Nesne Yönelimli Programlama

Nesne Yönelimli Programlama Ders İçeriği

Hafta 1: Giriş ve C++ Temelleri

- Programlamaya Giriş
- C++'ın Genel Yapısı ve Tarihçesi
- C++ Geliştirme Ortamının (IDE) Kurulumu
- İlk Program: `Hello World`
- Temel Veri Tipleri (int, float, char, bool)
- Operatörler (Aritmetik, İlişkisel, Mantıksal)
- Girdi/Çıktı İşlemleri (`cin` , `cout`)

Hafta 2: Kontrol Yapıları ve Döngüler

- Koşul İfadeleri (`if` , `else if` , `else`)
- `switch` Yapısı
- Döngüler: `for` , `while` , `do-while`
- Break, Continue ve Return Kullanımı
- Nested (İç İçe) Döngüler

Hafta 3: Fonksiyonlar

- Fonksiyon Tanımı ve Kullanımı
- Parametreler ve Geri Dönüş Değerleri
- Fonksiyon Prototipleri
- Overloading (Aşırı Yükleme)
- Recursive (Özyinelemeli) Fonksiyonlar

Hafta 4: Diziler ve Karakter Dizileri

- Tek Boyutlu Diziler
- Çok Boyutlu Diziler (Matrisler)
- Dizilerde Bellek Yönetimi
- Karakter Dizileri (C-Stringler)
- Dizilerde Sıralama ve Arama Algoritmaları

Hafta 5: Pointerlar (İşaretçiler)

- Pointer Kavramı ve Kullanımı
- Pointer Aritmetiği
- Diziler ve Pointerlar
- Fonksiyonlarda Pointer Kullanımı
- Bellek Yönetimi: `new` ve `delete` Anahtar Kelimeleri

Hafta 6: Yapılar ve Birlikler

- Yapılar (Struct) ile Veri Gruplama
- Yapılar İçinde Pointerlar
- Birlikler (Union) ile Bellek Paylaşımı
- Bit Alanları (Bit Fields)

Hafta 7: Nesne Yönelimli Programlamaya Giriş

- Nesne Yönelimli Programlamanın Temelleri
- Sınıflar ve Nesneler
- Sınıf Tanımı ve Nesne Oluşturma
- Erişim Belirleyicileri: `public`, `private`, `protected`
- Yapıcı (Constructor) ve Yıkıcı (Destructor) Fonksiyonlar

Hafta 8: Veri Kapsülleme ve Soyutlama

- Veri Kapsülleme Kavramı
- Getter ve Setter Fonksiyonları
- Veri Gizliliği
- Soyutlama Nedir? Nesnelerle İşlemler
- Uygulama: Basit Bir Sınıf Tasarımı

Hafta 9: Kalıtım (Inheritance)

- Kalıtımın Temelleri
- Türemiş ve Taban Sınıflar
- Erişim Belirleyicileri ile Kalıtım
- Çok Biçimlilik (Polymorphism) ve Virtual Fonksiyonlar
- Override ve Overload Farkları

Hafta 10: Çok Biçimlilik (Polymorphism) ve Sanal Fonksiyonlar

- Polymorphism Kavramı
- Sanal Fonksiyonlar (Virtual Functions)
- Saf Sanal Fonksiyonlar ve Soyut Sınıflar (Abstract Classes)
- Dinamik Bağlama (Dynamic Binding)
- Virtual Destructor Kullanımı

Hafta 11: Operatör Aşırı Yükleme (Operator Overloading)

- Operatör Aşırı Yüklemenin Amacı
- Temel Operatörlerin Aşırı Yüklenmesi
- İkili (Binary) ve Tekli (Unary) Operatör Aşırı Yükleme
- `<<` ve `>>` Operatörlerinin Aşırı Yüklenmesi
- Uygulama: Bir Matris Sınıfında Operatör Aşırı Yükleme

Hafta 12: Şablonlar (Templates)

- Fonksiyon Şablonları
- Sınıf Şablonları
- Şablon Parametreleri
- Generic Programlamanın Avantajları
- Şablonlarda Sınırlamalar ve İstisnalar

Hafta 13: İstisna Yönetimi (Exception Handling)

- Hata Yönetimi
- `try`, `catch` ve `throw` Blokları
- Standart İstisnalar (Standard Exceptions)
- Özel İstisnaların Tanımlanması
- İstisna Güvenli Kod Yazma Teknikleri

Hafta 14: Dosya İşlemleri ve Proje Sunumu

- Dosya İşlemlerine Giriş: `ifstream`, `ofstream`, `fstream`
- Metin ve İkili Dosya Okuma/Yazma
- Dosya İşaretleyicileri ve Pozisyonlar

C++ Temel Veri Tipleri, Operatörler ve Girdi/Çıktı İşlemleri

Giriş

- C++ programlamada temel kavramlara giriş
- Programlama dillerinin temeli ve C++'ın önemi
- Temel veri tipleri, operatörler ve girdi/çıktı işlemlerinin programlamadaki rolü

Temel Veri Tipleri

- C++'ta kullanılan temel veri tipleri ve bunların bellek üzerindeki karşılıkları
- Programların işleyişinde veri tiplerinin rolü

`int` Veri Tipi

- Tam sayıları temsil eder. Negatif ve pozitif tam sayıları içerebilir.
- Bellek boyutu: genellikle 4 byte (platforma göre değişebilir)
- Örnekler: 1, -42, 1000
- **Kullanım Alanları:** Sayma, döngülerde sayıcı, sayısal hesaplamalar.

`float` Veri Tipi

- Kesirli sayıları temsil eder. Genellikle 6-7 basamak hassasiyet sunar.
- Bellek boyutu: genellikle 4 byte
- Örnekler: 3.14, -0.001, 2.0
- **Kullanım Alanları:** Bilimsel hesaplamalar, fiziksel ölçümler.

`double` Veri Tipi

- Daha hassas kesirli sayıları temsil eder. Genellikle 15-16 basamak hassasiyet sunar.
- Bellek boyutu: genellikle 8 byte
- Örnekler: 3.14159, -0.0001, 100.123456
- **Kullanım Alanları:** Finansal hesaplamalar, hassas bilimsel hesaplamalar.

`char` Veri Tipi

- Tek bir karakteri temsil eder. ASCII karakter setini kullanır.
- Bellek boyutu: genellikle 1 byte
- Örnekler: 'a', 'Z', '5'
- **Kullanım Alanları:** Karakter dizileri, kullanıcı girdisi olarak karakterler.

`bool` Veri Tipi

- Mantıksal değerleri temsil eder. `true` veya `false` değerlerini alır.
- Bellek boyutu: genellikle 1 byte
- **Kullanım Alanları:** Koşul ifadelerinde, durum kontrollerinde.

Veri Tiplerinin Kullanımı

- Veri tiplerinin belirlenmesi ve nasıl kullanıldıkları hakkında bilgi.
- C++'ta veri tiplerinin kullanımı ile ilgili örnek kod:

```
int a = 10; // Tam sayı
float b = 5.5; // Kesirli sayı
char c = 'A'; // Karakter
bool isActive = true; // Mantıksal değer
```

Operatörler

- C++'ta operatörlerin tanımı ve kullanım alanları.
- Operatörlerin programlama mantığındaki önemi ve farklı türleri.

Aritmetik Operatörler

- Aritmetik işlemler için kullanılan operatörler.
- Temel Aritmetik Operatörler:
  - `+` : Toplama
  - `-` : Çıkarma
  - `*` : Çarpma
  - `/` : Bölme
  - `%` : Modül (kalan alma)

Aritmetik Operatör Örnekleri

- Aritmetik operatörlerin kullanımına dair örnekler:

```
int x = 10;
int y = 3;
int sum = x + y; // Toplama: 13
int diff = x - y; // Çıkarma: 7
int product = x \* y; // Çarpma: 30
int quotient = x / y; // Bölme: 3
int remainder = x % y; // Modül: 1
```

İlişkisel Operatörler

- İlişkisel operatörlerin tanımı ve kullanımı.
- Temel İlişkisel Operatörler:
  - `==` : Eşitlik
  - `!=` : Eşitsizlik
  - `>` : Büyüktür
  - `<` : Küçüktür
  - `>=` : Büyük veya eşit
  - `<=` : Küçük veya eşit

İlişkisel Operatör Örnekleri

- İlişkisel operatörlerin kullanımına dair örnekler:

```
if (x == y) {
cout << "x eşit y"; // Eşitlik kontrolü
}
if (x > y) {
cout << "x, y'den büyüktür"; // Büyüklük kontrolü
}
```

Girdi/Çıktı İşlemleri

- C++'ta girdi/çıktı işlemlerinin önemi.
- `cin` ve `cout` kullanımı ile veri alışverişi sağlama.

`cout` Kullanımı

- `cout` ile ekrana veri yazdırma:
- Örnek:

```
1   cout << "Merhaba, dünya!" << endl; // "Merhaba, dünya!" yazdırır
```

`cin` Kullanımı

- `cin` ile kullanıcıdan veri alma:
- Örnek:

```
int number;
cout << "Bir sayı girin: ";
cin >> number; // Kullanıcıdan bir sayı alır
```

Girdi/Çıktı İşlemlerinde Formattırma

- Veri formatlama ve kontrolü.
- Örnekler:

```
#include <iomanip> // Format ayarları için
cout << fixed << setprecision(2) << 3.14159; // 3.14 yazdırır
```

Hatalı Girişleri Yönetme

- Kullanıcının hatalı girişlerini kontrol etme.
- Örnek kod ile açıklama:

```
int number;
while (true) {
cout << "Bir sayı girin: ";
if (cin >> number) break; // Geçerli giriş
cout << "Hatalı giriş, lütfen tekrar deneyin." << endl;
cin.clear(); // Hata durumunu sıfırla
cin.ignore(numeric\_limits\<streamsize>::max(), '\n'); // Girdiyi temizle }
```

Girdi/Çıktı İşlemleri Özet

- Girdi/çıktı işlemlerinin önemi ve genel kullanım.
- Kullanıcı etkileşiminin sağlanması ve programın esnekliği.

Uygulama Örneği

- Temel veri tipleri, operatörler ve girdi/çıktı işlemlerinin birleşimi.
- Örnek program:

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

int main() {
int a, b;
cout << "Birinci sayıyı girin: ";
cin >> a; // Kullanıcıdan ilk sayıyı al
cout << "İkinci sayıyı girin: ";
cin >> b; // Kullanıcıdan ikinci sayıyı al
cout << "Toplam: " << a + b << endl; // Toplamı hesapla ve yazdır
return 0;
}
```

C++'ta `using namespace` İfadesi

- İsim alanları, program içinde kullanılan isimlerin gruplandığı yapılardır
- `using namespace` ifadesi, kod yazımını kolaylaştırmak amacıyla kullanılır

İsim Alanı (Namespace) Nedir?

- İsim alanları, isimlerin (fonksiyonlar, değişkenler, sınıflar vb.) çakışmasını önlemek için tasarlanmıştır
- Farklı isim alanlarında aynı isimde öğeler bulunabilir
- Örnek:

```
namespace Math {
    int add(int a, int b) {
        return a + b;
    }
}

namespace Geometry {
    int add(int a, int b) {
        return a + b;
```

Neden İsim Alanları Kullanılır?

- **İsim Çakışmalarını Önleme:** İsim alanları, aynı isimdeki öğelerin çakışmasını önler
- **Kod Düzenleme:** Daha iyi bir organizasyon sağlar, kodun okunabilirliğini artırır
- **Büyük Projelerde Yönetim:** Özellikle büyük projelerde, farklı kütüphanelerden gelen isimlerin karışmaması için faydalıdır

`using namespace` İfadesinin Amacı

- Belirli bir isim alanındaki öğeleri doğrudan kullanmayı sağlar
- Kodun okunabilirliğini ve yazımını kolaylaştırır
- **Örnek:**

```cpp
#include <iostream>
using namespace std; // std isim alanındaki öğeleri doğrudan kullanma

int main() {
    cout << "Merhaba, dünya!" << endl; // std::cout yerine sadece cout
    return 0;
}
```

```cpp
#include <iostream>
int main() {
    std::cout << "Merhaba, dünya!" << endl;
    return 0;
}
```

`using namespace` Kullanımında Dikkat Edilmesi Gerekenler

- **İsim Çakışmaları:** Aynı isimde öğelerin çakışma riski vardır
- **Kodun Anlaşılabilirliği:** Çok fazla `using namespace` kullanımı, kodu karışık hale getirebilir

```
namespace First {
    void show() {
        cout << "First namespace" << endl; }
}
namespace Second {
    void show() {
        cout << "Second namespace" << endl; }
}
using namespace First;
using namespace Second; // Hata: show() çakışması

int main() {
    show(); // Hangi show() fonksiyonunun çağrılacağı belirsiz
    return 0;
}
```

İsim Çakışmalarını Önleme

- Belirli isim alanlarından yalnızca gerekli olan öğeleri almak için `using` ifadesini kullanmak:

```
using First::show; // Yalnızca First isim alanından
            // show() fonksiyonunu kullan
```

- Alternatif olarak, tam isimle kullanmak da bir seçenektir:

```
int main() {
    First::show(); // First isim alanındaki show() fonksiyonunu çağır
    Second::show(); // Second isim alanındaki show() fonksiyonunu çağır
    return 0;
}
```

`using` ile Aliasing

- `using` ifadesi ile yeni isimler oluşturma:

```
namespace MyNamespace {
    using Integer = int; // Integer artık int türünü temsil eder
}

int main() {
    MyNamespace::Integer x = 5; // x bir int türünde
    cout << "Değer: " << x << endl;
    return 0;
}
```

Alıştırma 1: Basit Hesaplama

Kullanıcıdan iki tam sayı girmesini isteyin. Bu sayıları toplayın, çıkarın, çarpın ve bölün, ardından sonuçları ekrana yazdırın.

Beklenen Çıktı:

```
Birinci sayıyı girin: 10
İkinci sayıyı girin: 5
Toplam: 15
Fark: 5
Çarpım: 50
Bölüm: 2
```

Alıştırma 1

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

int main() {
    int sayi1, sayi2;

    // Kullanıcıdan iki tam sayı girmesini isteyin
    cout << "Birinci sayıyı girin: ";
    cin >> sayi1;
    cout << "İkinci sayıyı girin: ";
    cin >> sayi2;

    // Toplama, çıkarma, çarpma ve bölme işlemleri
    int toplam = sayi1 + sayi2;
    int fark = sayi1 - sayi2;
    int carpim = sayi1 * sayi2;
    float bolum = static_cast<float>(sayi1) / sayi2;
    // Bölme işlemi için float dönüşümü
```

```
    // Sonuçları ekrana yazdırma
    cout << "Toplam: " << toplam << endl;
    cout << "Fark: " << fark << endl;
    cout << "Çarpım: " << carpim << endl;

    if (sayi2 != 0) { // Bölme işlemi için 0 kontrolü
        cout << "Bölüm: " << bolum << endl;
    } else {
        cout << "Bölme hatası: İkinci sayı sıfır olamaz!" << endl;
    }

    return 0;
}
```

Alıştırma 2: Alan Hesaplama

Kullanıcıdan bir dairenin yarıçapını isteyin. Dairenin alanını hesaplayın (Alan = π * r^2) ve sonucu ekrana yazdırın. π için 3.14 değerini kullanın.

Beklenen Çıktı:

```
Yarıçapı girin: 7
Dairenin alanı: 153.86
```

Alıştırma 2

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

int main() {
    const double PI = 3.14; // Pi sayısı
    int yaricap;
    double alan
    // Kullanıcıdan yarıçap değerini al
    cout << "Yarıçapı girin: ";
    cin >> yaricap;
    // Alan hesaplama formülü: π * r²
    alan = PI * yaricap * yaricap;
    // Sonucu ekrana yazdır
    cout << "Dairenin alanı: " << alan << endl;

    return 0;
}
```

Alıştırma 3: Karakter Girişi

Kullanıcıdan bir karakter girmesini isteyin ve bu karakterin ASCII değerini hesaplayarak ekrana yazdırın.

Beklenen Çıktı:

```
Bir karakter girin: A
Karakter: A,
ASCII Değeri: 65
```

Alıştırma 3

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

int main() {
    char karakter;

    // Kullanıcıdan bir karakter girmesini iste
    cout << "Bir karakter girin: ";
    cin >> karakter;

    // Karakterin ASCII değerini hesapla ve yazdır
    cout << "Karakter: " << karakter << ", ASCII Değeri: " << int(karakter) << endl;

    return 0;
}
```

Alıştırma 4: Mantıksal Kontrol

Kullanıcıdan bir sayı girmesini isteyin. Girilen sayının pozitif, negatif veya sıfır olduğunu kontrol edin ve uygun mesajı ekrana yazdırın.

Beklenen Çıktı:

```
Bir sayı girin: -3
Sayı negatif.
```

Alıştırma 4

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

int main() {
    int sayi;

    // Kullanıcıdan bir sayı girmesini iste
    cout << "Bir sayı girin: ";
    cin >> sayi;

    // Sayının pozitif, negatif veya sıfır olup olmadığını kontrol et
    if (sayi > 0) {
        cout << "Sayı pozitif." << endl;
    } else if (sayi < 0) {
        cout << "Sayı negatif." << endl;
    } else {
        cout << "Sayı sıfır." << endl;
    }

    return 0;
}
```

Alıştırma 5: Not Hesaplama

Kullanıcıdan bir öğrencinin notunu (0-100 arası) girmesini isteyin ve notun hangi harf notuna karşılık geldiğini belirleyin. Not aralıkları şu şekilde olsun:

- 90-100: A
- 80-89: B
- 70-79: C
- 60-69: D
- 0-59: F

Beklenen Çıktı:

```
Notunuzu girin: 85
Harf Notu: B
```