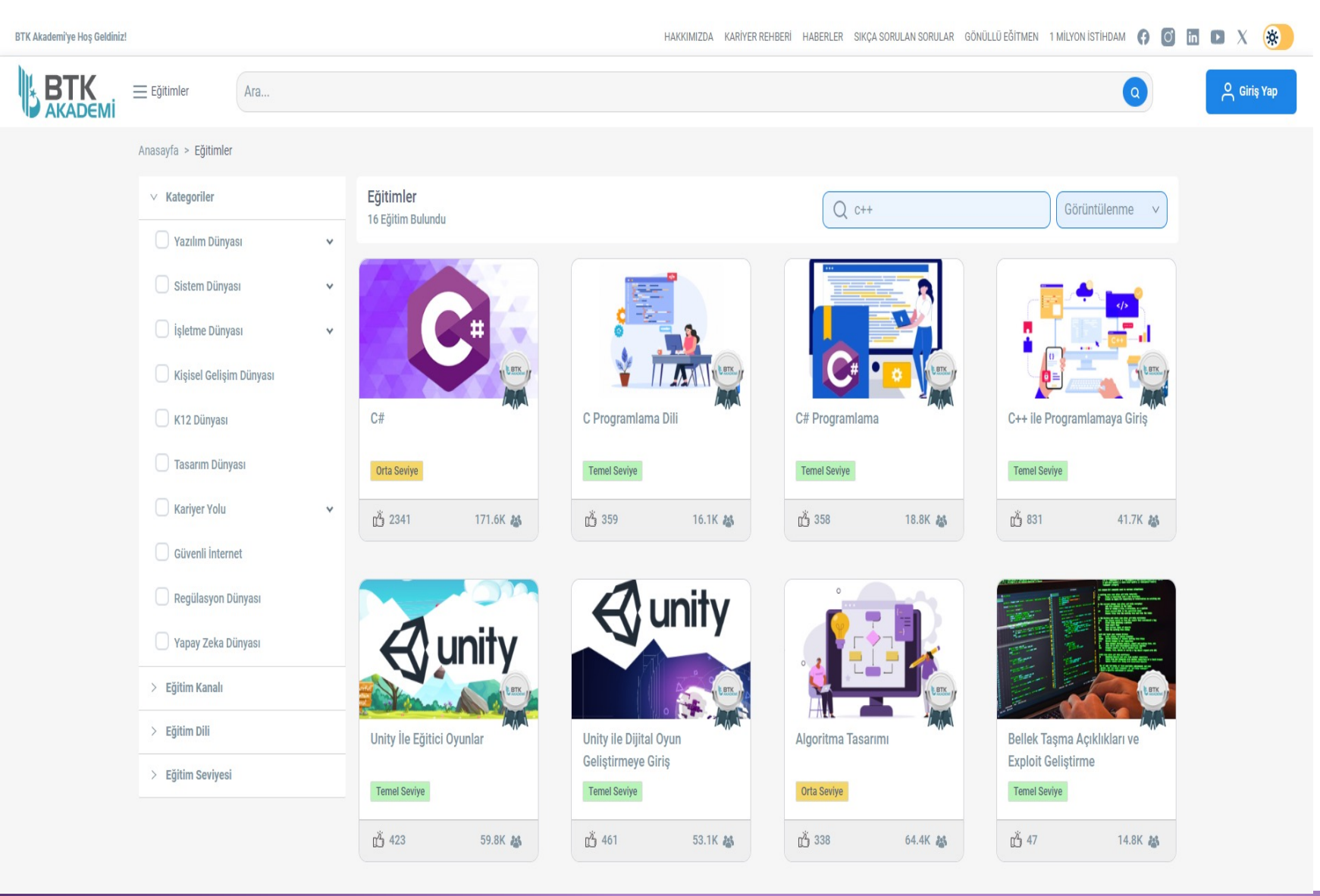
Programlama dilleri modülünde; Java, C++, C#, Dart, PHP, Javascript, Python programlama dillerinden birisi seçilir.

DERS SEÇİM KRİTERİ ŞEMASI

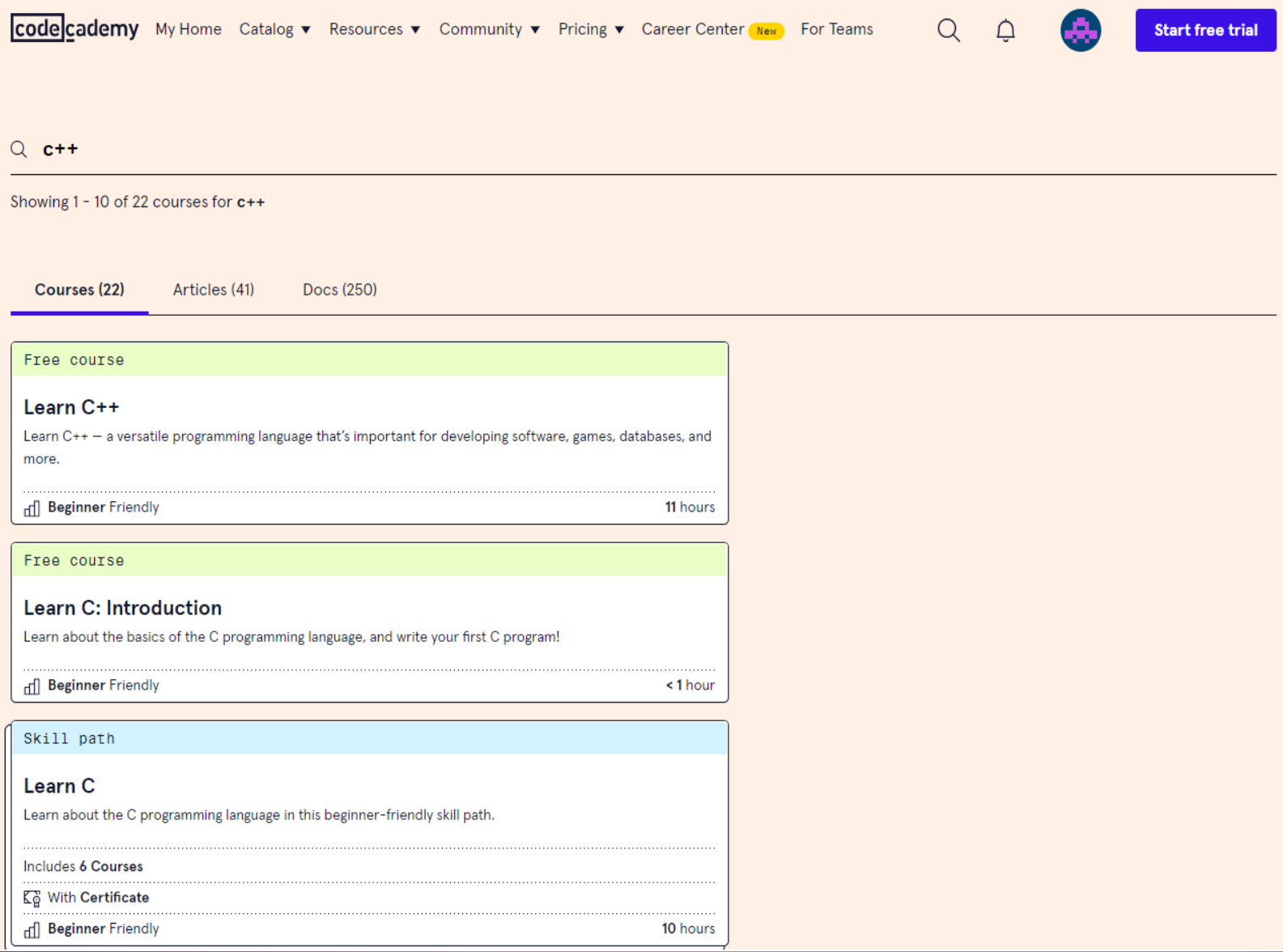


MODÜL	ÜNİTE	KAZA-NIM SAYISI	DERS SAATİ	YÜZDE ORANI (%)
PROGRAMLAMA DİLLERİ (C++)	1. Ünite: C++ Çalışma Ortamı	3	14	20
	2. Ünite: Karar ve Döngü Yapıları	3	14	20
	3. Ünite: Fonksiyonlar ve Diziler	4	16	20
	4. Ünite: Formlar	3	14	20
	5. Ünite: Veri Tabanı İşlemleri	6	14	20

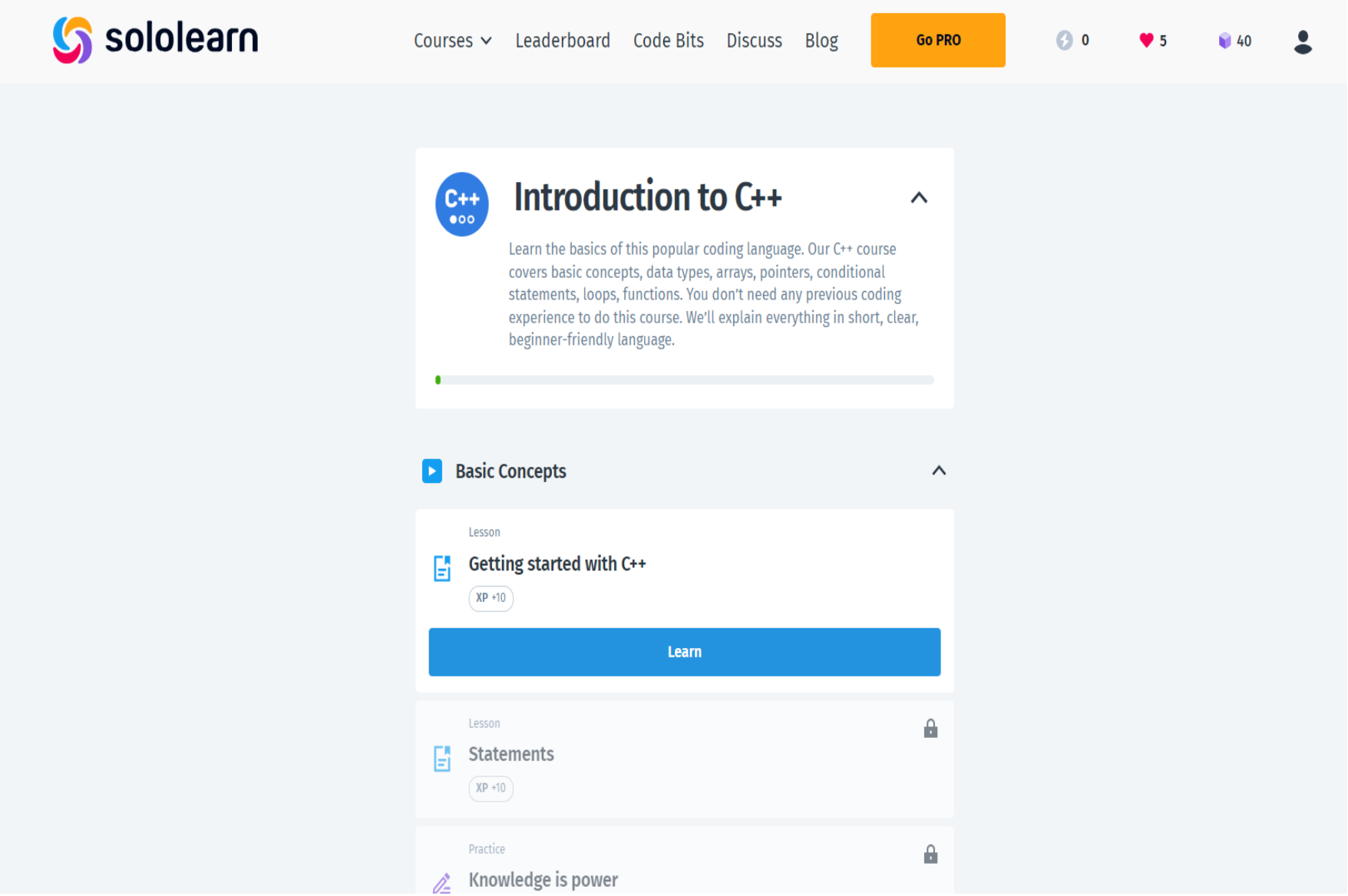
ÖĞRENME KAYNAKLARI



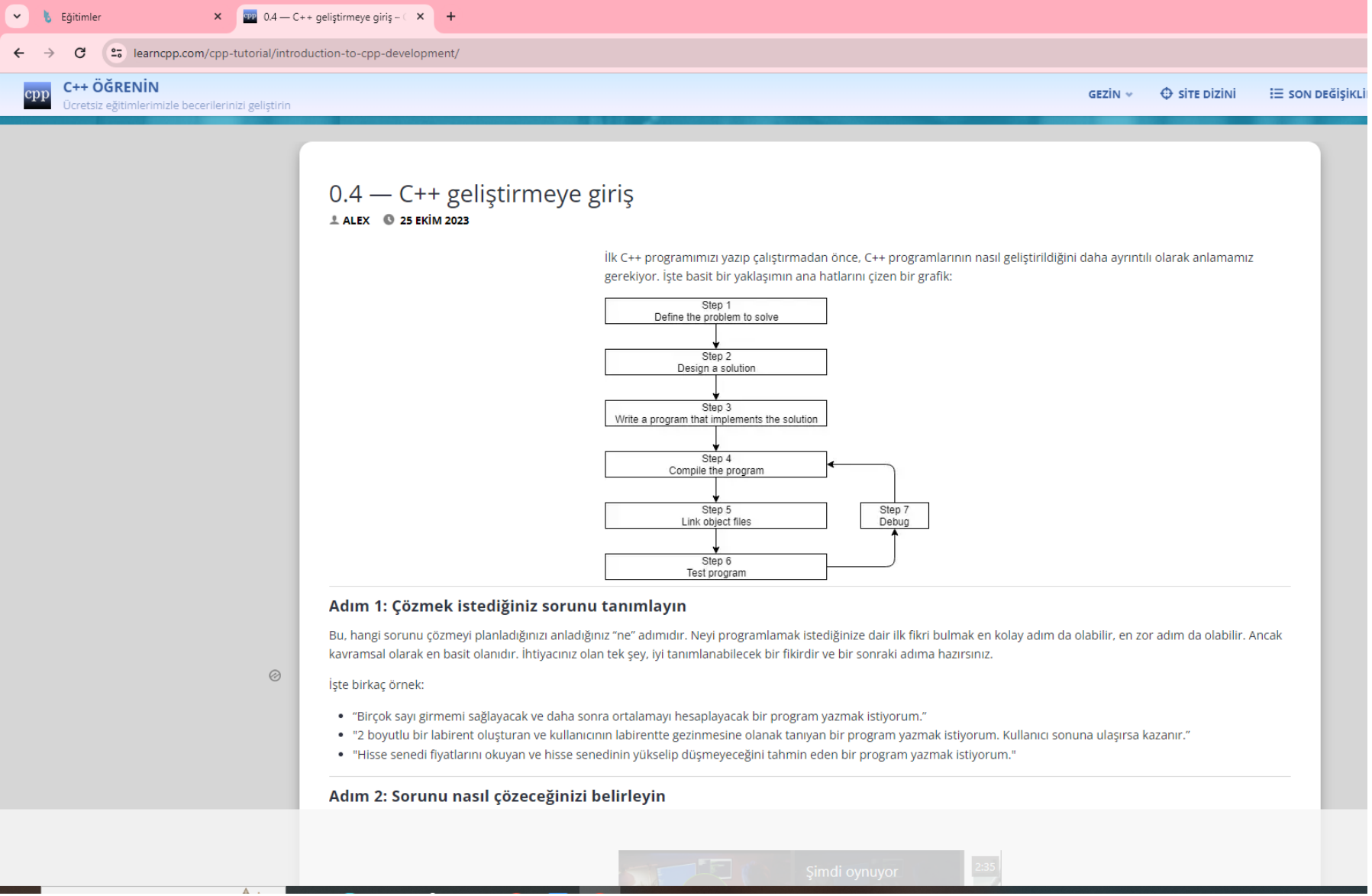
BTK Akademi: Türkiye'de Bilgi Teknolojileri ve İletişim Kurumu tarafından sunulan ücretsiz eğitim platformudur. Programlama, siber güvenlik, veri bilimi gibi birçok alanda kurslar sunmaktadır.



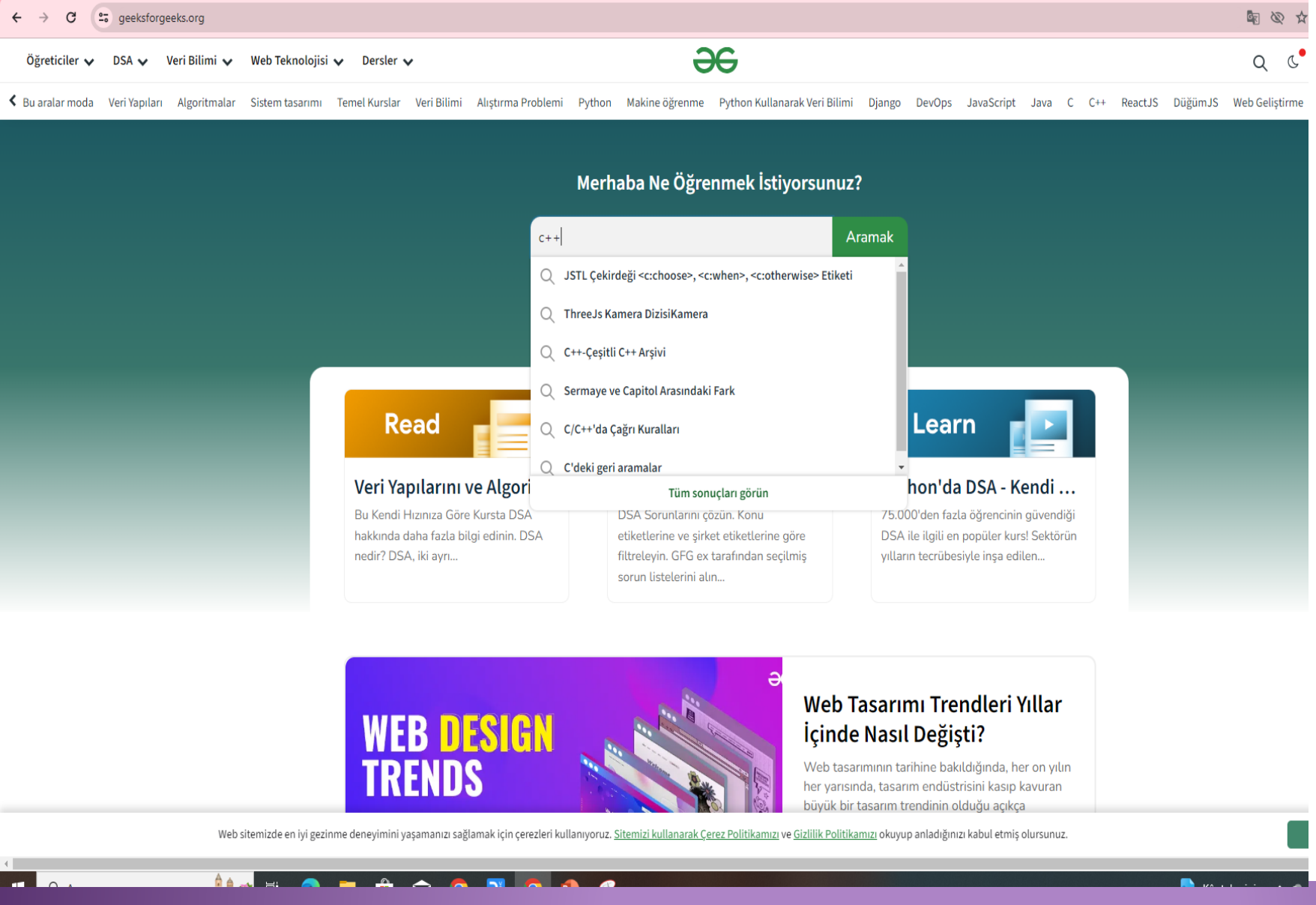
Codecademy: Kullanıcılara çeşitli programlama dilleri ve teknoloji konularında interaktif kurslar sunan bir online öğrenme platformudur. Kullanıcılar, web geliştirme, veri bilimi, bilgisayar bilimi ve daha pek çok alanda temel ve ileri düzey becerileri geliştirebilirler. Kendi hızlarında öğrenme imkanı sunan Codecademy, pratik yaparak ve gerçek projeler üzerinde çalışarak öğrenmeyi teşvik eder.



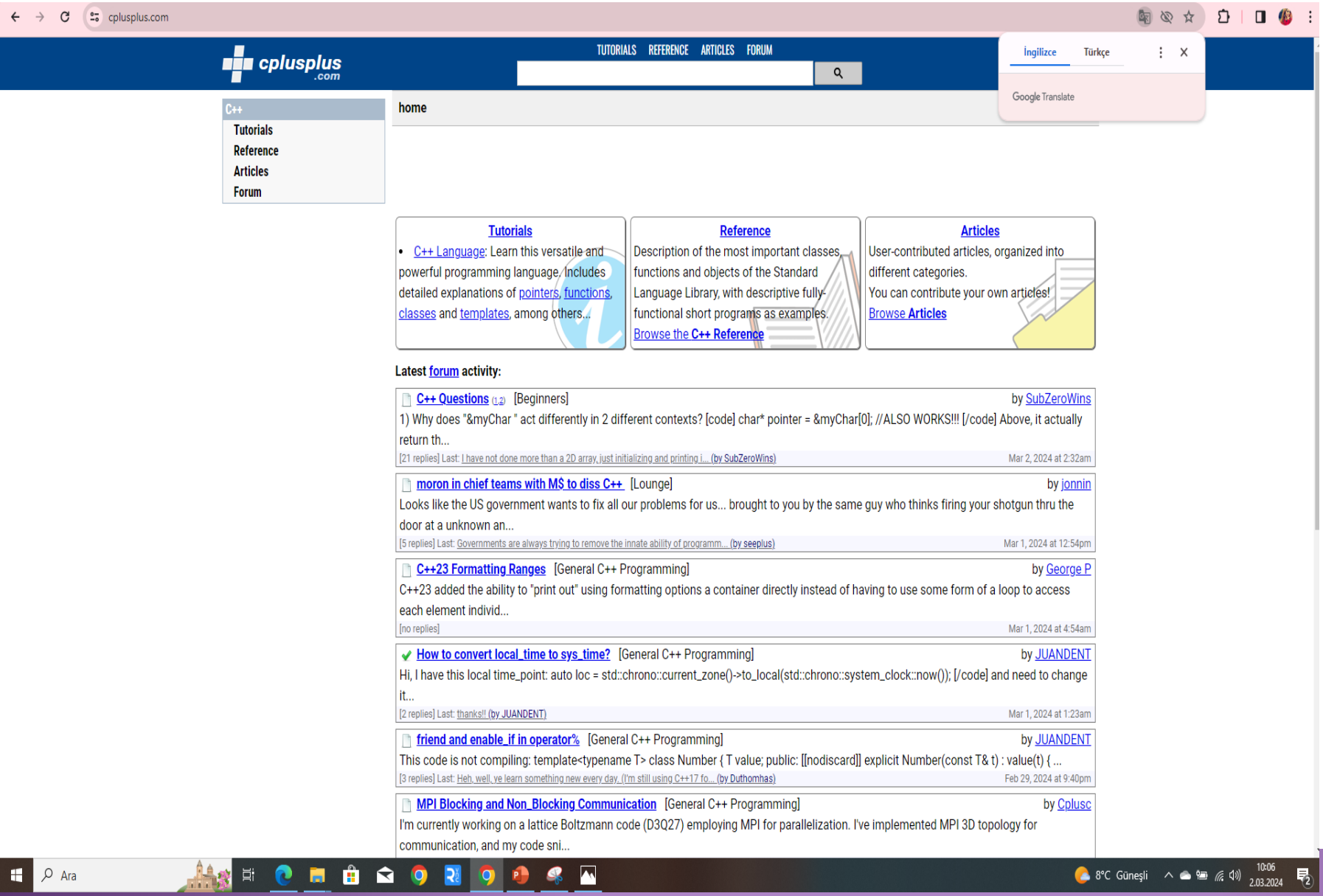
Sololearn: Mobil ve web platformunda erişilebilen SoloLearn, kullanıcıların kendi hızlarında öğrenebilecekleri C++ dahil birçok programlama dilini sunar.



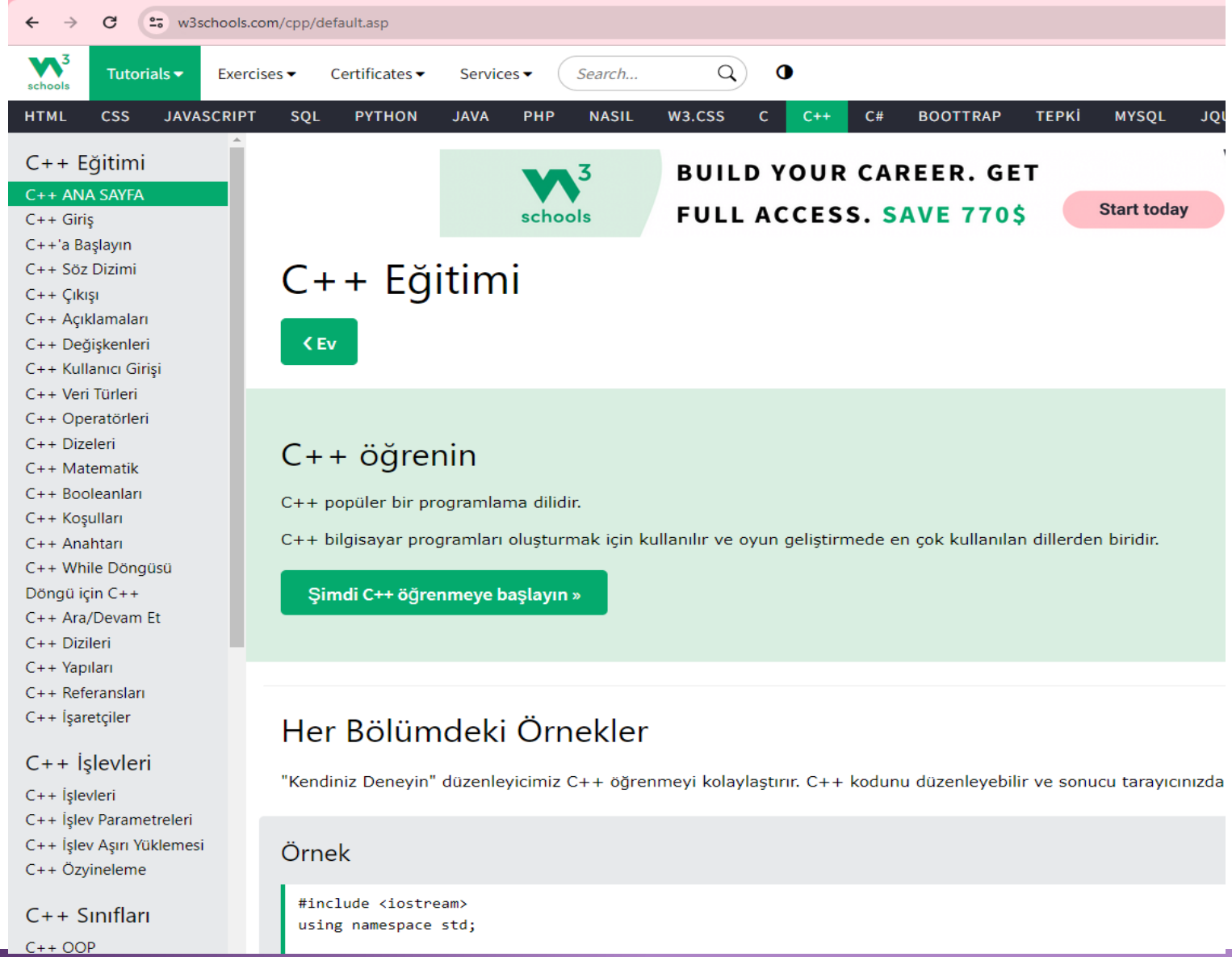
LearnCpp.com: C++ öğrenmek isteyenler için kapsamlı bir kaynaktır. Temelden ileri seviyeye kadar konuları kapsayan dersler sunar.



GeeksforGeeks: Temel programlama konseptlerinden ileri seviye algoritmalar ve problem çözümlerine kadar geniş bir içerik yelpazesi sunar. C++ için özel bölümler bulunmaktadır.



Cplusplus.com: C++'ın temellerinden ileri seviye konularına kadar geniş bir dökümantasyon sunar. Ayrıca bir forum bölümü de bulunmaktadır, burada sorularınızı sorabilir ve diğer programcılarla etkileşimde bulunabilirsiniz.



W3Schools: Temel programlama dilleri ve web geliştirme teknolojileri üzerine odaklanan W3Schools, C++ için de temel bir giriş sağlar.

KİTAPLAR

1. **"C++ primer" (stanley B. Lippman, josée lajoie, ve barbara E. Moo)**: C++11 standartlarını da içerecek şekilde güncellenmiş bu kitap, yeni başlayanlar için mükemmel bir kaynaktır.
2. **"Effective c++" (scott meyers)**: bu kitap, c++ programlama dilinde daha etkili programlama yapmak için 55 önemli kural sunar. Orta ve ileri seviye programcılar için uygundur.
3. **"The c++ programming language" (bjarne stroustrup)**: c++'ın yaratıcısı tarafından yazılan bu kitap, dilin en kapsamlı ve ayrıntılı açıklamalarından birini sunar. İleri seviye öğrenciler ve profesyoneller için idealdir.

PROGRAMLAMA DİLLERİ C++ DERSİ AMACI

C++ programlama dilinde, temel çalışma ortamı, karar - döngü yapıları, fonksiyonlar, diziler ve koleksiyonlar, form oluşturma ile veri tabanı işlemleri gibi konuları içeren becerilerin kazandırılması ve ayrıntılı biçimde ele alınması amaçlanmaktadır.

C++

ÜNİTE - KAZANIM VE AÇIKLAMALARI

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

1.1. C++ Programlama Dilini Tanır.

A) C++ programlama dilinin doğuş süreci ve neden ihtiyaç duyulduğu söylenir.

C++'ın önemli olmasının ana nedenlerinden birkaç tanesi şunlardır:

- **Yüksek performans**: C++, derlenmiş bir dil olması nedeniyle, özellikle hızın kritik olduğu sistemlerde (işletim sistemleri, gömülü sistemler) tercih edilir.
- **Nesne yönelimli programlama (oop) desteği**: Kodun yeniden kullanılabilirliğini, organizasyonunu ve bakımını kolaylaştırır.
- **Geniş kullanım alanları**: Masaüstü uygulamaları, sunucu (backend) uygulamaları, oyun geliştirme, ve daha birçok alanda kullanılır.
- **Büyük standart kütüphane**: Standart kütüphanesi, birçok işlevsel ve kullanışlı kütüphane ve aracı içerir, bu da geliştirme sürecini hızlandırır.

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

Günlük Hayatta Kullanıldığı Dünya Çapındaki Projeler

C++, birçok önemli ve büyük ölçekli projede kullanılmaktadır. İşte bazı örnekler:

İşletim sistemleri: Microsoft windows'un çekirdek bileşenleri, linux'un bazı parçaları ve macos'un çekirdek bileşenleri C++ kullanılarak geliştirilmiştir.

Tarayıcılar: Google chrome'un tarayıcı motoru olan chromium, mozilla firefox ve microsoft edge gibi popüler web tarayıcılarının önemli parçaları c++ ile yazılmıştır.

Oyun geliştirme: C++, unreal engine ve cryengine gibi oyun motorlarında kullanılmakta ve bu motorlar, dünya çapında popüler birçok video oyununun geliştirilmesinde kullanılmaktadır (örneğin, fortnite, counter-strike).

Finans sektörü: Yüksek frekanslı ticaret sistemleri gibi finansal işlemlerin hızının kritik önem taşıdığı sistemlerde c++ tercih edilir.

Veritabanları: Mysql, mongodb ve birçok önemli veritabanı yönetim sistemi, performansı artırmak için c++ kullanılarak geliştirilmiştir.

Yapay zeka ve makine öğrenmesi: Tensorflow gibi bazı makine öğrenmesi kütüphaneleri, performans kritik bileşenlerinde c++ kullanır.

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

B) Genel C++ programlama dili kavramları ve söz dizimi anlatılır.

Temel söz dizimi, veri tipleri, operatörler, kontrol ifadeleri, fonksiyonlar ve nesne yönelimli programlama kavramları gibi temel konulara değinilmesi önemlidir.

Temel Söz Dizimi ve Veri Tipleri

Örnek: C++'ta bir değişken tanımlamak, günlük hayatta bir kutuya etiket yapıştırmak gibidir. Örneğin, bir kutuya "kitaplar" etiketi yapıştırırsınız ve bu kutu artık sadece kitapları içerebilir. C++'ta **int kitapSayisi = 5;** ifadesi, **"kitapSayisi"** adında bir kutuya 5 değerini atar ve bu kutu sadece tam sayı (int) türünde değerler içerebilir.

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

Operatörler ve Kontrol İfadeleri

Örnek: Bir karar vermek gerektiğinde kullanılır. "Eğer yağmur yağıyorsa, şemsiyemi alacağım." gibi.. C++'ta bu, **if(yagmurYagiyor) { semsiyeAl(); }** şeklinde yazılır. Bu, günlük karar verme süreçlerinin kodla ifade etmenin bir yoludur.

Fonksiyonlar

Örnek: Bir yemek tarifi düşünün. Belli adımları takip ederek bir sonuç (yemek) elde edilir. C++'ta fonksiyonlar da benzer şekilde çalışır; belirli bir görevi yerine getiren kod bloklarıdır. **void yapYemek() { /* yemek yapma adımları */ }** fonksiyonu, bir yemek yapma "tarifi" gibi düşünülebilir.

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

1.2. Kod düzenleyici arayüzünde temel işlemleri gerçekleştirir.

A) C++ programlama dili arayüzünde kod yazma sürecinde hata ayıklama, otomatik tamamlama ve belge yönetimi araçları incelenir.

Hata ayıklama

- Hata ayıklamanın, kodda meydana gelen hataları (bug'ları) bulma, tanımlama ve düzeltme süreci olduğunu açıklanabilir.
- Sentez hataları (syntax errors), mantık hataları (logical errors) ve çalışma zamanı hataları (runtime errors) arasındaki farklara değinilir.

Örnek: Bir tarifteki adımları takip ederken yapılan bir hata gibi düşünülebilir. Eğer tarifin adımları doğru izlenmezse, sonuç beklenenin dışında olur. Hata ayıklama süreci, tarifteki hangi adımın yanlış yapıldığını bulup düzeltmeye benzer.

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

Otomatik Tamamlama

- Otomatik tamamlamanın, programcının daha hızlı kod yazmasını sağlayan ve potansiyel hataları azaltan bir araç olduğu açıklanabilir.
- Bu özelliğin, kullanılan kod editörleri veya Geliştirme Ortamları (IDE) tarafından nasıl sağlandığını örneklerle gösterilebilir.

Örnek: Bir akıllı telefonun klavyesindeki kelime önerme özelliğine benzetilebilir. Nasıl ki telefonunuz yazmakta olduğunuz kelimenin tamamını tahmin edip öneride bulunuyorsa, otomatik tamamlama da programlama dillerinde benzer bir işlev görür.

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

Belge Yönetimi

- Kodun, yorum satırları eklenerek ve belgelendirme araçları kullanılarak nasıl daha anlaşılır hale getirilebileceğine değinilir.
- İyi bir belge yönetiminin, hem kişisel projelerde hem de takım projelerinde neden önemli olduğunu vurgulayın.

Örnek: Bir kullanma kılavuzunun bir ürünün nasıl kullanılacağını adım adım açıklamasına benzer. Kodunuzun 'kullanma kılavuzu'nu oluşturmak, sizin ve başkalarının kodu gelecekte daha kolay anlamasını ve kullanmasını sağlar.

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

B) Yeni bir C++ arayüzü projesi açar, proje dosyalarını düzenler, projenin yapılandırması, dosyaların eklenmesi ve düzenlenmesi işlemleri uygulatılır.

Bu kazanım anlatırken, projenin yapılandırılması, dosyaların eklenmesi ve düzenlenmesi gibi temel konseptlere değinmelidir.

Proje Yapılandırması

Bir C++ projesinin temel yapı taşlarını ve bir projenin nasıl organize edildiğini açıklanabilir. Projelerin, kaynak dosyaları (.cpp), başlık dosyaları (.h) ve yapılandırma dosyalarından oluştuğunu belirtin.

Bir projenin derleme ayarlarının, bağımlılıklarının ve hedef platformunun nasıl yapılandırılacağı öğretilir.

Örnek: Bir ev inşa etmeye benzetilebilir. Ev inşa etmek için önce bir plana (proje yapılandırması) ihtiyacınız var, ardından temel yapı taşlarını (dosyaları) yerleştirir ve son olarak evi istediğiniz gibi düzenlersiniz (proje ayarları ve bağımlılıklar).

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

Dosyaların Eklenmesi ve Düzenlenmesi

- Projeye yeni dosyalar eklemenin ve mevcut dosyaları düzenlemenin önemini vurgulanabilir. Dosyaların projeye nasıl dahil edileceği ve organizasyonunun projenin okunabilirliği ve yönetilebilirliği üzerindeki etkisini anlatılabilir.
- Kaynak kodun ve başlık dosyalarının ayrılmasının neden gerekli olduğunu ve bu ayrımın projelerde nasıl yapılacağını açıklanabilir.

Örnek: Bir projeyi, büyük bir okul projesi veya takım çalışması olarak düşünebilirsiniz. Her dosya, projenin bir parçasıdır (örneğin, bir raporun bölümleri). Dosyaları doğru şekilde eklemek ve düzenlemek, takım üyelerinin çalışmayı daha kolay anlamasını ve üzerinde çalışmasını sağlar, tıpkı herkesin projenin hangi bölümü üzerinde çalışacağını bilmeleri gibi.

Pratik Uygulama

Öğrencilere, basit bir "merhaba dünya" C++ projesi oluşturup, bu projeye birkaç dosya ekleyip düzenleme görevi verin. Bu, öğrencilere teorik bilgileri pratikle pekiştirme fırsatı tanır.

Öğrencilere, proje dosyalarının nasıl düzenleneceğini ve proje yapılandırmasının nasıl yapılandırılacağını adım adım gösteren bir rehber veya video sağlayın.

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

C) Yazdığı kodları düzenli bir şekilde yönetmesi, değişiklikleri kaydetmesi ve ardından derleyerek çalışan bir uygulama yapması anlatılır.

Bu kazanımı anlatırken aşağıdaki noktalara değinilebilir:

Kod Yönetimi

Kodların düzenli ve anlaşılır olması gerektiğini vurgulayın. Bu, yorum satırları eklemeyi, kodu mantıklı bölümlere ayırmayı ve anlamlı değişken adları kullanmayı içerir.

Örnek: Kod yönetimini, bir kitabın bölümlerini düzenlemeye benzetebilirsiniz. Tıpkı her bölümün belirli bir konuyu ele alması gibi, kodunuzun da belirli görevleri yerine getiren bölümlere (fonksiyonlara/modüllere) ayrılması gerekir.

Değişikliklerin Kaydedilmesi

Versiyon kontrol sistemlerinin (örneğin, Git) önemini açıklayın. Bu sistemler, kod üzerinde yapılan değişikliklerin kaydedilmesini, geri alınmasını ve farklı versiyonlar arasında geçiş yapılmasını sağlar.

Örnek: Bir belge üzerinde çalışırken "kaydet" ve "farklı kaydet" seçeneklerini kullanmanız, yazdığınız her bölümü veya değişikliği kaydetmenize olanak tanır. Versiyon kontrolü, bu işlemin çok daha gelişmiş bir versiyonudur ve işbirliği yaparken de büyük kolaylık sağlar.

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

Derleme ve Uygulama Yapımı

Kodun nasıl derleneceği ve çalışan bir uygulamaya nasıl dönüştürüleceği konusunda bilgi verin. Derleme sürecinin, yazılan kodun bilgisayarın anlayabileceği bir formata çevrilmesi işlemi olduğunu açıklayın.

Örnek: Bir yemek tarifini takip edip, malzemeleri bir araya getirerek ve belirli adımları uygulayarak bir yemeği hazırlamaya benzetebilirsiniz. Tıpkı tarifteki adımların sonucunda yemeğin ortaya çıkması gibi, derleme süreci de kodunuzun çalışır bir program haline gelmesini sağlar.

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

1.3. Temel veri türlerini ve değişken işlemlerini açıklar.

A) Temel veri türleri (int, string, double vb.) ve farklı tipteki veriler (tam sayılar, metin, ondalık sayılar vb.) incelenir.

Temel Veri Türleri

- **int (Tamsayılar):** Yaş, sınıf mevcudu, bir kitaptaki sayfa sayısı gibi sayılabilir değerler.

Örnek: Bir sınıftaki öğrenci sayısı. Öğrenci sayısı kesirli olamaz, yani tam sayıdır.

- **string (Metin):** İsimler, adresler, herhangi bir metin veya kelime grubu.

Örnek: Bir kişinin adı. Örneğin, "Ahmet" veya "Zeynep" birer metin verisidir.

- **double (Ondalık Sayılar):** Ağırlık, boy, para miktarı gibi kesirli değerler.

Örnek: Bir ürünün kilogram cinsinden ağırlığı. Örneğin, 1.5 kg elma.

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

Farklı Tipteki Veriler

Tam Sayılar ve Ondalık Sayılar Arasındaki Fark: Tam sayılar kesir içermezken, ondalık sayılar virgülden sonra değerler taşıyabilir.

Örnek: Bir pizza 8 dilime bölündüğünde, her bir dilim pizza 1/8 oranında bir kesir değerine karşılık gelir. Ancak, bir sınıftaki öğrenci sayısı gibi tam sayılarla ifade edilir.

Metin ve Sayısal Veriler Arasındaki Fark: Metin verileri harf, rakam ve diğer karakterlerden oluşurken, sayısal veriler matematiksel işlemler için kullanılır.

Örnek: Bir telefon numarası metin olarak saklanabilir çünkü genellikle matematiksel bir işlem yapılmaz. Ancak, iki sayının toplamı gibi işlemler için sayısal veri türleri kullanılır.

Öğretim Stratejileri

Karşılaştırmalı Öğrenme: Farklı veri türlerinin kullanıldığı örnekler üzerinden karşılaştırmalar yaparak öğrencilere somut farkları gösterin.

İnteraktif Uygulamalar: Öğrencilere çeşitli veri türleriyle basit kod parçacıkları yazdırarak uygulamalı öğrenme fırsatı sunun.

Günlük Nesnelerle İlişkilendirme: Veri türlerini öğrencilerin günlük hayatta karşılaştıkları nesne ve durumlarla ilişkilendirerek anlatın. Bu, konseptleri daha anlaşılır kılar.

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

B) Program Arayüzünde Değişkenler Oluşturulur, Değerlerle İlişkilendirilir Ve Bu Değerler Güncellenir.

Değişkenlerin Oluşturulması

Değişkenlerin, verileri saklamak için kullanılan isimlendirilmiş yerler olduğunu açıklanabilir. C++'da bir değişken tanımlarken türünün (int, double, string vb.) belirtilmesi gerektiğini vurgulanır.

Örnek:

cpp Copy code

```cpp
int yas = 20;
double boy = 1.75;
string ad = "Ahmet";
```

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

Değerlerle İlişkilendirme

Değişkenlere başlangıç değeri atamanın, değişkenin oluşturulduğu anda yapılabilen bir işlem olduğunu belirtin. Ayrıca, değişkenlere daha sonra da değer atamanın mümkün olduğunu açıklayın.

Örnek:

cpp Copy code

```cpp
yas = 21;
```

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

Değerlerin Güncellenmesi

Değişkenlerin değerlerinin programın çalışması sırasında güncellenebileceğini, bu işlemin değişkenin sakladığı bilgiyi değiştirdiğini anlatın.

Örnek:

cpp Copy code

```cpp
yas = yas + 1;
```

Günlük hayattan bir örnek verilecek oluşa; Bir kişinin banka hesabını düşünün. Hesapta saklanan para miktarı (balance), bir değişkendir. Para yatırma işlemi, bu değişkene değer ekler (balance = balance + yatirilanPara;); para çekme işlemi ise değişkenden değer çıkarır (balance = balance - cekilenPara;). Bu, değişkenlerin nasıl güncellendiğini anlamak için somut bir örnektir.

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

C) Matematiksel operatörler (+, -, *, / vb.) ile mantıksal operatörler (==, !=, <, > vb.) kullanılarak temel aritmetik ve karşılaştırma işlemleri gerçekleştirilir.

MATEMATİKSEL OPERATÖRLER

- Matematiksel operatörlerin (+, -, *, /) temel aritmetik işlemleri gerçekleştirmek için kullanıldığını açıklayın. Ayrıca, bölme işlemi sırasında tam sayı bölmesi ve ondalık sayı bölmesi arasındaki farklara değinin.

Örnek: Alışveriş yaparken toplam tutarı hesaplama (toplama), indirim uygulandığında fiyattan düşme (çıkarma), bir ürünün birden fazla alındığında toplam fiyatı (çarpma), bir hesaptan kişi başı payı bölme (bölme).

MANTIKSAL OPERATÖRLER

- Mantıksal operatörlerin (==, !=, <, >, <=, >=) iki değeri veya ifadeyi karşılaştırmak için kullanıldığını açıklayın. Bu operatörlerin genellikle koşullu ifadeler (if-else yapıları) içinde kullanıldığını vurgulayın.

Örnek: Bir filmi izlemek için yaşınızın yeterli olup olmadığını kontrol etme (>, <), bir ürünün stokta olup olmadığını kontrol etme (== veya !=).

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

Ç) Bir değişkenin değeri diğer bir değişkene atanarak veya aralarında matematiksel işlemler yapılarak veri aktarımı ve dönüşümü gerçekleştirilir.

Değer Ataması ve Matematiksel İşlemler

- Bir değişkenin değerinin, başka bir değişkene nasıl atandığını gösterin. Bu, temel bir atama işlemidir (= operatörü kullanılır).

Örnek: Bir kumbaradaki para miktarını düşünün. Eğer 10 TL'niz varsa ve bunu kumbaraya koyarsanız, artık kumbaradaki toplam para miktarı (değişken) bu yeni değere "atanmış" olur.

- İki değişken arasında temel matematiksel işlemler (+, -, *, /) yapılabilir ve sonuç başka bir değişkene atanabilir.

Örnek: Bir pastayı arkadaşlarınızla paylaşmak istiyorsunuz. Sizin elinizdeki (bir değişken) ve arkadaşınızın elindeki (diğer değişken) pasta miktarlarını toplayarak, toplamda kaç parça pasta olduğunu (sonucun atanacağı bir başka değişken) hesaplayabilirsiniz.

1. ÜNİTE: C++ ÇALIŞMA ORTAMI

Anlatım Teknikleri

Somutlaştırma: Matematiksel işlemleri ve değer atamasını, öğrencilerin günlük hayatta karşılaşabileceği somut durumlarla ilişkilendirerek açıklayın. Bu, kavramların daha kalıcı öğrenilmesini sağlar.

Görselleştirme: Değişkenler arasındaki veri akışını gösteren diyagramlar veya akış şemaları kullanın. Bu, öğrencilerin süreci görsel olarak anlamalarına yardımcı olur.

Pratik Uygulama: Öğrencilere, öğrendikleri kavramları kullanarak basit C++ programları yazma görevleri verin. Örneğin, iki sayıyı toplayıp sonucu başka bir değişkene atayan bir program yazmalarını isteyin.

2. ÜNİTE: KARAR VE DÖNGÜ YAPILARI

2.1. Karar İfadelerini Örneklerle Açıklar.

A) Programın belirli koşullara göre farklı davranışlar sergilemesini sağlayan karar ifadelerini farklı durumlar altında kontrol edip programlarında belirli şartlara göre yönlendirmesi anlatılır.

Karar İfadelerinin Anlatılması

İf ifadesinin, belirli bir koşulun doğru (true) olup olmadığını kontrol etmek için nasıl kullanıldığını açıklayın.

Else ve else if ifadelerinin, if koşulu yanlış (false) olduğunda alternatif yollar sunmak için nasıl kullanıldığını anlatın.

Switch ifadesinin, bir değişkenin birden fazla olası değerini kontrol etmek için nasıl kullanıldığını gösterin.

Karar Alma Örneği: Eğer hava yağmurluysa (koşul), o gün okula giderken şemsiye alır (if). Eğer hava güneşliyse, şemsiye almadan çıkar (else).

Birden Fazla Seçenek Örneği: Bir restoranda yemek seçimi yapmak gibi düşünebilirsiniz. Menüdeki yemekler (switch durumu), sipariş verirken numaralarına göre seçilir. Eğer 1 numarayı seçerseniz pizza, 2 numarayı seçerseniz hamburger alırsınız (case durumları). Seçim yapılmazsa, "Lütfen geçerli bir seçim yapın" (default durumu) uyarısı verilir.

2. ÜNİTE: KARAR VE DÖNGÜ YAPILARI

B) Karar ifadeleriyle ilgili karşılaştırmanın mantıksal operatörlerle (eşitlik, büyüklük-küçüklük kontrolü vb.) tanımlanması ve karşılaştırılması örneklendirilir.

Karar İfadeleri

İf ifadesi, belirli bir koşulun doğru (true) olup olmadığını kontrol eder. Eğer koşul doğruysa, if bloğu içindeki kodlar çalıştırılır.

Else ifadesi, if koşulu yanlış (false) olduğunda çalışacak kod bloğunu belirtir.

Else if ile birden fazla koşul sırayla kontrol edilebilir.

Switch ifadesi, bir değişkenin alabileceği çeşitli değerlere göre farklı kod bloklarını çalıştırmak için kullanılır.

Örnek: Eğer yağmur yağıyorsa (koşul), şemsiye alırız (if bloğu içindeki eylem); yağmur yağmıyorsa (koşul yanlışsa), şemsiye almaya gerek yoktur (else bloğu içindeki eylem).

2. ÜNİTE: KARAR VE DÖNGÜ YAPILARI

Mantıksal Operatörlerle Karşılaştırma

- Mantıksal operatörler, koşulların tanımlanmasında kullanılır. Eşitlik (==), eşitsizlik (!=), büyüklük (>), küçüklük (<), büyük eşit (>=), küçük eşit (<=) gibi operatörler, değişkenlerin değerlerini karşılaştırmak için kullanılır.
- Bu operatörler, if ve else if ifadelerindeki koşulları tanımlamak için kullanılır. Karar ifadeleriyle ilgili karşılaştırmanın mantıksal operatörlerle (eşitlik, büyüklük-küçüklük kontrolü vb.) tanımlanması ve karşılaştırılması örneklendirilir.

Örnek: Bir lunaparkta, bir oyuna binmek için yaş sınırı 12'dir. Eğer çocuğun yaşı 12 veya daha büyükse (yas >= 12), oyuna binebilir. Aksi takdirde, oyuna binemez.

2. ÜNİTE: KARAR VE DÖNGÜ YAPILARI

C) Programlamanın temel yapı taşı olan karar ifadelerini, kullanıcıdan alınan girdilere veya belirli şartlara bağlı olarak kullanacağı örnek uygulamalar yaptırılır.

Kullanıcı Girdisi: C++ programlama dilinde, kullanıcıdan bilgi almak için genellikle **cin** kullanılır ve karar ifadeleri (**if, else if, else**) bu girdilere bağlı olarak çeşitli akışlar oluşturmak için kullanılır.

Örnek: Bir eğlence parkı uygulaması düşünün, kullanıcının yaşı ve boyuna göre hangi oyunlara binebileceğini söylüyoruz. Eğer kullanıcı gençse ve boyu kısa ise, sadece çocuk oyunlarına binebilir. Eğer genç fakat boyu uzunsa, tüm oyunlara binebilir. Eğer kullanıcı yetişkinse, yaşından dolayı korku evine giremez.

2. ÜNİTE: KARAR VE DÖNGÜ YAPILARI

2.2. Mantıksal operatörleri kullanarak koşullar oluşturur.

A) Mantıksal operatörler (AND, OR, NOT) anlatılır.

AND Operatörü (&&)

AND operatörü, iki koşulun da doğru olması durumunda true (doğru) bir sonuç verir.

OR Operatörü (||)

OR operatörü, iki koşuldan en az birinin doğru olması durumunda true (doğru) bir sonuç verir.

NOT Operatörü (!)

NOT operatörü, bir koşulun tersini almak için kullanılır; eğer koşul false (yanlış) ise, true (doğru) sonucu verir.

2. ÜNİTE: KARAR VE DÖNGÜ YAPILARI

B) İki veya daha fazla koşulun aynı anda doğru olması gerektiği durumlarda mantıksal operatörleri (AND, OR, NOT) birleştirerek kullanması sağlanır.

AND ve OR Operatörlerinin Birleştirilmesi:

Bir sinema salonuna giriş için, bir biletinizin olması VEYA sinema salonu çalışanı olmanız gerekirken, aynı zamanda salonun açık olması (VE) gerektiğini düşünün.

NOT Operatörü ile Birleştirme:

Bir görevin tamamlanması için, görevin bitirilmemiş olması (NOT) VE verilen sürenin dolmamış olması gerektiğini düşünün.

2. ÜNİTE: KARAR VE DÖNGÜ YAPILARI

B) Mantıksal operatörler karmaşık koşulların analizi için farklı örneklerde çalışılır.

Senaryo: Oyun Karakteri Yükseltme Sistemi

Bir oyun karakterinin yükseltme yapabilmesi için belirli koşulların sağlanması gerektiğini varsayalım. Karakter, yeterli deneyim puanına (XP), yeterli altına ve özel bir nesneye sahip olmalıdır. Ayrıca karakterin sağlık durumunun tam olması ve belirli bir seviyenin üzerinde olması gerekmektedir. Karakterin bu yükseltmeyi yapabilmesi için tüm koşulların sağlanması gerekmektedir.

```cpp
#include <iostream>
using namespace std;

int main() {
    int deneyimPuanı;
    int altinMiktari;
    bool ozelNesneVarMi;
    int karakterSeviyesi;
    int karakterCanı;

    // Kullanıcıdan girdiler alınıyor
    cout << "Karakterinizin deneyim puanını giriniz: ";
    cin >> deneyimPuanı;
    cout << "Karakterinizin altın miktarını giriniz: ";
    cin >> altinMiktari;
    cout << "Özel nesneye sahip misiniz? (1 evet, 0 hayır): ";
    cin >> ozelNesneVarMi;
    cout << "Karakterinizin seviyesini giriniz: ";
    cin >> karakterSeviyesi;
    cout << "Karakterinizin canını giriniz (Maksimum 100): ";
    cin >> karakterCanı;

    // Koşulların kontrolü
    bool yukseltmeYapabilirMi = (deneyimPuanı > 5000) && (altinMiktari > 200) &&
                                ozelNesneVarMi && (karakterSeviyesi > 10) &&
                                (karakterCanı == 100);

    if (yukseltmeYapabilirMi) {
        cout << "Tebrikler! Karakterinizi yükseltebilirsiniz." << endl;
    } else {
        cout << "Yükseltme için gerekli koşullar sağlanmamıştır." << endl;
    }

    return 0;
}
```

2. ÜNİTE: KARAR VE DÖNGÜ YAPILARI

2.3. Döngü yapılarını kullanarak tekrarlayan işlemler gerçekleştirir.

A) Döngü yapıları ile belirli işlemlerin tekrarlanması öğretilir.

Döngünün Tanımı: Bir döngü, belirli bir koşul sağlandığı sürece bir dizi komutun veya işlemin sürekli olarak tekrar edilmesidir. Programlamada, bu genellikle bir dizideki öğeler üzerinden geçmek, bir sayıyı belirli bir değere ulaşana kadar artırmak veya azaltmak veya bir koşul doğru olduğu sürece bir işlemi yapmak için kullanılır.

Alarm Saati: Her sabah alarmınızın çalması gibi düşünebilirsiniz. Alarm çaldığında, bir dizi eylemi (yataktan kalkmak, dişlerinizi fırçalamak vb.) tekrar edersiniz. Bu eylemler, güne başlamanız için gerekli rutindir ve her sabah tekrar eder.

Sosyal Medya Akışı: Sosyal medyada arkadaşlarınızın yeni gönderilerini görmek için sayfayı sürekli yenilemek, bir döngünün günlük hayattaki bir örneğidir. Yeni içerik olup olmadığını kontrol etmek için bu işlemi belirli aralıklarla tekrarlarsınız.

2. ÜNİTE: KARAR VE DÖNGÜ YAPILARI

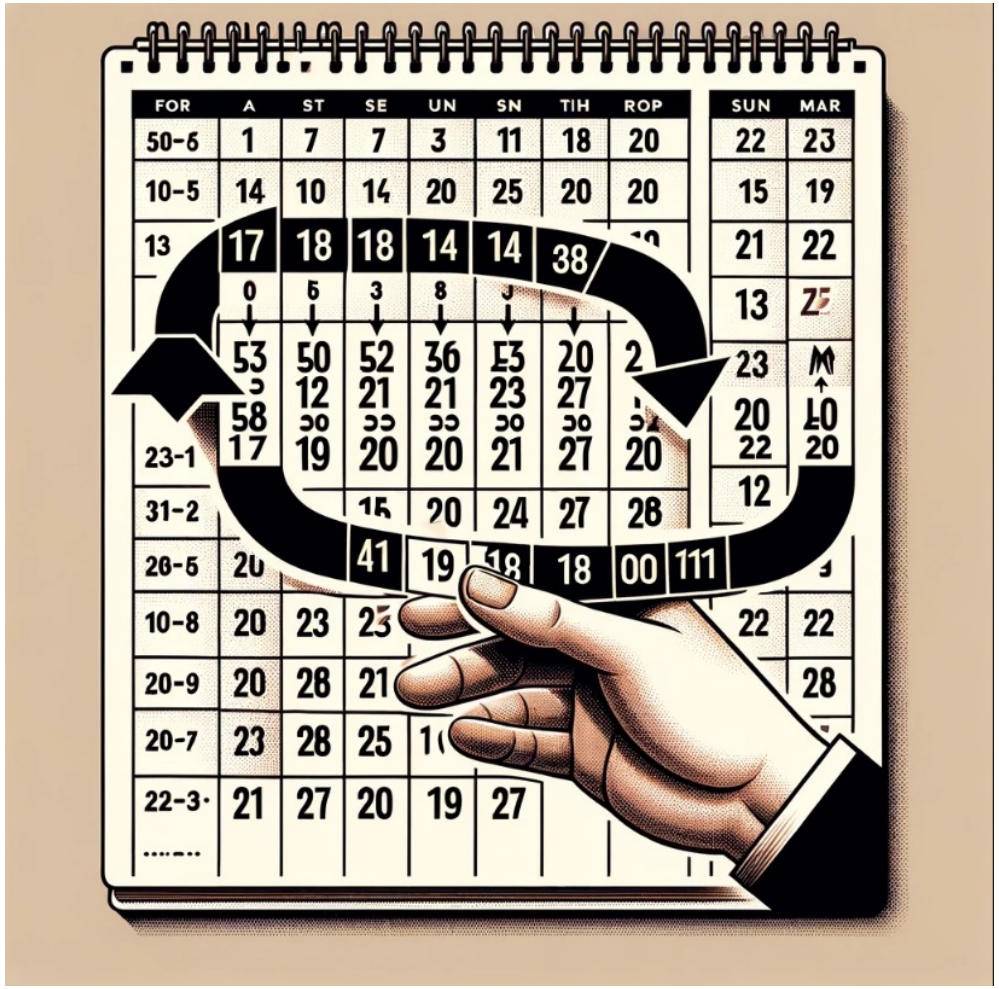
- Bu görsel ile belirli koşullar altında aynı eylemlerin birden çok kez tekrarlanması anlatılabilir.

2. ÜNİTE: KARAR VE DÖNGÜ YAPILARI

B) Farklı türdeki döngü yapılarının (for, while, do-while) çalışma prensibi açıklanır.

For Döngüsü: programlamada kullanılan bir kontrol yapısıdır ve belirli bir kod bloğunu belirli bir sayıda veya bir koleksiyon üzerindeki her öğe için tekrar tekrar çalıştırmak amacıyla kullanılır. for döngüsü, genellikle sayılabilir veya yinelemeye açık nesneler üzerinde işlem yaparken kullanılır.

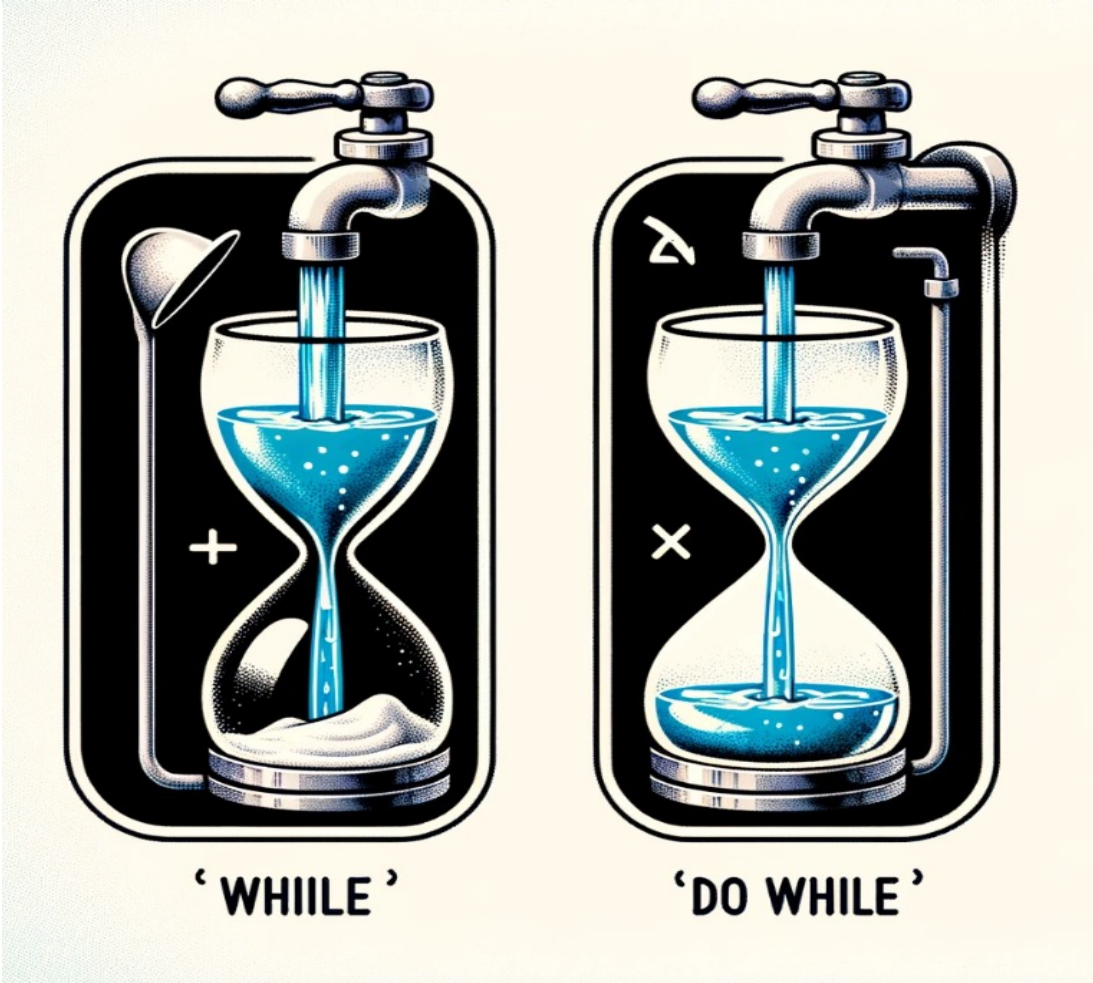
Takvimdeki günler sırayla numaralandırılmış ve her günü sırasıyla gösteren bir ok veya el, 'for' döngüsünün ilerleyişini simgelemektedir. Bu, bir dizi üzerinde iterasyon yapma kavramını açıkça göstermektedir.



2. ÜNİTE: KARAR VE DÖNGÜ YAPILARI

While Döngüsü: Koşul doğru (true) olduğu sürece yürütülür. Döngünün her bir yinelemesi başlamadan önce koşul kontrol edilir.

Do While Döngüsü: Koşul ne olursa olsun en az bir kez çalıştırılır, çünkü koşul yinelemenin sonunda kontrol edilir.



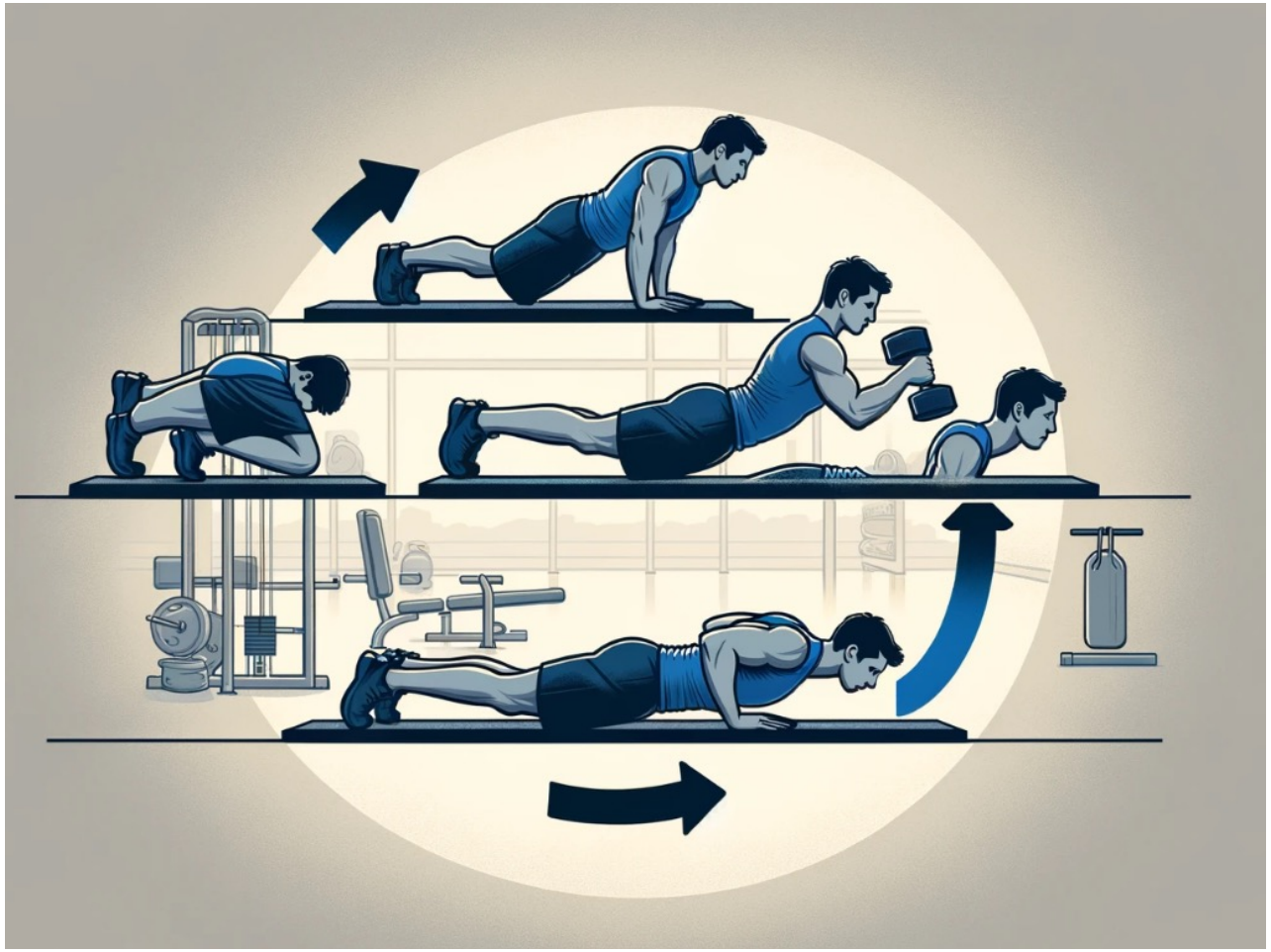
2. ÜNİTE: KARAR VE DÖNGÜ YAPILARI

C) Veri koleksiyonları üzerinde dolaşma, belirli bir aşamaya kadar işlem yapma, işlemi devam ettirme ve tekrar eden görevleri otomatikleştirme işlemlerinde döngüleri kullanma durumları örneklendirilir.

- **Alışveriş Listesi Örneği:** Alışveriş listesindeki her bir ürünü kontrol etmek ve markette bu ürünleri tek tek bulup sepete koymak. Bir for veya while döngüsü, listemizdeki her bir madde için tekrarlanabilir ve her maddenin sepete eklenip eklenmediğini kontrol edilir.

2. ÜNİTE: KARAR VE DÖNGÜ YAPILARI

- **Spor Salonu Antrenmanı Örneği:** Spor salonunda belirli sayıda set ve tekrar yapma. Örneğin, 3 set 10 tekrar şınav çekmek. Bir for döngüsü, her set için 10 şınav çekmemizi ve sonra bir sonraki sete geçmemizi sağlar.



3. ÜNİTE: FONKSİYONLAR VE DİZİLER

3.1. C++ programlama dilinde kullanılan fonksiyonları tanır.

A) C++ programlama dilinde fonksiyonların nasıl tanımlandığı ve çağrıldığı anlatılır.

C++ programlama dilinde fonksiyonların tanımlanması ve çağrılması, belirli bir görevi yerine getirmek için kod parçacıklarını organize etme yöntemidir. Fonksiyonlar, belirli bir işi yapmak için gereken kodu gruplandırır ve programın başka yerlerinden bu fonksiyonları çağırarak kod tekrarını azaltır ve programı daha okunabilir hale getirir.

Fonksiyon Tanımlama

Fonksiyonlar, bir işi gerçekleştiren ve isteğe bağlı olarak bazı verileri alan ve bazı sonuçlar döndüren kod bloklarıdır.

c++ Copy code

```c++
// Bir sayının karesini hesaplayan fonksiyon
int kare(int x) {
    return x * x;
}
```

3. ÜNİTE: FONKSİYONLAR VE DİZİLER

Fonksiyon Çağrılması

Bir fonksiyon tanımlandıktan sonra, onu kullanmak için çağırmak gerekir. Bu, fonksiyonun adını ve parantez içinde gerekli argümanları yazarak yapılır**.**

```c++
int sonuc = kare(5); // 'kare' fonksiyonunu çağırır ve sonucu 'sonuc' değişkenine ata
```

3. ÜNİTE: FONKSİYONLAR VE DİZİLER

B) Fonksiyonların parametreleri iletme ve değer döndürme işlemleri gösterilir.

Bu kavramlar günlük hayattan bir örnekle öğrencilere açıklanabilir:

Bir restoranda sipariş vermek, fonksiyonların parametre almasına ve değer döndürmesine benzetilebilir. Müşteri olarak siz (program), garsona (fonksiyon) ne yemek istediğinizi (parametreler) söylersiniz ve garson size istediğiniz yemeği (değer döndürme) getirir.

Sipariş Verme: Bir müşterinin menüyü incelediği ve ardından garsona yemek siparişi verdiği bir sahne. Menüdeki seçenekler fonksiyon parametrelerini temsil eder ve müşterinin yaptığı seçim fonksiyona geçirilen argümanları temsil eder.

Yemek Servisi: Garsonun müşteriye yemeği getirdiği sahne. Bu, fonksiyonun işlemi tamamladıktan sonra bir değer döndürmesine benzetilir.

3. ÜNİTE: FONKSİYONLAR VE DİZİLER

Bu sahneleri C++ koduyla ilişkilendirmek için, fonksiyon parametrelerinin fonksiyonun alabileceği argümanlar olduğunu ve fonksiyonun geri döndürdüğü değerin ise işlem sonucunda elde edilen sonuç olduğunu anlatabilir.

c++ Copy code

```c++
// Fonksiyon tanımı
int hesaplaToplam(int fiyat, int adet) {
    return fiyat * adet;
}

// Fonksiyon çağrısı
int toplamTutar = hesaplaToplam(10, 3); // 10 birim fiyatında 3 adet ürün siparişi
```

3. ÜNİTE: FONKSİYONLAR VE DİZİLER

3.2. Nesne tabanlı programlamayı anlar ve C++ programlama arayüzü içerisinde nasıl uygulandığını öğrenir.

A) Nesne tabanlı programlamanın temel kavramları (sınıflar, nesneler, kalıtım, çok biçimlilik) açıklanır ve nesne tabanlı programlamanın avantajları anlatılır.

C++ programlamada nesne tabanlı programlamanın (NTP) temel kavramlarını öğrencilere anlatmak için günlük hayattan alınan örnekler ve metaforlar kullanmak etkilidir.

Sınıflar ve Nesneler

Bir sınıf, bir yemek tarifi gibidir; yani bir kek yapmak için gerekli malzemeleri ve yapılış aşamalarını içeren bir reçete. Bir nesne ise bu tariften yapılan gerçek kektir.

Kalıtım

Bir ailenin çocuklarına özelliklerini (saç rengi, göz rengi vs.) aktarması gibi düşünülebilir. Ana sınıf (ebeveyn), alt sınıflara (çocuklar) bazı özelliklerini ve davranışlarını aktarır.

Çok Biçimlilik

Farklı tip telefonların aynı şarj cihazı ile şarj olması gibi düşünülebilir. Her telefon farklı özelliklere sahip olsa da, hepsi aynı arayüzü (şarj portunu) kullanarak enerjilerini doldurabilirler.

3. ÜNİTE: FONKSİYONLAR VE DİZİLER

Nesne Tabanlı Programlamanın Avantajları

Kodun Yeniden Kullanılabilirliği: Bir kez yazılan sınıf başka programlarda da kullanılabilir.

Düzen ve Okunabilirlik: Program, modüllere ayrılarak daha anlaşılır hale gelir.

Bakımın Kolaylığı: Sınıf kodu değiştiğinde, bu sınıfı kullanan tüm nesneler otomatik olarak güncellenir.

Güvenlik: Sınıflar, verileri ve fonksiyonları kapsülleyerek dış etkilere karşı korur.

3. ÜNİTE: FONKSİYONLAR VE DİZİLER

Örneğin, bir "Araba" sınıfı oluşturalım. Bu sınıf, arabaların genel özelliklerini tanımlar. Daha sonra "SporAraba" sınıfı ile kalıtım yoluyla bu genel özellikleri miras alır ve üzerine ek özellikler ekleyerek çok biçimlilik özelliğini gösterir.

c++ Copy code

```c++
// Sınıf Tanımı
class Araba {
public:
    string marka;
    string model;
    int yil;

    void calistir() {
        cout << "Araba çalışıyor." << endl;
    }
};

// Kalıtım Yoluyla Alt Sınıf Tanımı
class SporAraba : public Araba {
public:
    void turboEkle() {
        cout << "Turbo güç eklendi." << endl;
    }

    // Çok Biçimlilik Örneği
    void calistir() {
        cout << "Spor Araba hızlı bir şekilde çalışıyor." << endl;
    }
};
```

3. ÜNİTE: FONKSİYONLAR VE DİZİLER

Bir etkinlik yönetimi sistemi düşünün. Bu sistem, farklı türdeki etkinlikleri kapsayabilir (konferanslar, konserler, atölye çalışmaları vb.). Yönetebilir, katılımcı kaydı yapabilir ve etkinlik sırasında çeşitli görevleri otomatize edebilir.

Sınıflar:

Etkinlik: etkinliğin adı, türü, tarihi, yeri gibi özellikleri ve etkinlikle ilgili fonksiyonları içerir.

Katılımcı: adı, soyadı, e-posta adresi gibi özellikleri ve etkinliğe kaydolma fonksiyonlarını içerir.

Organizatör: etkinlikleri düzenleme, katılımcı listelerini yönetme yetenekleri gibi fonksiyonları içerir.

Kalıtım ve çok biçimlilik:

Konferans, **konser** ve **atölye** sınıfları, **etkinlik** sınıfından türetilir ve her biri farklı özellikler ve fonksiyonlar ekleyerek çok biçimliliği kullanır.

c++ Copy code

```c++
#include <iostream>
#include <list>
using namespace std;

// Etkinlik sınıfı
class Etkinlik {
protected:
    string ad;
    string tur;
    string tarih;
    string yer;
public:
    Etkinlik(string ad, string tur, string tarih, string yer) : ad(ad), tur(tur), tar
    virtual void etkinlikDetaylariniGoster() {
        cout << ad << " etkinliği " << tarih << " tarihinde " << yer << " adresinde g
    }
};

// Konser sınıfı
```

3. ÜNİTE: FONKSİYONLAR VE DİZİLER

Etkinlik', 'Katılımcı' ve 'Organizatör' sınıflarını tanımlayan kod parçaları, bu sınıfların nitelikleri ve yöntemleri de kod içinde gösteriliyor. 'Konferans', 'Konser' ve 'Atölye' sınıfları 'Etkinlik' sınıfından türetilerek kalıtım gösteriliyor ve bu alt sınıfların eşsiz işlevleri, çok biçimliliği temsil ediyor.

Bu kod parçası, bir etkinlik yönetim sisteminin basit bir versiyonunu modeller ve etkinlikleri, katılımcıları ve organizatörleri temsil eder. Her sınıfın kendine özgü işlevleri vardır ve Concert sınıfı, Event sınıfından türetilerek çok biçimliliği sergiler.

```cpp
// Event base class
class Event {
public:
    string name;
    string location;
    string time;
    list<string> attendees;

    Event(string n, string loc, string t) : name(n), location(loc), time(t) {}

    virtual void advertise() {
        cout << "Join us at " << name << " located at " << location << " on " << tim
    }
};

// Participant class
class Participant {
public:
    string firstName;
    string lastName;

    Participant(string fName, string lName) : firstName(fName), lastName(lName) {}

    void registerForEvent(Event& e) {
        e.attendees.push_back(firstName + " " + lastName);
        cout << firstName << " " << lastName << " has registered for " << e.name << e
    }
};

// Organizer class
class Organizer {
public:
    string orgName;

    Organizer(string name) : orgName(name) {}

    void scheduleEvent(Event& e) {
        cout << "Event " << e.name << " has been scheduled by " << orgName << endl;
    }
};

// Concert class derived from Event
class Concert : public Event {
public:
    string bandName;

    Concert(string n, string loc, string t, string band) : Event(n, loc, t), bandNam

    void advertise() override {
        cout << "Come and see " << bandName << " perform at " << name << "!" << endl;
    }
};
```

3. ÜNİTE: FONKSİYONLAR VE DİZİLER

3.3. Değer ve referans tipleri arasındaki farkı kavrar.

A) Değer tipleri (int, float, bool) ve referans tipleri (sınıflar, diziler) arasındaki temel farklar açıklanır.

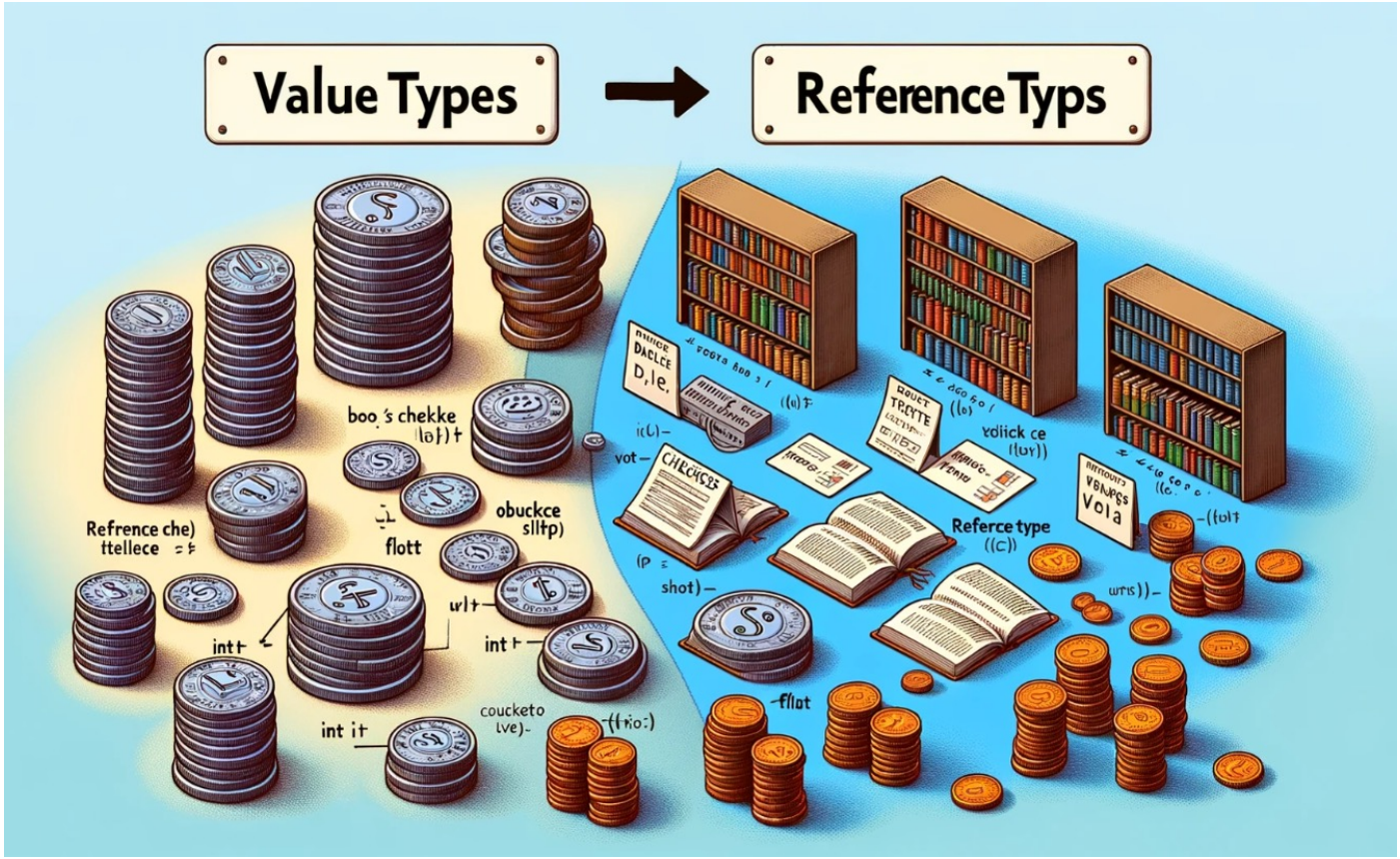
Değer tipleri: Değişkenin değerini doğrudan saklar. Bu değerler, belleğin yığın (stack) bölümünde saklanır. int, float, char, bool ve C++'ın tüm temel veri tipleri değer tipidir.

Referans tipleri: Bir nesnenin veya bir dizinin bellekteki adresini tutar. Bu adresler ve referansla gösterilen nesneler, belleğin yığın dışındaki alanlarında (örneğin, heap) saklanır. Sınıflar, diziler, pointerlar ve referanslar C++'da referans tipidir.

3. ÜNİTE: FONKSİYONLAR VE DİZİLER

Sol Taraf (Değer Tipleri): Değer tipleri senaryosu, int veya float gibi ayrı ve bağımsız değerleri temsil eden madeni paralarla tasvir edilmiştir. Her bir para birimi benzersizdir ve diğerleriyle bağlantılı değildir, bu da değer tiplerinin veriyi doğrudan nasıl tuttuğunu gösterir.

Sağ Taraf (Referans Tipleri): Referans tipleri senaryosu, kütüphaneden ödünç alınan kitaplarla görselleştirilmiştir. Kitaplar, nesneleri veya dizileri temsil eder ve her bir kitap ödünç alma fişi, kütüphanedeki kitaba referans verir, bu da referans tiplerinin bir nesnenin bellekteki konumuna nasıl atıfta bulunduğuna benzer.

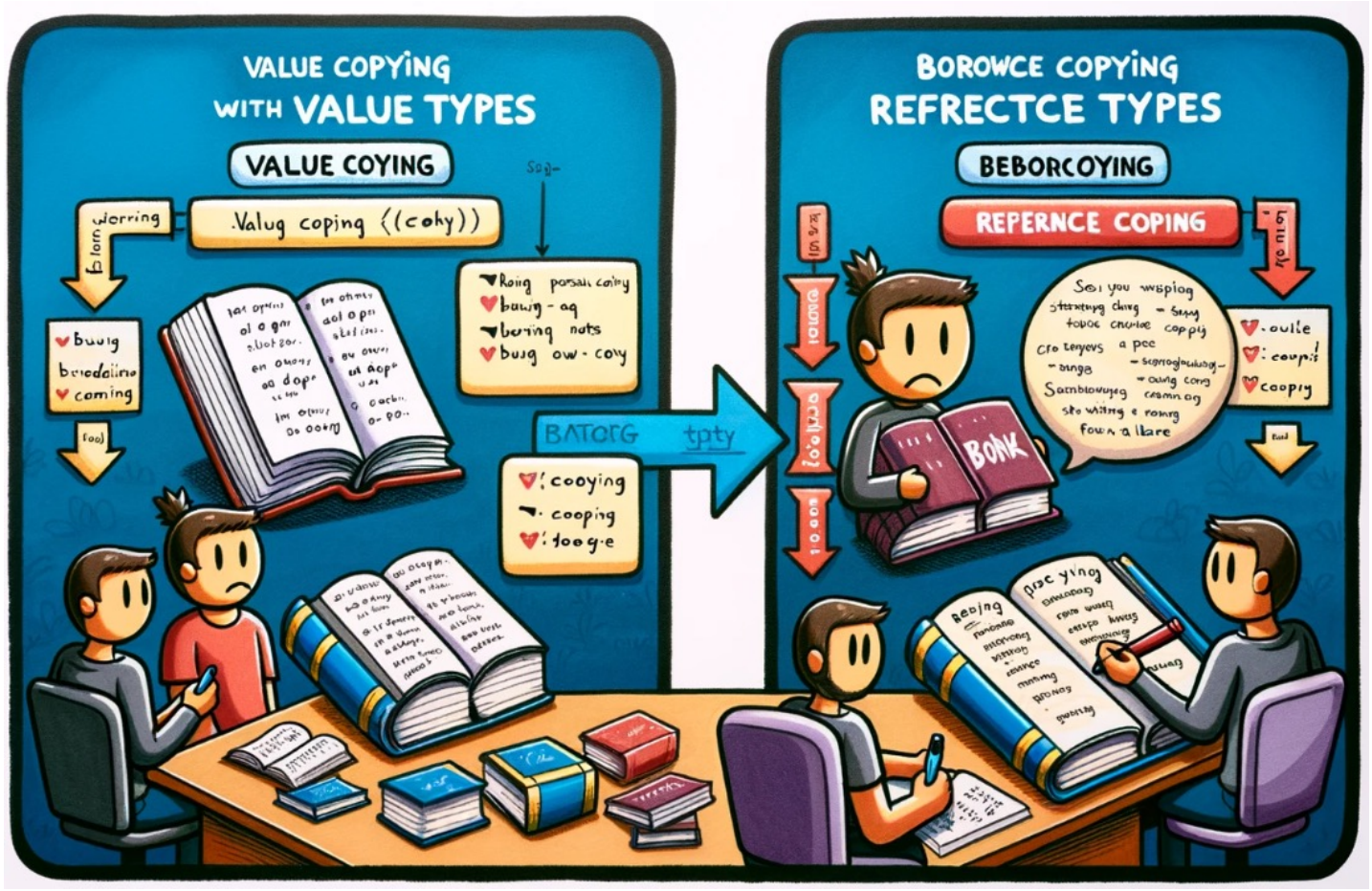


3. ÜNİTE: FONKSİYONLAR VE DİZİLER

B) Değer tiplerinin değer kopyalama, referans tiplerinin referans kopyalama üzerinde nasıl çalıştığı açıklanır.

Değer Kopyalama (Değer Tipleri için): Bu, bir arkadaşınıza bir kitap önerdiğinizde ve o arkadaşınızın gidip kendi kopyasını satın alması gibidir. Her iki kişinin de aynı kitabın farklı fiziksel kopyaları vardır; bir kişi kitabının üzerine not alsa bile, bu durum diğerinin kitabını etkilemez.

Referans Kopyalama (Referans Tipleri için): Bu, bir kitaplıkta bulunan bir kitabı ödünç almanız gibidir. Kitabın sadece bir kopyası vardır ve siz kitabın referansını (yani, kitabı ödünç almayı) arkadaşınıza verirsiniz. Eğer kitap üzerine not alırsanız, arkadaşınız da aynı kitabı ödünç aldığında bu notları görecektir çünkü her ikisi de kitabın aynı fiziksel kopyasını paylaşmaktadır



3. ÜNİTE: FONKSİYONLAR VE DİZİLER

Değer Tipleri

Basit, değişmez veri tipleriyle çalışırken (örneğin, sayılar, doğruluk değerleri) veya küçük yapılar ve sınıflarla işlem yaparken.

Avantajları: Değer tipleri ile çalışmak, verilerin bağımsız kopyalarını oluşturur, bu da yan etkileri azaltır ve kodun tahmin edilebilirliğini artırır. Hızlı erişim ve veri güvenliği sağlar.

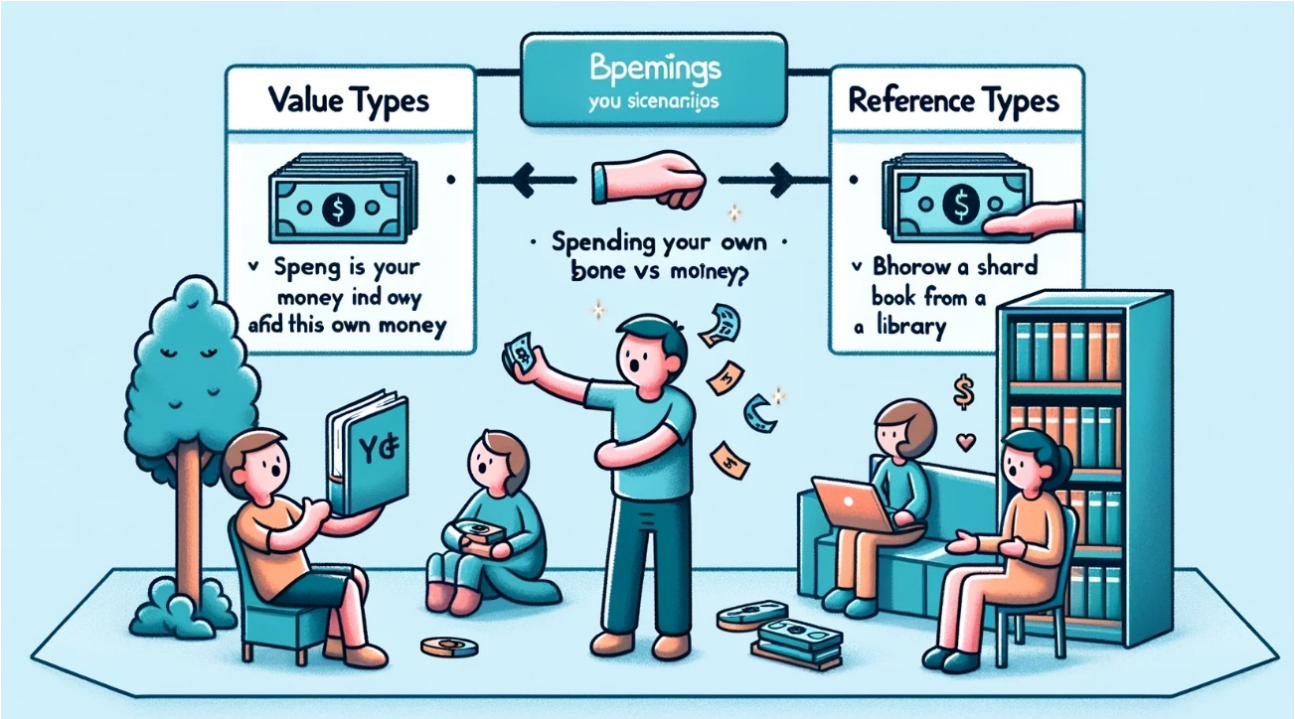
Para Kopyalama: Kendi paranızı harcamak gibi. Harcadığınız para, bütçenizden düşer ve başka bir yerde kullanılamaz. Bu, değer tiplerinin bağımsızlığını ve izolasyonunu temsil eder.

Referans Tipleri

Büyük veri yapıları veya nesnelerle çalışırken (örneğin, büyük diziler, sınıflar) veya veri üzerinde birden fazla fonksiyon veya metodun işlem yapması gerektiğinde.

Avantajları: Referans tipleri, bellek kullanımını optimize eder ve büyük veri yapılarını yönetmek için daha etkilidir. Değişiklikler, referansı paylaşan tüm nesneler arasında görülebilir, bu da veri tutarlılığını sağlar.

Kitap Ödünç Alma: Kütüphaneden bir kitap ödünç almak ve arkadaşınıza önermek. Arkadaşınız, aynı kitabı ödünç alır ve üzerindeki notları görebilir. Bu, referans tiplerinin paylaşılan kullanımını ve etkileşimini temsil eder.



3. ÜNİTE: FONKSİYONLAR VE DİZİLER

3.4. Tek boyutlu ve çok boyutlu dizileri kullanır.

A) Tek boyutlu ve çok boyutlu dizileri basit biçimde nasıl oluşturup kullanacağı anlatılır.

Tek Boyutlu Diziler

Bir sınıftaki öğrencilerin sınav puanlarını tutmak. Burada her öğrenciye ait tek bir sınav puanı vardır ve bu puanlar lineer bir liste şeklinde düzenlenebilir.

Bir liste içerisinde sırayla dizilmiş sınav puanları. **(Örneğin: 75, 82, 90, 55, 88)**

cpp Copy code

```cpp
#include <iostream>
using namespace std;

int main() {
    // 5 öğrencinin sınav puanlarını tutan tek boyutlu dizi
    int sinavPuanlari[5] = {75, 82, 90, 55, 88};

    // Dizideki puanları yazdır
    for(int i = 0; i < 5; i++) {
        cout << "Ogrenci " << (i+1) << ": Puan = " << sinavPuanlari[i] << endl;
    }

    return 0;
}
```

3. ÜNİTE: FONKSİYONLAR VE DİZİLER

Çok Boyutlu Diziler

Bir sınıftaki öğrencilerin farklı derslerden aldıkları sınav puanlarını tutmak. Burada her öğrenci için matematik, fen bilimleri, ve edebiyat derslerinden alınan puanlar gibi birden fazla sınav puanı vardır ve bu puanlar matris formunda düzenlenebilir.

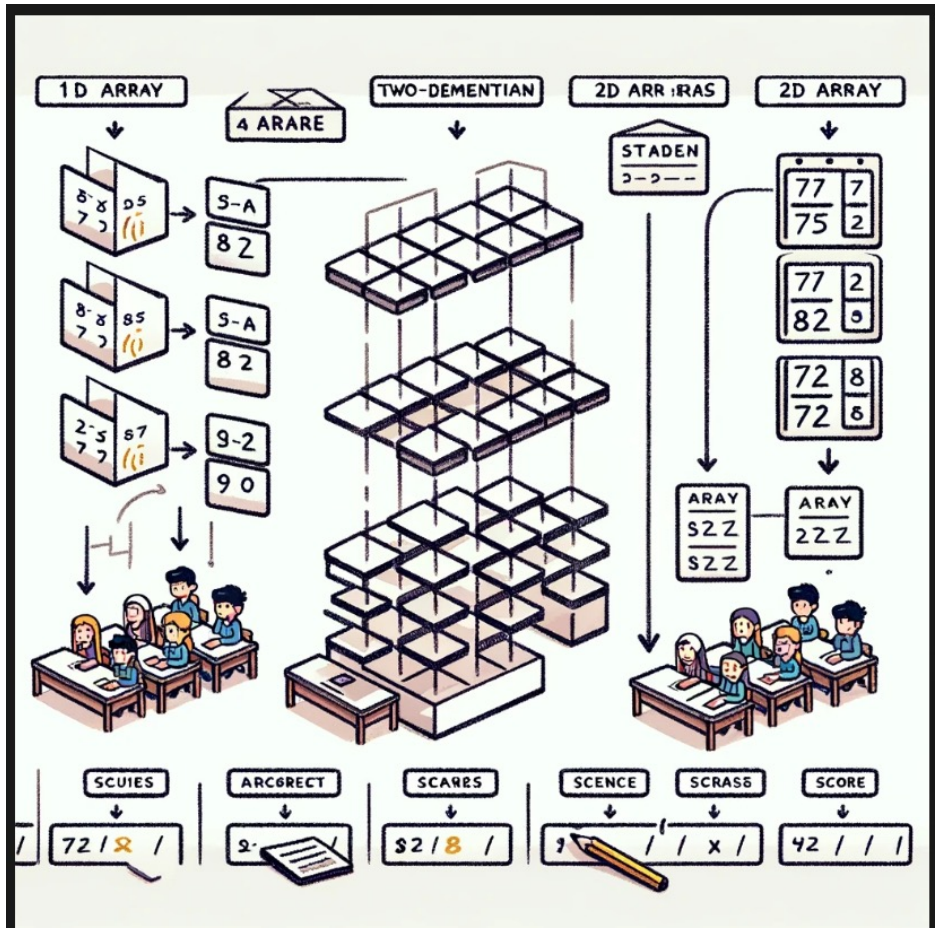
Bir tablo veya matris içerisinde, satırlar öğrencileri ve sütunlar farklı derslerden alınan sınav puanlarını temsil eder.

cpp Copy code

```cpp
#include <iostream>
using namespace std;

int main() {
    // 3 öğrencinin 3 farklı dersten aldığı puanları tutan çok boyutlu dizi
    int puanlar[3][3] = {
        {75, 85, 78}, // 1. öğrencinin puanları
        {88, 92, 91}, // 2. öğrencinin puanları
        {65, 74, 69}  // 3. öğrencinin puanları
    };

    // Dizideki puanları yazdır
    for(int i = 0; i < 3; i++) {
        cout << "Ogrenci " << (i+1) << ": ";
        for(int j = 0; j < 3; j++) {
            cout << puanlar[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

3. ÜNİTE: FONKSİYONLAR VE DİZİLER

Bu görsel, öğrenci sınav puanlarının kullanıldığı bir bağlamda, Tek boyutlu dizilerin basit bir puan serisi olarak nasıl temsil edildiğini, 2 boyutlu dizilerin ise farklı derslerden alınan puanların bir ızgara veya tablo formatında nasıl organize edildiğini gösterir.



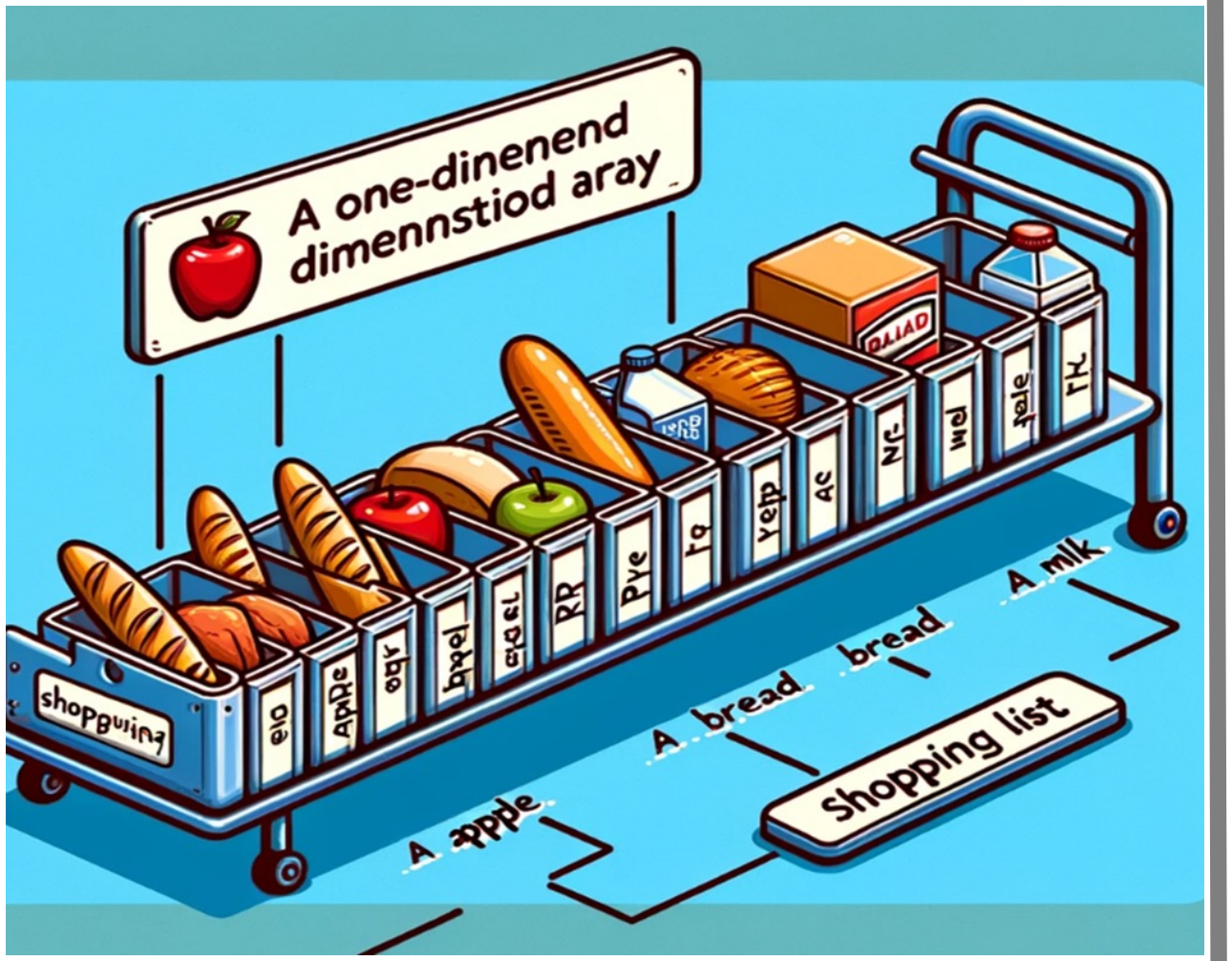
3. ÜNİTE: FONKSİYONLAR VE DİZİLER

B) Dizi elemanlarına nasıl erişileceği ve dizi işlemlerinin nasıl gerçekleştirileceği açıklanır.

Tek Boyutlu Dizi Örneği

Bir alışveriş listesi düşünün. Alışveriş listesi, farklı ürünlerin (örneğin, elma, ekmek, süt) bir listesidir ve bu ürünler tek bir satırda sıralanabilir.

```cpp
#include <iostream>
using namespace std;

int main() {
    string alisverisListesi[3] = {"Elma", "Ekmek", "Süt"};

    // Dizi elemanlarına erişim ve yazdırma
    for(int i = 0; i < 3; i++) {
        cout << alisverisListesi[i] << endl;
    }

    return 0;
}
```

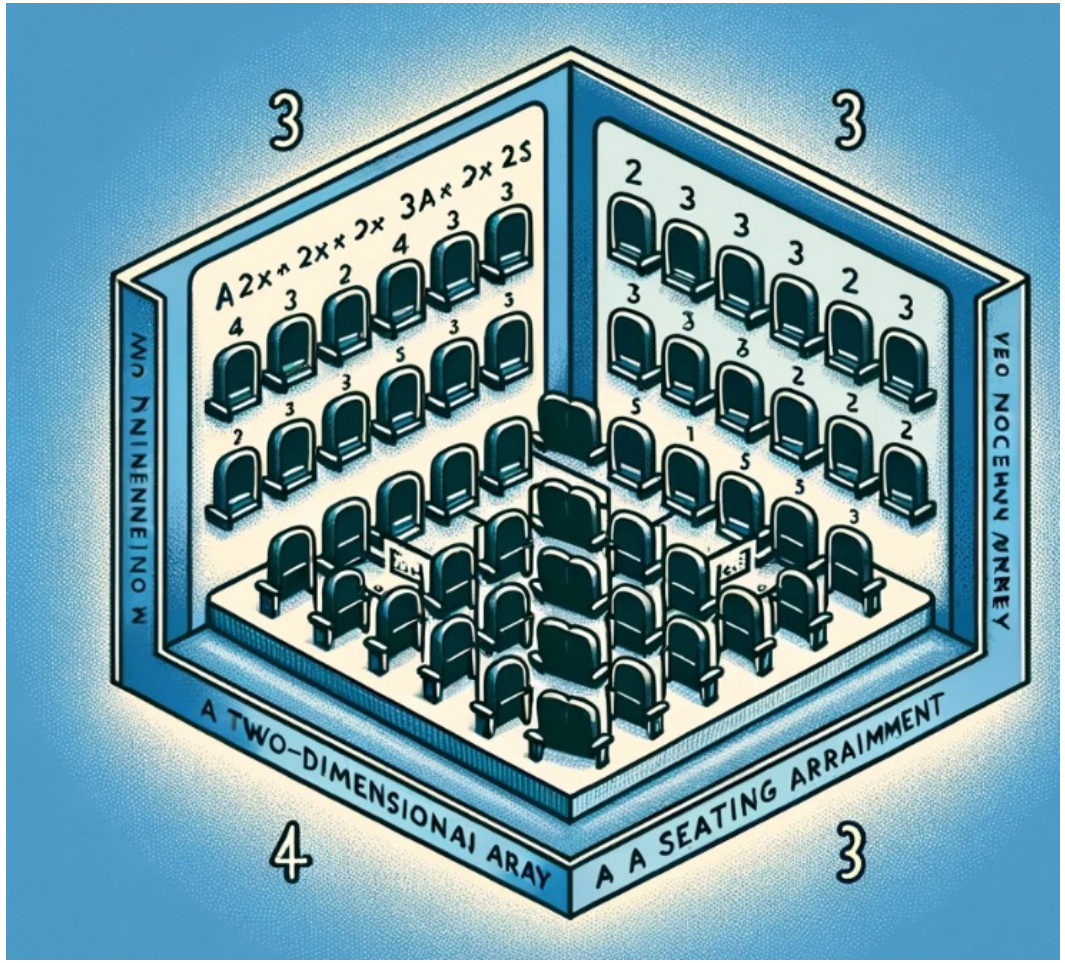


3. ÜNİTE: FONKSİYONLAR VE DİZİLER

Çok Boyutlu Dizi Örneği

Bir sinema salonundaki koltuk düzeni. Sinema salonu, birkaç sıra ve her sırada birkaç koltuk içerecek şekilde düzenlenmiştir.

```cpp
#include <iostream>
using namespace std;

int main() {
    // 3×3'lük bir sinema salonu koltuk düzeni
    int koltuklar[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

    // Çok boyutlu dizi elemanlarına erişim ve yazdırma
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            cout << "Koltuk " << koltuklar[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```



3. ÜNİTE: FONKSİYONLAR VE DİZİLER

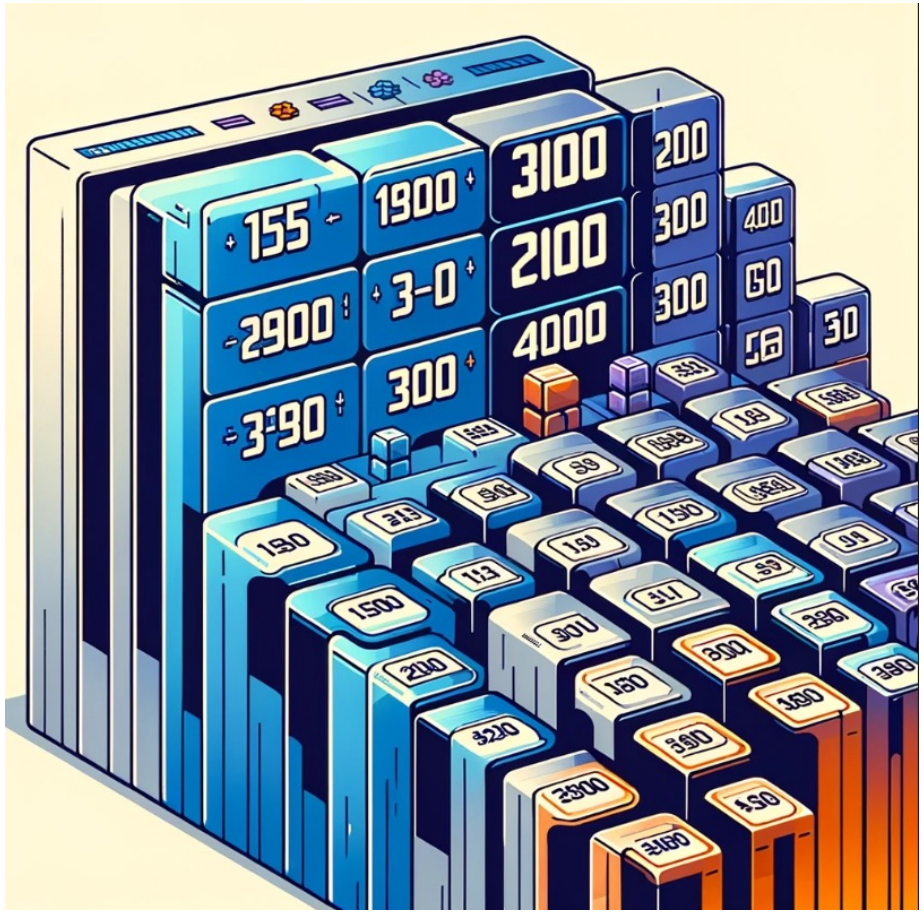
C) Dizilerin farklı veri tipleri ile nasıl kullanılacağı ve işlevselliği anlatılır.

Tek Boyutlu Dizi Örneği: Oyuncu Puanları

Senaryo: Bir oyunda, oyuncuların elde ettiği puanları tutan bir liste.

cpp Copy code

```cpp
#include <iostream>
using namespace std;

int main() {
    // 5 oyuncunun puanlarını tutan tek boyutlu dizi
    int oyuncuPuanlari[5] = {1500, 3200, 2900, 4100, 3000};

    // Puanları yazdır
    for(int i = 0; i < 5; i++) {
        cout << "Oyuncu " << i+1 << ": Puan = " << oyuncuPuanlari[i] << endl;
    }

    return 0;
}
```

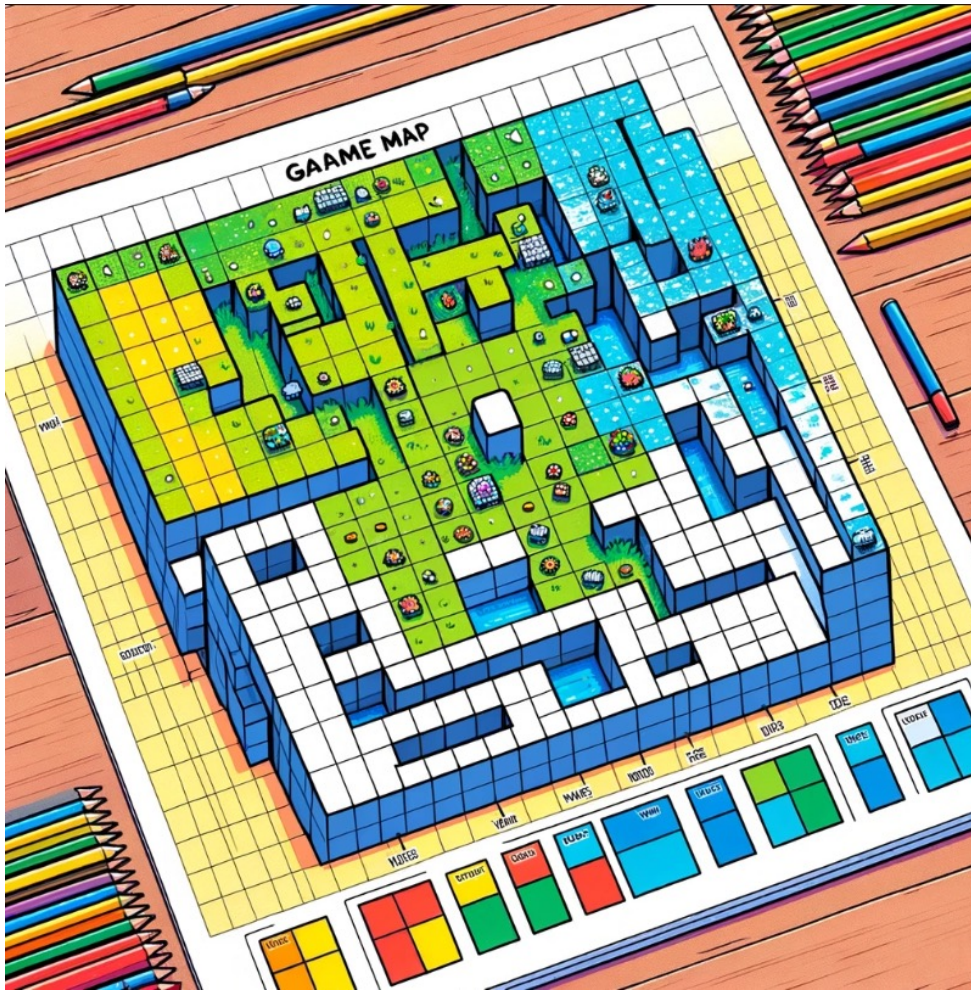


3. ÜNİTE: FONKSİYONLAR VE DİZİLER

Çok Boyutlu Dizi Örneği: Oyun Haritası

Senaryo: Bir oyunun 2D haritası, farklı türdeki alanları (örneğin, zemin, duvar, su) temsil eden sayılarla belirlenir.

```cpp
#include <iostream>
using namespace std;

int main() {
    // 5×5'lik bir oyun haritası
    int oyunHaritasi[5][5] = {
        {1, 1, 1, 1, 1},
        {1, 0, 0, 0, 1},
        {1, 0, 2, 0, 1},
        {1, 0, 0, 0, 1},
        {1, 1, 1, 1, 1}
    };

    // Harita üzerinde gezinme
    for(int i = 0; i < 5; i++) {
        for(int j = 0; j < 5; j++) {
            cout << oyunHaritasi[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```



4. ÜNİTE: FORMLAR

4.1. Formları, form sınıflarını, kontrol sınıflarını ve konteyner sınıflarını tanır ve bu yapıları kullanır.

A) İşletim sistemi uygulamalarında kullanılan temel arayüz bileşenleri ve form sınıflarının nasıl oluşturulup kullanılacağı anlatılır.

C++ programlamada "form", genellikle grafik kullanıcı arayüzü (GUI) uygulamalarında kullanılan bir terimdir. Bir form, kullanıcıya görsel olarak bilgi sunan ve kullanıcıdan girdi alabilen bir pencere veya diyalog kutusu gibi arayüz elemanlarını içerir. Bu terim, bazı programlama ortamları ve kütüphanelerinde, özellikle de Microsoft'un Visual Studio gibi IDE'lerinde ve Qt, wxWidgets, veya GTK gibi çapraz platform GUI kütüphanelerinde sıkça karşılaşılan bir kavramdır.

Formlar, metin kutuları, butonlar, etiketler, listeler ve diğer kullanıcı arayüzü widget'ları gibi bir dizi kontrol veya bileşen içerebilir. Bu kontroller, kullanıcı etkileşimini yönetmek, veri girişini kabul etmek ve uygulamanın işlevselliğini gerçekleştirmek için programlama mantığı ile birleştirilir.

C++'da form oluşturmak ve kullanmak için doğrudan dil desteği bulunmamaktadır; bu işlevsellik, üçüncü taraf kütüphaneler veya framework'ler aracılığıyla sağlanır. Bu kütüphaneler, form oluşturma, olay işleme (event handling) ve kullanıcı arayüzü elemanlarının davranışlarını kontrol etme gibi konuları kapsar. C++ ile GUI geliştirmek, bu kütüphanelerin sağladığı sınıflar ve metodlar kullanılarak yapılır.

4. ÜNİTE: FORMLAR

Form Oluşturma Süreci

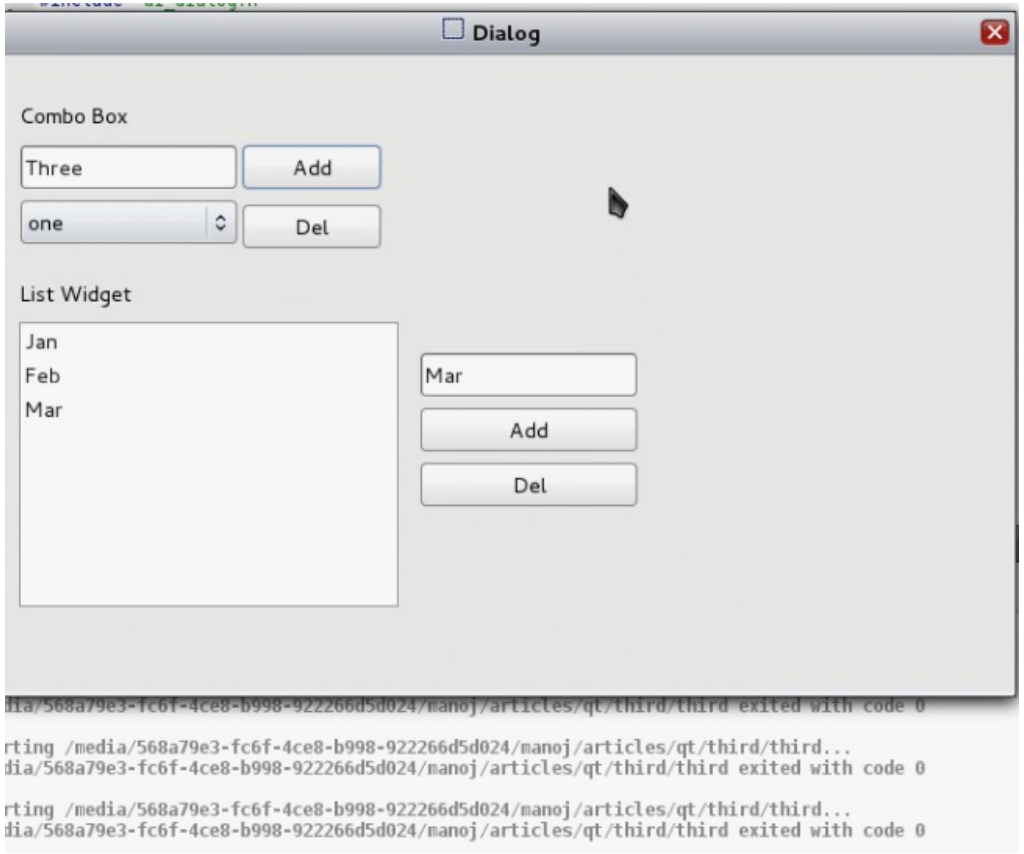
Kütüphane Seçimi: İlk adım, GUI uygulaması geliştirmek için kullanılacak kütüphaneyi seçmektir (Qt, wxWidgets, GTK, vb.).

Form Tasarımı: Seçilen kütüphanenin tasarım araçları veya kod kullanılarak form ve içindeki kontroller tasarlanır.

Olay Yönetimi: Buton tıklamaları, metin değişiklikleri gibi kullanıcı etkileşimleri için olay yöneticileri (event handlers) tanımlanır.

Veri Bağlama: Form elemanları ile uygulama verileri arasında bağlantı kurulur (örneğin, bir metin kutusuna girilen verinin bir değişkene atanması).

Test ve Hata Ayıklama: Formun ve içerdiği kontrollerin beklenildiği gibi çalıştığından emin olmak için uygulama test edilir ve hata ayıklama yapılır.



4. ÜNİTE: FORMLAR

B) Kontrol sınıfları (butonlar, metin kutuları, etiketler vb.) tanıtılır ve bu kontrollerin form içine nasıl yerleştirileceği gösterilir.

Bir kafeterya sipariş sistemi örneğini kullanabilirsiniz. Kafeteryada bir menüden yiyecek ve içecek seçmek, seçimlerinizi bir sipariş formuna yazmak ve siparişinizi göndermek için bir butona basmak, C++ ile GUI kontrollerini kullanarak bir form oluşturmanın gerçek d.ünya karşılığıdır.

Kontrol Sınıfları ve Kullanımları

Butonlar (QPushButton): Kafeterya örneğinde, siparişinizi göndermek için bir butona basarsınız. Bu, programda bir eylemi tetiklemek için kullanılır. Örneğin, bir "Sipariş Gönder" butonu, kullanıcının seçimlerini toplayıp işlemek üzere bir fonksiyonu çağırabilir.

Metin Kutuları (QLineEdit): Sipariş formunda adınızı veya masanızın numarasını yazmanız gereken bir alan düşünün. Bu, kullanıcıdan giriş almak için kullanılır. Programda, bir kullanıcının girdiği metni almak veya göstermek için metin kutuları kullanılır.

Etiketler (QLabel): Menüdeki her yemeğin yanında bulunan açıklamalar gibi, bilgi sunmak için kullanılır. Programda, metin veya görselleri göstermek için etiketler kullanılır.

4. ÜNİTE: FORMLAR

Kontrollerin form içine nasıl yerleştirileceğini anlatmak için, bu kafeterya sipariş formunu bir bilgisayar ekranına dönüştürdüğünüzü hayal edin. Formun üst kısmında sipariş verenin adını gireceği bir metin kutusu, altında seçimleri yapabileceği çeşitli butonlar ve her seçeneğin yanında seçeneği açıklayan etiketler bulunur. Siparişin altında, siparişi göndermek için bir "Gönder" butonu yer alır.

Qt Designer ile Form Tasarımı: Qt Designer kullanarak, bir form üzerine butonlar, metin kutuları ve etiketler yerleştirilir. Her bir widget, kullanıcının etkileşimine göre tasarlanır ve yerleştirilir.

Widget Özelliklerinin Ayarlanması: Her kontrolün özellikleri (örneğin, boyutu, etiketi, fontu) Qt Designer içinde veya kod üzerinden ayarlanır.

Signal ve Slot Mekanizması: Butonların tıklanma olayları gibi kullanıcı etkileşimleri, belirli fonksiyonların tetiklenmesi için programlanır. Bu, Qt'nin signal ve slot mekanizması ile gerçekleştirilir.

4. ÜNİTE: FORMLAR

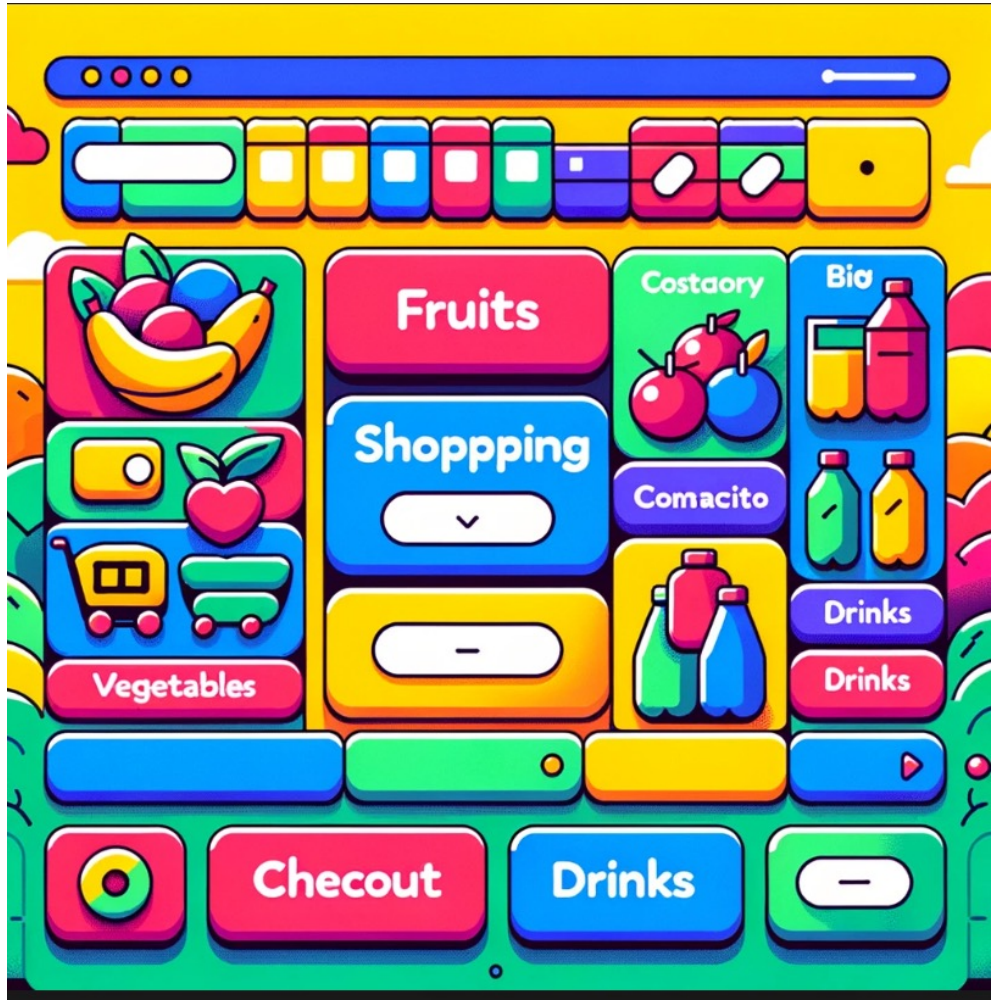
C) Konteyner sınıflarını (panel, grup kutusu, sekme kontrolü vb.) kullanarak formdaki bileşenleri düzenleme ve gruplama anlatılır.

Konteyner sınıflarını anlatırken, Bir alışveriş sepeti örneğini kullanabiliriz. Alışveriş sepeti, farklı türdeki ürünleri (meyveler, sebzeler, içecekler vb.) bir arada tutmanıza ve düzenlemenize olanak tanır. Her bir bölme ya da raf, belirli bir ürün grubunu saklamak için ayrılmıştır, bu da alışverişi daha organize hale getirir.

Panel (QWidget): Farklı widget'ları bir arada tutar. Bir alışveriş sepetinin kendisi gibi düşünülebilir, içinde farklı ürünler (widget'lar) barındırır.

Grup Kutusu (QGroupBox): İlgili widget'ları bir başlık altında gruplar. Bu, alışveriş sepetindeki belirli bir raf ya da bölme gibi, örneğin sadece içecekleri tutan bir bölüm.

Sekme Kontrolü (QTabWidget): Kullanıcıların farklı gruplar arasında kolayca geçiş yapmalarını sağlar, her sekme farklı bir widget setini gösterir. Alışveriş sepetinizde farklı katmanlar veya çekmeceler gibi düşünebilirsiniz, her biri farklı türde ürünler için.



4. ÜNİTE: FORMLAR

4.2. İletişim kutularını kullanır.

A) MessageBox, OpenFileDialog, SaveFileDialog iletişim kutularını kullanıcılarla iletişim kurmak için eklemesi

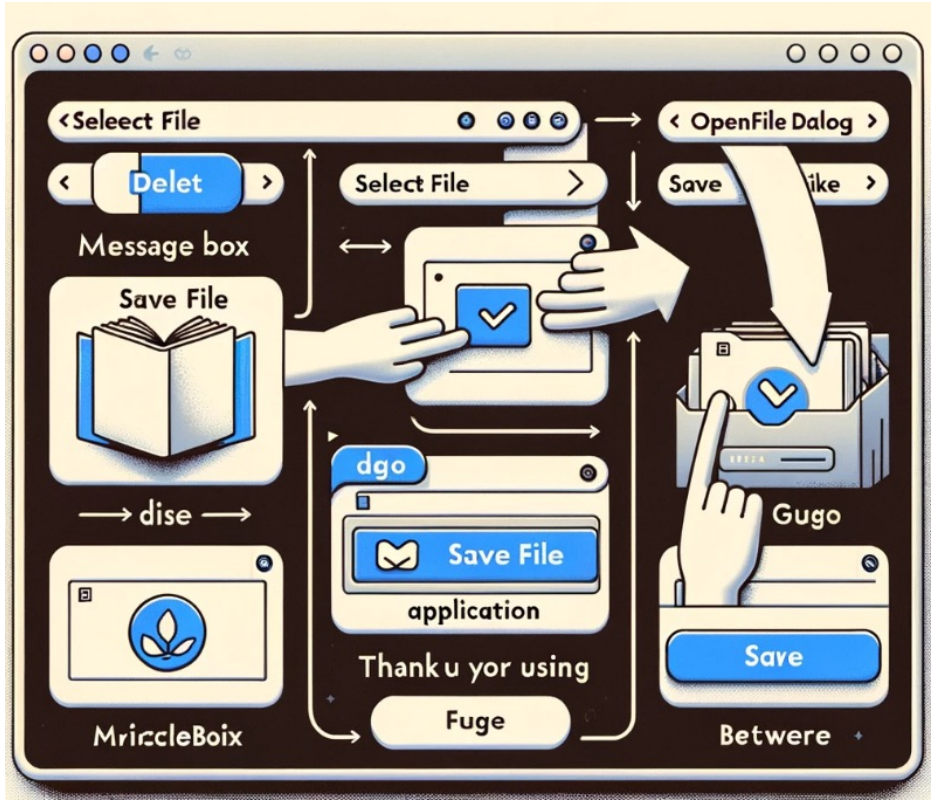
anlatılır.

C++ ile geliştirilen uygulamalarda kullanıcılarla iletişim kurmak için sıkça kullanılan iletişim kutularını (dialog boxes) anlatırken, günlük hayattaki fiziksel durumlarla ilişkilendirilebilir. Bu durumu, bir kitapçıda yaşanan bir alışveriş deneyimiyle örneklendirebiliriz:

MessageBox: Bu, kullanıcıya bilgi vermek, uyarıda bulunmak veya bir onay almak için kullanılır. Kitapçı örneğinde, mağazanın kapanmak üzere olduğunu belirten bir anons gibi düşünebilirsiniz. "Mağaza 10 dakika içinde kapanacak, lütfen alışverişinizi tamamlayın."

OpenFileDialog: Kullanıcıdan dosya seçmesini istediğinizde kullanılır. Kitapçıda bir kitap seçmeye benzetilebilir. Sanki müşteriye, "Hangi kitabı satın almak istersiniz?" diye soruyorsunuz ve müşteri raflardan bir kitap seçiyor.

SaveFileDialog: Kullanıcıdan bir dosyayı kaydetmek için konum ve isim belirlemesini istediğinizde kullanılır. Bu, kitapçıdan seçilen kitabı satın aldıktan sonra, çantanıza koyma işlemine benzer. Müşteriye, "Bu kitabı çantanızın hangi bölümüne koymak istersiniz?" diye sorar gibi bir durum.



4. ÜNİTE: FORMLAR

B) Kullanıcıya bilgi vermek, doğrulama yapmak veya hataları bildirmek için iletişim kutularını kullanarak kullanıcıdan veri alma ve uyarı mesajları ekleme örnekleri incelenir.

Bir sinema bileti rezervasyon sistemi düşünün. Kullanıcının bir film seçimi yapması, seçtiği filme uygun bir seans seçmesi ve ardından rezervasyonu onaylaması gerekiyor. Bu süreçte, kullanıcıdan veri almak, doğrulama yapmak ve olası hataları bildirmek için çeşitli iletişim kutuları kullanılır.

4. ÜNİTE: FORMLAR

Örnek Senaryo:

Film Seçimi:

Kullanıcı, bir film seçmek için bir dropdown menüsünden (ComboBox) seçim yapar.

Seans Seçimi:

Film seçildiğinde, kullanıcıya uygun seansları gösteren bir başka dropdown menü sunulur. Kullanıcı bir seans seçer.

Doğrulama ve Uyarı Mesajları:

Eğer kullanıcı herhangi bir film veya seans seçmeden "Rezerve Et" butonuna basarsa, bir MessageBox ile "Lütfen bir film ve seans seçiniz" uyarısı verilir.

Film ve seans seçimi yapıldıktan sonra "Rezerve Et" butonuna basıldığında, kullanıcıya seçimlerini doğrulaması için bir MessageBox gösterilir: "Seçtiğiniz film XXX, seans saati YYY. Rezervasyonu onaylıyor musunuz?"

Kullanıcı "Evet" dediğinde, rezervasyon tamamlanır ve "Rezervasyonunuz başarıyla tamamlandı" mesajıyla bilgilendirilir.

Kullanıcı "Hayır" dediğinde, işlem iptal edilir.

4. ÜNİTE: FORMLAR

C) Farklı senaryolar için iletişim kutularının mesaj içeriklerinin, başlıklarının ve düğme metinlerinin kişiselleştirilmesi uygulamaları ile dinamik ve kullanıcı dostu arayüzler oluşturulması durumları fark ettirilir.

Farklı senaryolar için iletişim kutularının mesaj içeriklerini, başlıklarını ve düğme metinlerini kişiselleştirerek dinamik ve kullanıcı dostu arayüzler oluşturma konseptini öğrenciye anlatmak için, günlük hayattan bir doğum günü partisi davetiyesi örneğini kullanabiliriz. Bu örneği, bir yazılım uygulamasındaki iletişim kutuları ile paralel bir şekilde açıklayabiliriz.

Bir doğum günü partisi düzenliyorsunuz ve farklı arkadaş gruplarınıza (okul arkadaşları, aile üyeleri, iş arkadaşları) göre kişiselleştirilmiş davetiyeler göndermek istiyorsunuz. Her davetiyenin içeriği, alıcının ilişkinize bağlı olarak değişir:

Okul arkadaşlarına: "Hey! Eğlenceli bir doğum günü partisine hazır mısın? Detaylar için kaydır!"

Aile üyelerine: "Sevgili ailem, sizinle birlikte olacağım bu özel günde sizi aramızda görmek isteriz. Lütfen katılımınızı onaylayın."

İş arkadaşlarına: "Profesyonel bir kutlama yapacağız ve sizin de aramızda olmanızı çok isterim. Lütfen katılım durumunuzu bildirin."

Bu, uygulamanızda kullanıcılara yönelik iletişim kurarken farklı iletişim kutuları kullanmanız gerektiğini gösterir. Mesajınızın tonu ve içeriği, hedef kitlenize ve durumunuza göre değişiklik göstermelidir.

4. ÜNİTE: FORMLAR

Yazılım Uygulaması:

Bir uygulama içinde kullanıcıya dosya kaydetme, silme veya bir işlemi onaylama gibi eylemler için iletişim kutuları kullanılırken, mesaj içeriği, başlık ve düğme metinleri o anki senaryoya göre kişiselleştirilmelidir:

Dosya kaydetme işlemi: "İşinizi kaydetmek ister misiniz?", Başlık: "Kaydetme Onayı", Düğmeler: ["Evet, Kaydet", "Hayır, İptal"]

Önemli bir dosyanın silinmesi: "Bu dosyayı silmek üzeresiniz. Devam etmek istiyor musunuz?", Başlık: "Silme Uyarısı", Düğmeler: ["Evet, Sil", "Hayır, Vazgeç"]

Bir işlemi onaylama: "İşlemi tamamlamak üzeresiniz. Onaylıyor musunuz?", Başlık: "Onay", Düğmeler: ["Onayla", "İptal"]

Bu yaklaşım, uygulamanın kullanıcı dostu ve etkileşimli olmasını sağlar, çünkü kullanıcıya verilen mesaj anlaşılır ve duruma özgüdür. Kullanıcılar, uygulamanın onlarla "konuştuğunu" ve spesifik durumlar için özel olarak tasarlandığını hisseder.

4. ÜNİTE: FORMLAR

4.3. Doğrulama ve veri bağlama işlemlerini öğrenir.

A) Kullanıcı tarafından sağlanan verilerin doğruluğunu kontrol etmek için kullanıcı girişi doğrulama işlemleri (validasyon) icra ettirilir.

Kullanıcı tarafından sağlanan verilerin doğruluğunu kontrol etmek için yapılan doğrulama (validasyon) işlemleri, bir uygulamanın güvenilir ve sağlam olmasını sağlar. Bu işlemler, kullanıcıların yanlışlıkla ya da kasıtlı olarak uygulamaya zarar verebilecek hatalı veriler girmesini önler.

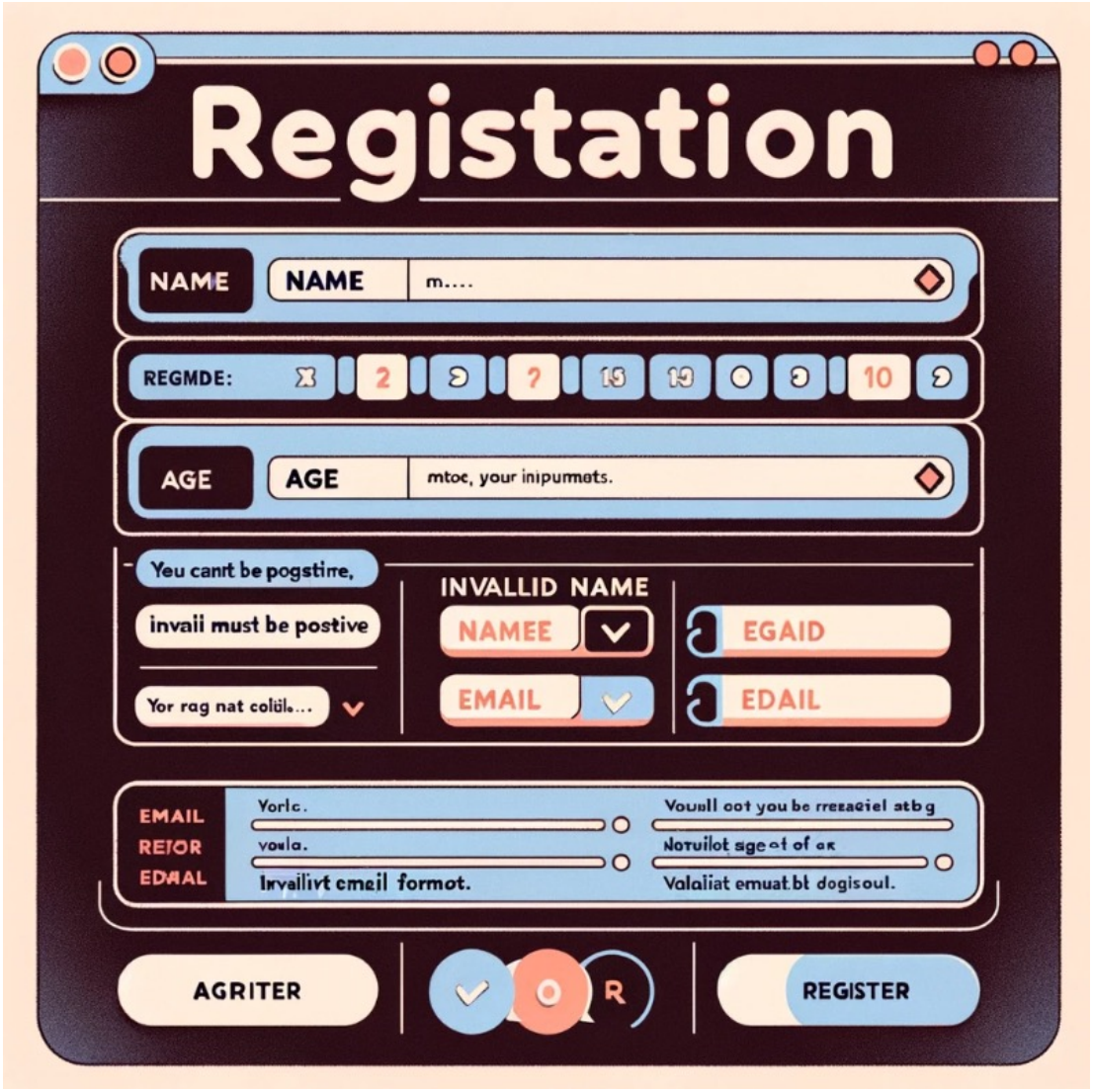
Örnek Senaryo ve Açıklama:

Senaryo: Bir kayıt formu düşünün. Bu formda kullanıcıdan ad, yaş ve e-posta adresi isteniyor. Her bir girişin doğruluğu, belirli kurallara göre kontrol edilir:

Ad: Sadece harfler içermelidir.

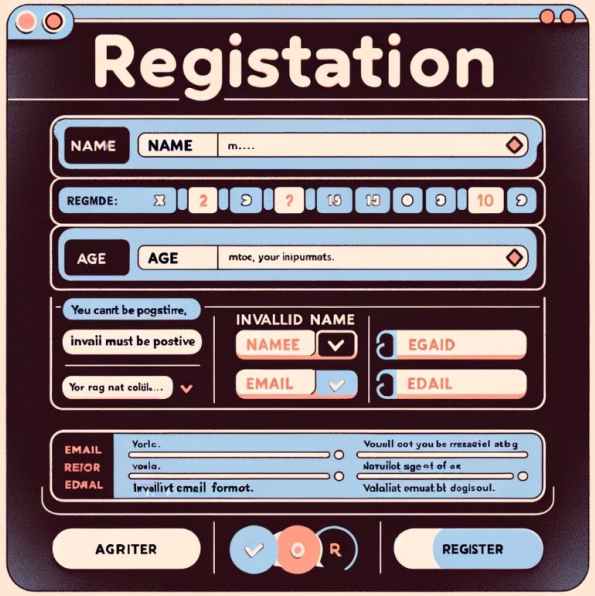
Yaş: Pozitif bir tam sayı olmalıdır.

E-posta Adresi: Geçerli bir e-posta formatına sahip olmalıdır (örneğin, kullanici@ornek.com).



4. ÜNİTE: FORMLAR

Form, 'Ad', 'Yaş' ve 'E-posta' alanlarını içerir ve alt kısımda bir 'Kaydol' butonu bulunur. Kullanıcı formu hatalı verilerle göndermeye çalıştığında, uygulama ilgili alanların yanında hata mesajları gösterir. Örneğin, 'Ad' alanında sayılar varsa, 'Yaş' negatifse veya 'E-posta' uygun formatta değilse, 'Geçersiz ad', 'Yaş pozitif olmalıdır' ve 'Geçersiz e-posta formatı' gibi hata mesajları görüntülenir. Arayüz, düzeltmelerin nerede gerektiğini açıkça belirterek, doğru veri gönderimini sağlamak için kullanıcı dostu bir şekilde tasarlanmıştır.




```cpp
#include <iostream>
#include <regex>

bool isValidName(const std::string& name) {
    // Ad sadece harfler içermelidir.
    return std::regex_match(name, std::regex("^[A-Za-z]+$"));
}

bool isValidAge(int age) {
    // Yaş pozitif bir tam sayı olmalıdır.
    return age > 0;
}

bool isValidEmail(const std::string& email) {
    // E-posta basit bir regex ile kontrol edilir.
    return std::regex_match(email, std::regex("^[\\w-\\.]+@([\\w-]+\\.)+[\\w-]{2,4}$"
}

int main() {
    std::string name, email;
    int age;

    std::cout << "Adınızı girin: ";
    std::cin >> name;
    std::cout << "Yaşınızı girin: ";
    std::cin >> age;
    std::cout << "E-posta adresinizi girin: ";
    std::cin >> email;

    if (!isValidName(name))
        std::cout << "Geçersiz ad. Lütfen sadece harf kullanın.\n";
    else if (!isValidAge(age))
        std::cout << "Geçersiz yaş. Yaş pozitif bir sayı olmalıdır.\n";
    else if (!isValidEmail(email))
        std::cout << "Geçersiz e-posta formatı.\n";
    else
        std::cout << "Kaydınız başarıyla tamamlandı!\n";

    return 0;
}
```

4. ÜNİTE: FORMLAR

B) Veri bağlama ve verileri kullanıcı arayüzüne bağlama işlemleri öğretilir.

Veri bağlama (data binding), bir uygulamanın veri kaynağı ile kullanıcı arayüzü (UI) bileşenleri arasındaki veri akışını otomatize eden bir teknikdir. Bu, verilerin UI'da otomatik olarak güncellenmesini sağlar, böylece kullanıcı arayüzü, uygulama verilerindeki değişiklikleri yansıtır. Bu işlem, manuel UI güncellemelerinin gerekliliğini ortadan kaldırır ve daha temiz, daha yönetilebilir kod yazımını teşvik eder.

Basit bir C++ ve Qt kullanarak veri bağlama işlemi gerçekleştiren bir uygulama tasarlayalım. Qt, veri bağlama için sinyal/slot mekanizmasını ve model/view mimarisini kullanarak güçlü bir destek sunar.

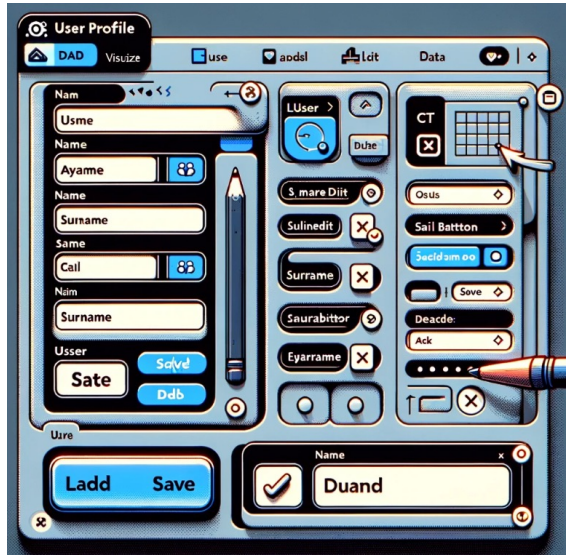
Görsel Tasarım: Formda, kullanıcının adı, soyadı ve e-postası için QLineEdit widget'ları yer alır. Bir "Kaydet" QPushButton ile kullanıcı değişiklikleri onaylar.

Veri Modeli Oluşturma: Kullanıcı bilgilerini tutacak basit bir model.

Veri Bağlama: QLineEdit widget'larının textChanged sinyallerini modeli güncelleyecek slotlara bağlama.

Kullanıcı Arayüzü ve Model Güncellemeleri: Kullanıcı veri girdiğinde veya düzenlediğinde, model ve UI otomatik olarak güncellenir.

Not: Aşağıdaki kod, Qt Creator ile bir Qt Widgets uygulaması oluştururken kullanılabilir. Qt'nin kurulu olduğu ve uygun şekilde yapılandırıldığı varsayılmaktadır




```cpp
// userProfile.h
#ifndef USERPROFILE_H
#define USERPROFILE_H

#include <QObject>

class UserProfile : public QObject {
    Q_OBJECT
    Q_PROPERTY(QString name READ name WRITE setName NOTIFY nameChanged)
    Q_PROPERTY(QString surname READ surname WRITE setSurname NOTIFY surnameChanged)
    Q_PROPERTY(QString email READ email WRITE setEmail NOTIFY emailChanged)

public:
    explicit UserProfile(QObject *parent = nullptr);

    QString name() const;
    void setName(const QString &name);

    QString surname() const;
    void setSurname(const QString &surname);

    QString email() const;
    void setEmail(const QString &email);

signals:
    void nameChanged();
    void surnameChanged();
    void emailChanged();

private:
    QString m_name;
    QString m_surname;
    QString m_email;
};
#endif // USERPROFILE_H
```

4. ÜNİTE: FORMLAR

C) Kullanıcı arayüzü ile etkileşimli uygulamalar geliştirmek için doğrulama ve veri bağlama örnekleri uygulatılır.

Kullanıcı arayüzü (UI) ile etkileşimli uygulamalar geliştirmek, modern yazılım geliştirmenin temel taşlarından biridir. Bu süreçte, kullanıcıdan alınan girdilerin doğruluğunu kontrol etmek (doğrulama) ve bu girdileri uygulamanın diğer bölümleriyle senkronize etmek (veri bağlama) önemlidir.

Senaryo : Bir kullanıcı kayıt formu düşünün. Bu formda kullanıcı adı, e-posta ve parola alanları yer alır. Kullanıcı adı en az 5 karakter, e-posta geçerli bir e-posta formatında ve parola en az 8 karakter olmalıdır. Kullanıcı formu doldurup "Kaydol" butonuna bastığında, her bir alanın doğruluğu kontrol edilir ve herhangi bir hata varsa kullanıcıya bildirilir.

4. ÜNİTE: FORMLAR

Kullanıcı adı, e-posta ve parola için üç QLineEdit widget'ı.

Doğrulama hatalarını göstermek için QLabel'lar.

Kullanıcı formu gönderdiğinde verileri işleyecek bir "Kaydol" QPushButton.

```cpp
// Qt ve C++ ile basit bir doğrulama ve veri bağlama örneği
#include <QApplication>
#include <QWidget>
#include <QLineEdit>
#include <QPushButton>
#include <QLabel>
#include <QVBoxLayout>
#include <QRegExpValidator>

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);

    QWidget window;
    window.setWindowTitle("Kullanıcı Kayıt Formu");

    QLineEdit *usernameEdit = new QLineEdit(&window);
    QLineEdit *emailEdit = new QLineEdit(&window);
    QLineEdit *passwordEdit = new QLineEdit(&window);
    passwordEdit->setEchoMode(QLineEdit::Password);

    QPushButton *registerButton = new QPushButton("Kaydol", &window);

    QLabel *errorLabel = new QLabel(&window);

    QVBoxLayout *layout = new QVBoxLayout(&window);
    layout->addWidget(usernameEdit);
    layout->addWidget(emailEdit);
    layout->addWidget(passwordEdit);
    layout->addWidget(registerButton);
    layout->addWidget(errorLabel);

    // Doğrulama kuralları
    QRegExpValidator *usernameValidator = new QRegExpValidator(QRegExp(".{5,}"), user
    QRegExpValidator *emailValidator = new QRegExpValidator(QRegExp("[\\w-]+@[\\w-]+\
    QRegExpValidator *passwordValidator = new QRegExpValidator(QRegExp(".{8,}"), pass

    usernameEdit->setValidator(usernameValidator);
    emailEdit->setValidator(emailValidator);
    passwordEdit->setValidator(passwordValidator);

    QObject::connect(registerButton, &QPushButton::clicked, [&]() {
        if (!usernameEdit->hasAcceptableInput() ||
            !emailEdit->hasAcceptableInput() ||
            !passwordEdit->hasAcceptableInput()) {
            errorLabel->setText("Hata: Lütfen formu doğru doldurun.");
        } else {
            errorLabel->setText("Kaydınız başarıyla alınmıştır.");
        }
    });

    window.setLayout(layout);
    window.show();

    return app.exec();
}
```

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

5.1. Veri tabanı yazılımlarını C++ arayüzü ile entegre eder.

A) Veri tabanı yazılımlarını kurma ve C++ uygulamalarıyla entegre etme anlatılır.

Veri tabanı yazılımları ve kullanım alanları,

C++ ve veri tabanı entegrasyonunun önemi,

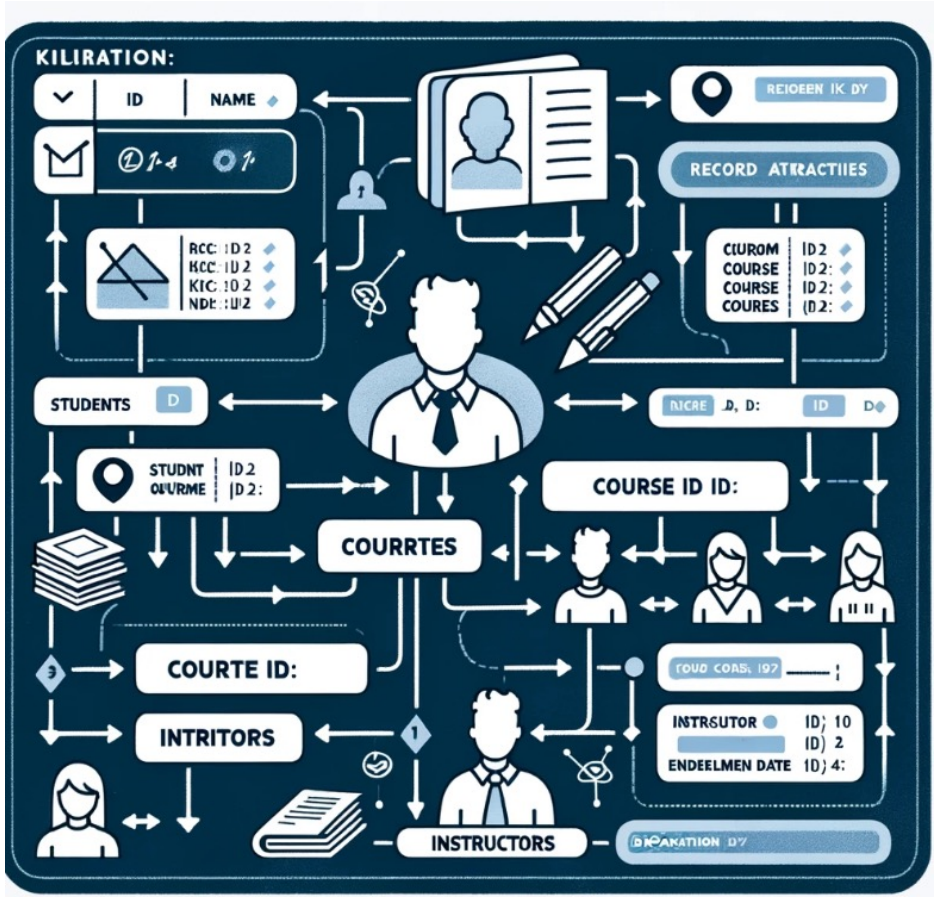
Kullanılacak veri tabanı yönetim sistemi (DBMS) ve C++ kütüphaneleri hakkında genel bilgi.

Veri Tabanı Yazılımı Kurulumu:

Seçilen DBMS'nin (MySQL, PostgreSQL, SQLite vb.) kurulum adımları,

Veri tabanı ve tabloların oluşturulması,

Basit veri ekleme, güncelleme, silme ve sorgulama işlemleri.



5. ÜNİTE: VERİ TABANI İŞLEMLERİ

B) Projelerde uygun veri tabanını seçmek için veri tabanı yazılım türleri öğretilir.

SQLite: SQLite, hafif bir ilişkisel veritabanı yönetim sistemi (RDBMS) ve C programlama dilinde yazılmıştır. SQLite, tek bir disk dosyasında bir SQL veritabanını barındırır ve C/C++ programları tarafından doğrudan kullanılabilir. Küçük çaplı projeler veya yerel depolama gerektiren uygulamalar için idealdir.

MySQL ve MariaDB: MySQL ve MariaDB, C++ ile yazılmış projelerde yaygın olarak kullanılan açık kaynaklı ilişkisel veritabanı yönetim sistemleridir. MySQL ve MariaDB, C API'leri aracılığıyla C++ programları ile entegre edilebilir. Özellikle web uygulamaları ve sunucu tabanlı sistemler için tercih edilirler.

PostgreSQL: PostgreSQL, özellikle güçlü SQL özellikleri, geniş uzantı desteği ve güvenilirliği ile bilinen bir açık kaynaklı ilişkisel veritabanı yönetim sistemidir. C++ programları PostgreSQL ile entegre edilebilir ve libpq C kütüphanesi aracılığıyla PostgreSQL veritabanına erişim sağlanabilir.

LevelDB ve RocksDB: LevelDB ve RocksDB, Google tarafından geliştirilmiş, düşük seviyede bir anahtar-değer deposudur. Bu veritabanları, özellikle performansı ön planda tutan uygulamalar için uygundur ve C++ ile kullanımı kolaydır.

LMDB (Lightning Memory-Mapped Database): LMDB, hafif bir anahtar-değer deposudur ve özellikle yüksek performanslı, düşük gecikmeli uygulamalar için uygundur. LMDB, C++ programları tarafından kullanılabilir ve doğrudan belleğe haritalama (memory-mapping) yöntemiyle çalışır.

Redis: Redis, hafızaya dayalı, anahtar-değer deposu ve mesaj aracısı olarak kullanılabilen bir veritabanı sistemidir. Redis, C++ programları ile entegre edilebilir ve redis-client gibi kütüphaneler aracılığıyla Redis sunucusuna erişim sağlanabilir.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

C) C++ programlama arayüzü ile veri tabanları arasında veri alışverişi yapma ve veri tabanı bağlantısını yönetme örnekleri inceletilir.

Veri Tabanları: İlk olarak, "veri tabanı" kavramını anlatmak önemlidir. Bir veri tabanı, bilgileri düzenli bir şekilde saklayan ve bu bilgilere erişim sağlayan bir sistemdir. Örneğin, bir kütüphane, bir mağaza veya bir okulun öğrenci bilgileri bir veri tabanında saklanabilir.

Örnekler

SQLite ile Veri Tabanına Bağlanma ve Sorgu Yapma: Öğrenciye basit bir veri tabanı olan SQLite'i tanıtın. SQLite, hafif bir veri tabanıdır ve C++ ile kullanılabilir. Öğrenciye, bir öğrenci veri tabanı oluşturduğunuzu ve bu veri tabanına C++ programı aracılığıyla nasıl erişebileceğinizi gösterin.

MySQL ile Veri Tabanına Bağlanma ve Veri Alışverişi Yapma: Öğrenciye daha büyük ölçekli bir veri tabanı olan MySQL'i tanıtın. MySQL, web siteleri ve büyük ölçekli uygulamalar için kullanılır. Öğrenciye, bir MySQL veritabanına C++ programı aracılığıyla nasıl bağlanacaklarını ve veri alışverişi yapacaklarını gösterin.

Veritabanları Arasında Veri Aktarımı: Öğrenciye, bir veri tabanından veri alıp, başka bir veri tabanına aktarmanın nasıl yapılabileceğini gösterin.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

5.2. Veri tabanı tasarımı yapar.

A) Veri tabanı tasarımı yapmayı ve veri tablolarını, alanlarını, ilişkilerini planlama öğretilir.

Senaryo: Bir Kütüphane Sistemi

Diyelim ki bir kütüphane sistemi tasarlamamız gerekiyor. Kütüphane sistemi, kitapların kataloglanması, üyelerin kaydı, ödünç alma işlemleri gibi birçok işlemi içerir. Bu senaryo için bir veri tabanı tasarlayalım.

1. Tablolar ve Alanlar:

Kitaplar Tablosu: Kitap ID (Primary Key), Kitap Adı, Yazar, Yayın Tarihi, Yayınevi, Kategori ID (Foreign Key)

Üyeler Tablosu: Üye ID (Primary Key), Ad, Soyad, Telefon, Adres

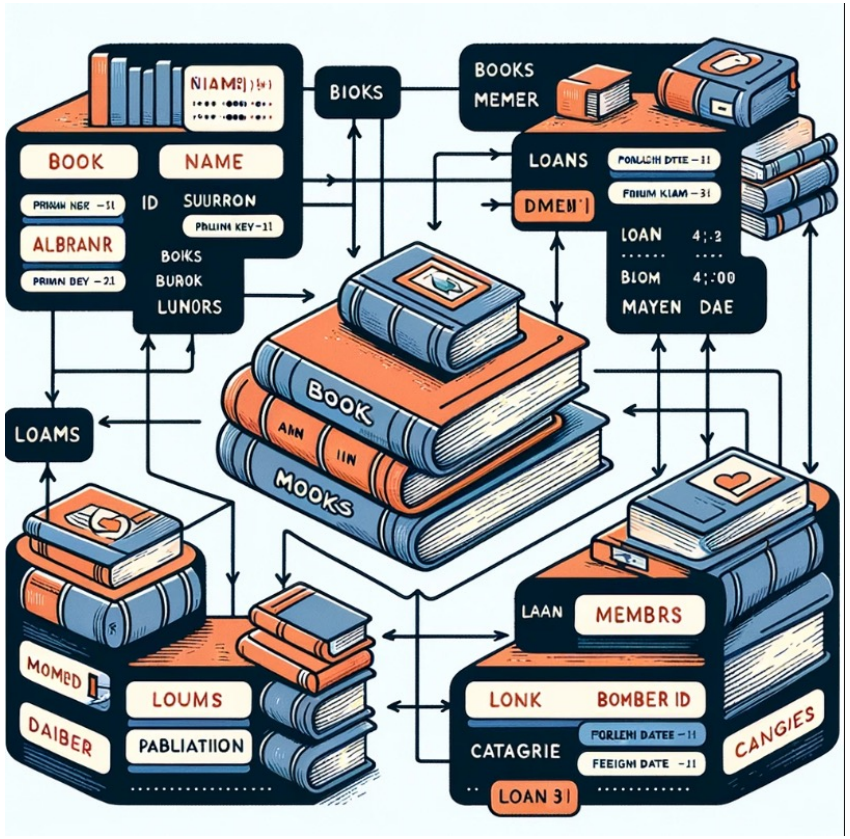
Ödünçler Tablosu: Ödünç ID (Primary Key), Üye ID (Foreign Key), Kitap ID (Foreign Key), Ödünç Tarihi, İade Tarihi

Kategoriler Tablosu: Kategori ID (Primary Key), Kategori Adı

2. İlişkiler:

Kitaplar ve Kategoriler arasında birçok birliktelik vardır. Bu nedenle, Kitaplar tablosundaki Kategori ID, Kategoriler tablosundaki Kategori ID'ye bir dış anahtar (Foreign Key) olarak atıfta bulunur.

Ödünçler tablosunda, Üye ID ve Kitap ID, sırasıyla Üyeler ve Kitaplar tablolarındaki ilgili alanlarla ilişkilendirilir.



5. ÜNİTE: VERİ TABANI İŞLEMLERİ

B) Veri tabanı şeması oluştururken verilerin tutarlı ve erişilebilir olmasını sağlayacak durumlar vurgulanır.

Normalizasyon: Veri tekrarının önlenmesi ve veri bütünlüğünün sağlanması için kullanılır. Veri tabanını mantıklı bir şekilde organize ederek, aynı verinin birden fazla yerde tutulmasını önler. Bu, veri güncellemelerinin, eklemelerinin ve silinmelerinin daha tutarlı ve hata yapma riskinin daha düşük olmasını sağlar.

Anahtar Kullanımı: Bir tablodaki kayıtları benzersiz bir şekilde tanımlayan bir veya daha fazla sütundur. Birincil anahtarlar (primary keys) ve yabancı anahtarlar (foreign keys) arasındaki ilişkiler, veri tabanındaki farklı tablolar arasında tutarlı bağlantılar kurar. Bu, veri tabanındaki ilişkisel bütünlüğü sağlar ve yanlış veri girişlerinin önüne geçer.

Veri Bütünlüğü Kuralları: Veri tabanına eklenen veya güncellenen verilerin belirli kurallara uymasını sağlar. Örneğin, bir öğrencinin notu sadece 0 ile 100 arasında olabilir. Bu tür kurallar, veri tabanına girilen verilerin mantıklı ve kullanılabilir olmasını sağlar.

Erişim İzinleri: Veri tabanına kimin erişebileceğini ve hangi verileri görebileceklerini kontrol eder. Bu, verilerin yalnızca yetkili kullanıcılar tarafından erişilebilir olmasını sağlar ve veri güvenliğini artırır.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

C) Farklı veri tabanı tasarım araçlarını kullanarak tasarım sürecini görselleştirme ve yönetme durumları örneklendirilir.

Veri tabanı tasarım sürecini görselleştirmek ve yönetmek için çeşitli araçlar mevcuttur. Bu araçlar, veri tabanı şemalarının oluşturulmasından, ilişkisel bağlantıların kurulmasına ve performans optimizasyonuna kadar geniş bir yelpazede destek sunar.

1. Entity Relationship Diagram (ERD) Araçları

Örnek Araçlar: Lucidchart, ER/Studio, Microsoft Visio

Kullanım Senaryoları: ERD araçları, veri tabanı tasarımcılarının entity'ler (varlıklar), entity özellikleri ve bu entity'ler arasındaki ilişkileri görselleştirmelerine olanak tanır. Bu, karmaşık veri yapılarını ve ilişkileri anlamayı kolaylaştırır. Örneğin, bir e-ticaret sistemi tasarlarken, ürünler, kullanıcılar, siparişler ve sipariş detayları gibi entity'ler arasındaki ilişkileri görselleştirmek için kullanılabilir.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

2. Veri Tabanı Tasarım ve Yönetim Araçları

Örnek Araçlar: MySQL Workbench, Oracle SQL Developer, Microsoft SQL Server Management Studio

Kullanım Senaryoları: Bu araçlar, veri tabanı şemalarının oluşturulması, sorgulama, veri tabanı yönetimi ve performans izleme gibi geniş bir yelpazedeki işlemleri destekler. Örneğin, bir veri tabanı yöneticisi, veri tabanı performansını izlemek ve sorgu optimizasyonu yapmak için bu tür bir aracı kullanabilir.

3. UML (Unified Modeling Language) Araçları

Örnek Araçlar: StarUML, Visual Paradigm

Kullanım Senaryoları: UML araçları, genellikle yazılım geliştirme sürecinde kullanılsa da, veri modelleme için de kullanılabilir. Bu araçlar, veri tabanı tasarımını sınıflar, paketler ve objeler gibi UML bileşenleri aracılığıyla modellemek için kullanılabilir. Örneğin, bir üniversite yönetim sisteminin veri tabanı tasarımını modellemek için kullanılabilir.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

4. Prototipleme Araçları

Örnek Araçlar: Axure RP, Sketch

Kullanım Senaryoları: Prototipleme araçları, genellikle kullanıcı arayüzü tasarımında kullanılsa da, veri tabanı tasarım süreçlerinde de faydalı olabilir. Bu araçlar, veri tabanıyla etkileşime girecek sistemlerin arayüz tasarımlarının prototiplerini oluşturmak için kullanılabilir. Örneğin, bir kütüphane yönetim sisteminin kullanıcı arayüzü tasarımında ve bu arayüzün veri tabanıyla olan etkileşimlerinde kullanılabilir.

5. NoSQL Veri Modelleme Araçları

Örnek Araçlar: MongoDB Compass, ArangoDB UI

Kullanım Senaryoları: NoSQL veri tabanları için tasarlanmış araçlar, belge tabanlı, anahtar-değer çiftleri, geniş sütun mağazaları ve graf veri tabanları gibi NoSQL veri modelleri üzerinde çalışır. Bu araçlar, ölçeklenebilir ve esnek veri modelleri oluşturmak için kullanılabilir. Örneğin, büyük veri setlerini işleyen bir sosyal medya analiz platformunun veri tabanı tasarımında kullanılabilir.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

5.3. Tablo işlemlerini gerçekleştirir.

A) Veri tabanında tablo oluşturma, veri ekleme, güncelleme ve silme işlemlerini öğrenir.

basit bir PERSON tablosu oluştururak, bu tabloya bir kayıt ekleme, eklenen kaydı güncelleme ve son olarak kaydı silme örneği ile konu anlatılabilir .

1. Tablo oluşturma

```
#include <iostream>
#include <sqlite3.h>

static int callback(void *NotUsed, int argc, char **argv, char **azColName){
    for(int i=0; i<argc; i++){
        std::cout << azColName[i] << " = " << (argv[i] ? argv[i] : "NULL") << std::en
    }
    std::cout << std::endl;
    return 0;
}

int main(int argc, char* argv[]){
    sqlite3* DB;
    char* messageError;
    int exit = sqlite3_open("example.db", &DB);
    std::string query = "CREATE TABLE IF NOT EXISTS PERSON("
                        "ID INT PRIMARY KEY NOT NULL, "
                        "NAME TEXT NOT NULL, "
                        "AGE INT NOT NULL);";

    exit = sqlite3_exec(DB, query.c_str(), NULL, 0, &messageError);
    if(exit != SQLITE_OK) {
        std::cerr << "Error Create Table" << std::endl;
        sqlite3_free(messageError);
    } else
        std::cout << "Table created Successfully" << std::endl;
```

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

5.3. Tablo işlemlerini gerçekleştirir.

A) Veri tabanında tablo oluşturma, veri ekleme, güncelleme ve silme işlemlerini öğrenir.

1. Veri Ekleme

```
// Veri ekleme
query = "INSERT INTO PERSON (ID, NAME, AGE) VALUES (1, 'John Doe', 30);";
exit = sqlite3_exec(DB, query.c_str(), callback, NULL, &messageError);
if(exit != SQLITE_OK) {
    std::cerr << "Error Insert" << std::endl;
    sqlite3_free(messageError);
} else
    std::cout << "Records created Successfully" << std::endl;

// Veri güncelleme
query = "UPDATE PERSON SET AGE = 31 WHERE ID = 1;";
exit = sqlite3_exec(DB, query.c_str(), callback, NULL, &messageError);
if(exit != SQLITE_OK) {
    std::cerr << "Error Update" << std::endl;
    sqlite3_free(messageError);
} else
    std::cout << "Record updated Successfully" << std::endl;
```

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

5.3. Tablo işlemlerini gerçekleştirir.

A) Veri tabanında tablo oluşturma, veri ekleme, güncelleme ve silme işlemlerini öğrenir.

1. Veri Güncelleme

```
// Veri güncelleme
query = "UPDATE PERSON SET AGE = 31 WHERE ID = 1;";
exit = sqlite3_exec(DB, query.c_str(), callback, NULL, &messageError);
if(exit != SQLITE_OK) {
    std::cerr << "Error Update" << std::endl;
    sqlite3_free(messageError);
} else
    std::cout << "Record updated Successfully" << std::endl;
```

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

5.3. Tablo işlemlerini gerçekleştirir.

A) Veri tabanında tablo oluşturma, veri ekleme, güncelleme ve silme işlemlerini öğrenir.

1. Veri Silme

```
    // Veri silme
    query = "DELETE FROM PERSON WHERE ID = 1;";
    exit = sqlite3_exec(DB, query.c_str(), callback, NULL, &messageError);
    if(exit != SQLITE_OK) {
        std::cerr << "Error Delete" << std::endl;
        sqlite3_free(messageError);
    } else
        std::cout << "Record deleted Successfully" << std::endl;

    sqlite3_close(DB);
    return 0;
}
```

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

B) Veri tabanında oluşturulan tabloları sorgulama ve filtreleme yeteneklerini yönetme çalışılır.

Veri tabanındaki tabloları sorgulama ve filtreleme, veri tabanı yönetiminde temel becerilerden biridir. Bu yetenekler, belirli kriterlere göre veri çekmek ve analiz etmek için çok önemlidir. Bu süreçte SQL dilinin SELECT ifadesi ve çeşitli filtreleme koşulları (WHERE, LIKE, IN, vb.) kullanılır.

örnek, C++ ve SQLite kullanarak basit bir sorgulama ve filtreleme işlemi gösterilir. Bu örnekte, daha önce oluşturulmuş olan PERSON tablosundan belirli kayıtları sorgulanabilir.

Öncelikle, sorgulama işlemi için bir callback fonksiyonu tanımlanır. Bu fonksiyon, sorgu sonucundaki her bir satır için çağrılır ve sonuçları ekrana basar.

Adım 1: Callback Fonksiyonu Tanımlama

cpp — Copy code

```cpp
#include <iostream>
#include <sqlite3.h>

static int callback(void *NotUsed, int argc, char **argv, char **azColName){
    for(int i=0; i<argc; i++){
        std::cout << azColName[i] << " = " << (argv[i] ? argv[i] : "NULL") << std::en
    }
    std::cout << std::endl;
    return 0;
}
```

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

Adım 2: Sorgulama ve Filtreleme İşlemi

cpp — Copy code

```cpp
int main(int argc, char* argv[]){
    sqlite3* DB;
    char* messageError;
    int exit = sqlite3_open("example.db", &DB);

    // Basit bir sorgulama işlemi
    std::string query = "SELECT * FROM PERSON;";
    exit = sqlite3_exec(DB, query.c_str(), callback, NULL, &messageError);
    if(exit != SQLITE_OK) {
        std::cerr << "Error Select" << std::endl;
        sqlite3_free(messageError);
    } else
        std::cout << "Records selected Successfully" << std::endl;

    // Filtreleme işlemi (WHERE koşulu kullanarak)
    query = "SELECT * FROM PERSON WHERE AGE > 25;";
    exit = sqlite3_exec(DB, query.c_str(), callback, NULL, &messageError);
    if(exit != SQLITE_OK) {
        std::cerr << "Error Select with WHERE" << std::endl;
        sqlite3_free(messageError);
    } else
        std::cout << "Records filtered by age Successfully" << std::endl;

    sqlite3_close(DB);
    return 0;
}
```

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

C) İşlevsel uygulamalar için tablo işlemlerini iş mantığıyla entegre etme durumu fark ettirilir.

1. İş Mantığının Anlaşılması

İş mantığı, bir uygulamanın nasıl çalıştığını ve kullanıcıların ihtiyaçlarını nasıl karşıladığını tanımlayan kurallar ve algoritmalardır. Öğrencilere, uygulamanın genel amacını ve kullanıcıların hangi işlemleri gerçekleştirebileceğini anlamaları gerektiğini açıklanabilir**.**

2. Veri Tabanı Tasarımı ve İş Mantığı Arasındaki İlişki

Veri tabanı tasarımının, uygulamanın iş mantığına hizmet etmesi gerektiğini vurgulanır. Örneğin, bir e-ticaret uygulamasında, kullanıcılar tarafından yapılan siparişlerin takibi için siparişler, ürünler ve kullanıcılar arasında ilişkilerin kurulması gerekir. Bu ilişkilerin nasıl modelleneceğini ve iş mantığıyla nasıl entegre edilmesi gerektiğine değinilir.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

3. CRUD İşlemleri ve İş Mantığı

CRUD (Create, Read, Update, Delete) işlemlerinin, uygulamanın çeşitli işlevlerini desteklemek için nasıl kullanıldığını örneklerle gösterin. Örneğin, bir kullanıcı profil güncelleme işlemi, bir "Update" işlemine nasıl dönüşebilir veya yeni bir siparişin sisteme eklenmesi bir "Create" işlemi olarak nasıl ele alınabilir açıklayın.

4. İşlemlerin Otomasyonu ve İyileştirilmesi

Bazı işlemlerin nasıl otomatize edilebileceği ve iş mantığına göre nasıl iyileştirilebileceğinden bahsedin. Örneğin, bir kullanıcının sipariş geçmişini otomatik olarak sorgulayıp listelemek veya belirli kriterlere göre ürün önerileri yapmak gibi.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

5. Hata Yönetimi ve Güvenlik

Veri tabanı işlemlerinin uygulama içerisinde güvenli bir şekilde nasıl gerçekleştirileceği ve olası hataların nasıl yönetileceği üzerinde durun. Özellikle, veri girişi sırasında hata kontrolü, SQL enjeksiyonu gibi güvenlik tehditlerine karşı önlemler ve kullanıcı verilerinin korunması gibi konuları ele alın.

6. Gerçek Dünya Örnekleriyle Uygulama

Öğrencilere, gerçek dünya uygulamalarından örnekler sunarak, teorik bilgilerin pratiğe nasıl dönüştürüleceğini gösterin. Bu, öğrencilerin öğrendiklerini somut örnekler üzerinden daha iyi anlamalarını ve kavramalarını sağlar.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

5.4. Temel SQL komutlarını kullanır.

A) Select, Insert, Update, Delete temel SQL komutlarını kullanarak verileri çekme, veri ekleme, var olan veriyi güncelleme ve silme işlemleri uygulatılır.

Kütüphanesi yönetim sistemi için bir veri tabanı tasarlatılabilir. ve bu sistem, kitapları, kütüphaneye kayıtlı üyeleri ve ödünç alınan kitapları yönetir. Bu senaryoda, kütüphane sistemine yeni kitaplar eklemek, üye kaydı oluşturmak, üyelerin ödünç aldıkları kitapları güncellemek ve gerektiğinde kitap ya da üye kayıtlarını silmek gibi işlemleri gerçekleştirilir**.**

Görevler:

Kitap Ekleme:

INSERT komutunu kullanarak, kitaplar tablosuna yeni bir kitap ekleyin. Kitap için gerekli bilgiler; kitap adı, yazarı, yayın yılı ve kategori olmalıdır.

Üye Kaydı Ekleme:

INSERT komutu ile üyeler tablosuna yeni bir üye ekleyin. Üye için gerekli bilgiler; üye adı, soyadı, telefon numarası ve e-posta adresi olmalıdır.

Ödünç Alınan Kitap Bilgisi Güncelleme:

Bir üyenin ödünç aldığı kitap bilgilerini güncellemek için UPDATE komutunu kullanın. Örneğin, ödünç alma tarihini veya iade tarihini güncelleyin.

Kitap Sorgulama:

SELECT komutunu kullanarak, belirli bir kriteri (örneğin, yazar adına göre) karşılayan kitapların listesini çıkarın.

Üye Silme:

Belirli bir kriteri (örneğin, üyelik süresi dolmuş olanlar) karşılayan üyeleri bulmak için SELECT kullanın ve bu üyeleri DELETE komutu ile sistemden silin.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

B) Veri tabanı sorguları oluştururken dinamik parametrelerle çalışma ve güvenli sorgu oluşturma becerileri geliştirilir.

Veri tabanı sorguları oluştururken dinamik parametreler kullanmak ve güvenli sorgular oluşturma becerilerini geliştirmek, veri tabanı yönetiminde çok önemli bir yere sahiptir. Bu beceriler, özellikle web uygulamaları ve diğer yazılım projelerinde, SQL injection gibi güvenlik açıklarını önlemeye yardımcı olur.

Dinamik Parametrelerle Çalışma

Dinamik Sorguların Tanıtımı: Uygulamaların kullanıcı girdilerine veya programın akışına göre değişken sorgular oluşturması gerekebilir. Bu tür sorgular, daha esnek ve etkileşimli uygulamalar geliştirmeye olanak tanır.

Parametre Kullanımının Önemi: Sorguların dinamik olarak oluşturulması sırasında, sabit değerler yerine parametrelerin kullanılması gerektiğini vurgulayın. Parametreler, sorgu metni ile kullanıcı girdilerinin ayrılmasını sağlayarak, SQL injection saldırılarına karşı koruma sağlar.

Örnek Uygulama Senaryoları: Web formundan alınan bir kullanıcı adını kullanarak o kullanıcıya ait bilgileri çekmek, veya belirli bir tarihten sonra yayınlanmış kitapları listelemek gibi dinamik sorguların pratik örneklerini sunun.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

Güvenli Sorgu Oluşturma

SQL Injection ve Güvenlik: SQL injection saldırılarının ne olduğunu, nasıl gerçekleştirildiğini ve bu tür saldırılara karşı korunmanın önemini anlatın. Güvenli sorgu teknikleri kullanılarak bu tür saldırılardan nasıl kaçınılacağı üzerinde durun.

Hazırlanmış Sorgular (Prepared Statements): Hazırlanmış sorguların, parametre değerlerini sorgu metninden ayrı tutarak SQL injection saldırılarına karşı nasıl bir koruma sağladığını açıklayın. Parametrelerin sorgu yürütülmeden önce belirlendiğini ve bu sayede sorgunun yapısının değiştirilemeyeceğini belirtin.

Veri Doğrulama ve Temizleme: Kullanıcıdan alınan girdilerin her zaman doğrulanması ve gerekirse temizlenmesi gerektiğini vurgulayın. Bu sürecin, beklenmeyen veya zararlı verilerin veri tabanına girilmesini önlemek için önemli olduğunu açıklayın.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

Uygulama ve Alıştırmalar

Pratik Alıştırmalar: Öğrencilere, dinamik parametreler kullanarak ve hazırlanmış sorgular oluşturarak güvenli sorgular yazmaları için pratik alıştırmalar verin. Bu, onların teorik bilgilerini pratiğe dökmelerine ve konuyu daha iyi anlamalarına yardımcı olur.

Güvenlik Senaryoları Analizi: Gerçek hayattan örneklerle, SQL injection saldırılarının nasıl önlenmiş olabileceğini tartışın. Bu, öğrencilerin güvenlik konularına daha duyarlı hale gelmelerini ve geliştirdikleri uygulamalarda güvenlik önlemlerini dikkate almalarını sağlar.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

C) Karmaşık SQL sorguları (Join operatörleri, alt sorgular, indeksleme vb.) ile veri tabanlarını etkili bir

şekilde sorgulama vurgulanır.

Karmaşık SQL sorguları, bir veri tabanından istenilen bilgileri elde etmek için kullanılan güçlü araçlardır. Join operatörleri, alt sorgular ve indeksleme gibi konseptler, veri tabanı sorgulama işlemlerini daha etkili ve verimli hale getirir.

Join Operatörleri

Join operatörleri, iki veya daha fazla tablo arasındaki ilişkiyi kullanarak, bu tablolardan gelen verilerin birleştirilmiş bir görünümünü oluşturmak için kullanılır. Bu, ilişkisel veri tabanlarının temel bir özelliğidir ve çeşitli join türleri vardır (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN).

Örnek: Bir kitapçı veri tabanını düşünün. Kitaplar tablosunda kitapların detayları ve Yazarlar tablosunda yazarların detayları bulunsun. Kitapların ve yazarların detaylarını birleştirip, hangi kitabın hangi yazar tarafından yazıldığını gösteren bir sorgu yazmak istiyorsunuz. Bu durumda, Kitaplar ve Yazarlar tablolarını yazar ID'si üzerinden JOIN edebilirsiniz.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

Alt Sorgular

Alt sorgular, bir SQL sorgusunun içinde yer alan başka bir SQL sorgusudur. Genellikle, ana sorguya veri sağlamak, koşul belirtmek veya sorgunun sonuç setini filtrelemek için kullanılır.

Örnek : Yine kitapçı veri tabanını ele alalım. En çok satan kitapların listesini elde etmek istiyorsunuz. Bu bilgiyi, satış sayılarına göre sıralanmış ve sadece en üstteki kitapları seçen bir alt sorgu ile elde edebilirsiniz.

İndeksleme

İndeksleme, veri tabanında sorgu performansını artırmak için kullanılan bir yöntemdir. Sorguların daha hızlı çalışmasını sağlamak amacıyla, sık kullanılan sütunlar üzerinde indeksler oluşturulur.

Örnek : Kitapçı veri tabanında, kullanıcıların sıklıkla yazar soyadına göre kitap aradıklarını düşünün. Yazarlar tablosunun soyadı sütununa bir indeks eklemek, bu tür sorguların çok daha hızlı çalışmasını sağlayacaktır.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

5.5. İlişkisel veri tabanları oluşturur.

A) Farklı tablolar arasındaki ilişkileri çözmesi için ilişkisel veri tabanlarının kullanımı öğretilir.

İlişkisel veri tabanları, veriler arasındaki ilişkileri tanımlayarak veri organizasyonu ve sorgulama işlemlerini kolaylaştırır.

1. İlişkisel Veri Tabanı Modellerinin Tanıtılması

İlişkisel Modelin Temelleri: İlişkisel veri tabanı modellerinin tablolara (veya "relation"lara) dayalı olduğunu açıklayın. Her tablo, benzer verilerin satır (kayıtlar) ve sütunlar (alanlar) olarak düzenlendiği bir yapıdır.

Birincil Anahtarlar ve Yabancı Anahtarlar: Birincil anahtarların her kaydı benzersiz bir şekilde tanımlamak için nasıl kullanıldığını ve yabancı anahtarların farklı tablolar arasındaki ilişkileri kurmak için nasıl kullanıldığını açıklayın.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

2. Tablolar Arası İlişkilerin Çeşitleri

Bir-bire İlişkiler: İki tablo arasında bir kaydın diğer tabloda yalnızca bir karşılığı olduğu durumları açıklayın.

Bir-çoka İlişkiler: Bir tablodaki bir kaydın, diğer tabloda birden çok kayıtla ilişkilendirilebileceği senaryoları tanıtın.

Çoka-çoka İlişkiler: Her iki tabloda da birden çok kaydın birbiriyle ilişkilendirilebileceği durumları açıklayın ve bu tür ilişkilerin yönetimi için ara tabloların (ilişki tabloları) nasıl kullanıldığını gösterin.

3. İlişkisel Veri Tabanı Sorgulama

Join Operatörlerini Kullanma: Farklı tablolar arasındaki ilişkileri sorgulamak için JOIN operatörlerinin (INNER JOIN, LEFT JOIN vb.) nasıl kullanıldığını anlatın. Gerçek hayattan örneklerle, bu operatörlerin kullanımını gösterin.

Kompleks Sorgular: Farklı tabloları birleştiren, koşullu ifadeler içeren ve alt sorgular kullanarak daha kompleks veri çekme işlemlerini nasıl yapabileceklerini öğretin.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

4. Pratik Uygulamalar ve Alıştırmalar

Senaryo Tabanlı Örnekler: Öğrencilere, gerçek dünya senaryolarını modelleyen ve farklı tablolar arası ilişkileri içeren örnek veri tabanı tasarımları sunun. Örneğin, bir okul sistemi, bir e-ticaret platformu veya bir kütüphane yönetim sistemi olabilir**.**

Alıştırmalar: Öğrencilere, öğrendikleri ilişkisel modelleme tekniklerini kullanarak verilen senaryolar üzerinde çalışmaları için görevler verin. Bu, onların ilişkisel veri tabanı tasarımı ve sorgulama becerilerini pekiştirmelerine yardımcı olur.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

B) Veri tabanı modellemesi için veri tabanı şeması düzgün biçimde normalleştirilir.

Veri tabanı normalleştirme, veri tekrarını azaltmak, veri bütünlüğünü artırmak ve veri tabanı işlemlerinin etkinliğini iyileştirmek için kullanılan bir dizi tekniktir. Veri tabanı modellemesinde normalleştirme sürecini öğrencilere anlatırken, konsepti basitleştirmek ve adım adım bir yaklaşım benimsemek önemlidir.

1. Normalleştirme Nedir ve Neden Önemlidir

Tanım: Normalleştirme, veri tabanı şemasını düzenlemek için kullanılan bir yöntemdir. Veri tekrarını önler, veri bütünlüğünü sağlar ve veri tabanı sorgularının daha verimli çalışmasına yardımcı olur.

Önemi: Veri tabanında tutarlılık ve etkinlik sağlar. Veri ekleme, güncelleme ve silme işlemlerinde oluşabilecek sorunları minimize eder.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

2. Normal Formlar

Birinci Normal Form (1NF): Her sütunun atomik değerler içermesi ve her satırın benzersiz bir anahtarla tanımlanması gerekir. Örnek olarak, bir adres sütunu 'Sokak', 'Şehir', 'Posta Kodu' gibi ayrı sütunlara bölünebilir.

İkinci Normal Form (2NF): 1NF'de olup, ayrıca tüm sütunların, tablonun anahtarına tam bağımlı olması gerekir. Yani, birincil anahtarın bir parçasına bağımlı olmamalıdır.

Üçüncü Normal Form (3NF): 2NF'de olup, ayrıca tüm sütunların, anahtar olmayan sütunlara bağımlı olmamalıdır. Yani, her sütun sadece birincil anahtara bağlı olmalıdır.

Boyce-Codd Normal Formu (BCNF): 3NF'de olup, daha katı bir kural seti sunar. Her determinantın kandidat anahtar olmasını gerektirir.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

3. Normalleştirme Süreci

Veri Tekrarının Tespiti: Veri tekrarını belirlemek için tablolarınızdaki verileri inceleyin. Veri tekrarını azaltmak için, ilgili verileri ayrı tablolara ayırmayı düşünün.

Anahtar Atama: Her tabloya benzersiz birincil anahtarlar atayın. Bu, tablolar arasındaki ilişkileri kurarken kullanılacaktır.

İlişkilerin Tanımlanması: Tablolar arasındaki ilişkileri belirleyin ve yabancı anahtarlar kullanarak bu ilişkileri oluşturun.

4. Pratik Uygulamalar

Örnek Senaryolar: Öğrencilere, normalleştirilmemiş veri tabanı şemaları verin ve onlardan bu şemaları adım adım normalleştirmelerini isteyin. Her normal forma geçişte, yapılan değişiklikleri ve bunun veri tabanı üzerindeki etkisini tartışın.

Alıştırmalar: Öğrencilere çeşitli senaryolar sunarak, kendilerinin veri tabanı şemalarını oluşturmalarını ve normalleştirme sürecini uygulamalarını sağlayın. Bu, teorik bilgilerin pratikle pekiştirilmesine yardımcı olur.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

C) İlişkisel veri tabanlarının verileri tutma ve ilişkilendirme yeteneklerini kullanarak karmaşık veri tabanı

yapıları oluşturulur.

1. Tabloları ve İlişkileri Tasarlamak

Tablo Tasarımı: Her tablo, belirli bir veri kümesini temsil etmelidir. Örneğin, bir e-ticaret sistemi için "Ürünler", "Müşteriler" ve "Siparişler" gibi tablolar oluşturulabilir. Her tablonun, o tabloya özgü verileri içeren sütunlara ihtiyacı vardır.

İlişkilerin Belirlenmesi: Tablolar arasındaki ilişkileri tanımlayın. Bu, birincil ve yabancı anahtarlar kullanılarak yapılır. Örneğin, "Siparişler" tablosundaki bir yabancı anahtar, "Müşteriler" tablosundaki bir müşteriyi o siparişe bağlar.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

2. Normalleştirme

Veri Bütünlüğü ve Tutarsızlık Önleme: Normalleştirme işlemi, veri tekrarını azaltmaya ve veri bütünlüğünü sağlamaya yardımcı olur. Normalleştirilmiş veri tabanları, güncelleme, ekleme ve silme işlemleri sırasında tutarsızlık riskini azaltır.

3. Join Operasyonları

Karmaşık Sorgular: İlişkisel veri tabanlarında, farklı tablolardan gelen verileri birleştirmek için JOIN operasyonları kullanılır. Bu, özellikle karmaşık veri tabanı yapılarında, ilgili verilere kolayca erişim sağlar.

4. İndeksleme

Sorgu Performansı: Karmaşık veri tabanı yapılarında, sorgu performansını optimize etmek için indeksler kullanılır. İndeksleme, sık sorgulanan sütunlarda aramaları hızlandırır ve veri erişim süresini kısaltır.

5. Güvenlik ve Erişim Kontrolü

Veri Güvenliği: İlişkisel veri tabanlarında, verilere kimin erişebileceğini kontrol etmek önemlidir. Rol tabanlı erişim kontrolleri ve izinler, verilerin yalnızca yetkili kullanıcılar tarafından görüntülenmesini ve değiştirilmesini sağlar.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

2. Normalleştirme

Veri Bütünlüğü ve Tutarsızlık Önleme: Normalleştirme işlemi, veri tekrarını azaltmaya ve veri bütünlüğünü sağlamaya yardımcı olur. Normalleştirilmiş veri tabanları, güncelleme, ekleme ve silme işlemleri sırasında tutarsızlık riskini azaltır.

3. Join Operasyonları

Karmaşık Sorgular: İlişkisel veri tabanlarında, farklı tablolardan gelen verileri birleştirmek için JOIN operasyonları kullanılır. Bu, özellikle karmaşık veri tabanı yapılarında, ilgili verilere kolayca erişim sağlar.

4. İndeksleme

Sorgu Performansı: Karmaşık veri tabanı yapılarında, sorgu performansını optimize etmek için indeksler kullanılır. İndeksleme, sık sorgulanan sütunlarda aramaları hızlandırır ve veri erişim süresini kısaltır.

5. Güvenlik ve Erişim Kontrolü

Veri Güvenliği: İlişkisel veri tabanlarında, verilere kimin erişebileceğini kontrol etmek önemlidir. Rol tabanlı erişim kontrolleri ve izinler, verilerin yalnızca yetkili kullanıcılar tarafından görüntülenmesini ve değiştirilmesini sağlar.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

Bir e-ticaret platformu düşünün. Bu platformda, müşteriler, ürünler ve siparişler olacak şekilde ilişkisel bir veri tabanı tasarlamak istiyorsunuz**:**

Müşteriler Tablosu: MüşteriID, İsim, Eposta gibi sütunları içerir.

Ürünler Tablosu: ÜrünID, ÜrünAdı, Fiyat gibi sütunları içerir.

Siparişler Tablosu: SiparişID, MüşteriID (Yabancı Anahtar), ÜrünID (Yabancı Anahtar), SiparişTarihi gibi sütunları içerir..

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

5.6. C++ programlama arayüzü ile oluşturulan veri tabanlarına bağlantı sağlar.

A) C++ programlama arayüzünde oluşturulan tablolardan veri çekme, veri güncelleme ve veri silme temel

işlemleri uygulatılır.

C++ programlama dili ve SQL kullanarak bir araba veri tabanı yönetim sistemi üzerinde çalışan bir uygulama geliştirdiğinizi düşünün. Bu sistemde, "Arabalar" adında bir tablo var ve bu tablo, her bir araba için ArabaID, Marka, Model, Yıl ve Renk gibi bilgileri içeriyor. Bu senaryoda, öğrencilerin C++ ve SQL bilgilerini kullanarak veri çekme, güncelleme ve silme işlemlerini gerçekleştirmeleri isteniyor.

Görevler

Veri Çekme (SELECT): "Arabalar" tablosundan tüm kayıtları çeken bir C++ fonksiyonu yazın. Bu fonksiyon, veri tabanındaki tüm arabaların listesini kullanıcıya göstermelidir.

Veri Güncelleme (UPDATE): Kullanıcıdan bir ArabaID alın ve bu ID'ye sahip arabanın rengini kullanıcının gireceği yeni bir renk ile güncelleyen bir C++ fonksiyonu yazın. Örneğin, kullanıcı bir arabanın rengini "Mavi"den "Kırmızı"ya değiştirmek isteyebilir.

Veri Silme (DELETE): Kullanıcıdan bir ArabaID alın ve bu ID'ye sahip arabanın veri tabanından tamamen silinmesini sağlayan bir C++ fonksiyonu yazın. Bu işlem, belirtilen ID'ye sahip arabanın "Arabalar" tablosundan kaldırılmasını sağlar.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

Arabalar Tablosu Örneği

ArabaID	Marka	Model	Yıl	Renk
1	Toyota	Corolla	2020	Mavi
2	Honda	Civic	2019	Beyaz
3	Ford	Mustang	2018	Kırmızı

Soru

Bir C++ uygulaması geliştiriyorsunuz ve bu uygulama, kullanıcıların "Arabalar" veri tabanı tablosu üzerinde çeşitli işlemler yapmasına olanak tanıyor. Uygulamanızın şu işlevleri yerine getirmesini sağlayacak kodları yazın:

Listeleme İşlemi: Kullanıcıya "Arabalar" tablosundaki tüm arabaların bir listesini gösterin.

Güncelleme İşlemi: Kullanıcıdan bir ArabaID ve yeni bir renk bilgisi alarak, o ID'ye sahip arabanın rengini güncelleyin.

Silme İşlemi: Kullanıcıdan bir ArabaID alarak, o ID'ye sahip arabanın veri tabanından silinmesini sağlayın**.**

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

B) Veri tabanı etkileşimli formlar tasarlatarak kullanıcıların arayüz üzerinden verilere erişme senaryoları

gerçekleştirilir.

C++ ve bir GUI kütüphanesi kullanarak, bir eczane yönetim sistemi için interaktif bir uygulama geliştirdiğinizi düşünün. Bu sistemde, "Ilaclar" adında bir tablo var ve bu tablo, her bir ilaç için IlacID, IlacAdi, Üretici, Fiyat ve StokAdedi gibi bilgileri içeriyor. Bu senaryoda, eczane çalışanlarının ve müşterilerin ilaçlarla ilgili verilere arayüz üzerinden erişebilmeleri ve bazı işlemler yapabilmeleri isteniyor.

Görevler

İlaç Listeleme: Eczane çalışanlarının ve müşterilerin mevcut tüm ilaçları listeleyebileceği bir arayüz tasarlayın. Bu arayüz, ilaçların tüm detaylarını göstermelidir.

İlaç Ekleme: Eczane çalışanlarının yeni ilaçlar ekleyebilmesi için bir form tasarlayın. Bu form, ilaç adı, üretici, fiyat ve stok adedi gibi bilgileri almalıdır.

İlaç Güncelleme: Eczane çalışanlarının mevcut ilaçların bilgilerini güncelleyebilmesi için bir arayüz tasarlayın. Bu işlem, ilacın fiyatı veya stok adedinde değişiklik yapmayı içerebilir.

İlaç Silme: Eczane çalışanlarının stokta kalmayan veya artık satılmayacak ilaçları sistemden kaldırabilmesi için bir özellik ekleyin

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

İlaçlar Tablosu Örneği

IlacID	IlacAdi	Üretici	Fiyat	StokAdedi
1	Paracetamol	Pharma Inc.	10	50
2	Ibuprofen	BestMed	15	100
3	Amoxicillin	HealthCorp	20	30

Soru

Bir C++ uygulaması geliştiriyorsunuz ve bu uygulama, eczane çalışanlarının ve müşterilerin "Ilaclar" veri tabanı tablosu üzerinde çeşitli işlemler yapmasına olanak tanıyor. Qt, wxWidgets veya başka bir C++ GUI kütüphanesi kullanarak, aşağıdaki işlevleri yerine getiren bir arayüz tasarlayın:

Listeleme İşlemi: Kullanıcılara, sistemde kayıtlı tüm ilaçların bir listesini gösteren bir arayüz bölümü tasarlayın.

Ekleme İşlemi: Eczane çalışanlarının yeni ilaçlar ekleyebileceği bir form tasarlayın. Form, ilgili tüm ilaç bilgilerini alacak şekilde tasarlanmalıdır.

Güncelleme İşlemi: Eczane çalışanlarının ilaçların fiyatını veya stok adedini güncelleyebilmesi için bir arayüz tasarlayın. İlgili ilacı seçmek ve güncellemek için bir mekanizma içermelidir.

Silme İşlemi: Eczane çalışanlarının seçili ilaçları sistemden kaldırabileceği bir işlev ekleyin. Bu işlev, seçilen ilacı veri tabanından silmelidir.

5. ÜNİTE: VERİ TABANI İŞLEMLERİ

İlaçlar Tablosu Örneği

IlacID	IlacAdi	Üretici	Fiyat	StokAdedi
1	Paracetamol	Pharma Inc.	10	50
2	Ibuprofen	BestMed	15	100
3	Amoxicillin	HealthCorp	20	30

Arama İşlemi: Kullanıcıların ilaç adına göre arama yapabilmesini sağlayan bir arama özelliği ekleyin. Kullanıcı bir ilaç adı parçası girdiğinde, sistem bu bilgiyi içeren tüm ilaçları listelemelidir.

İlaç Bilgisi Güncelleme İşlemi: Eczane çalışanlarının bir ilacın fiyatını veya stok adedini güncelleyebilmesi, ilacın IlacIDsi ile belirlenen spesifik bir ilaç seçmeli ve yeni değerleri güncellenmeli

İlaç ve Üretici Bilgilerini Birleştirme İşlemi: Sistemde ilaçlar ve üreticileri arasında bir ilişki kurun kullanıcıların ilaçların detaylı bilgilerini ve bu ilaçları üreten şirketlerin bilgilerini bir arada görebilmesini sağlayın.