Nesne Yönelimli Programlamaya Giriş

Bu hafta

- iç içe sınıflar
- inline
- static üyeler

İç İçe Sınıflar (Nested Classes)

İç İçe Sınıf Nedir?

- **İç içe sınıf**, bir sınıfın içinde tanımlanan başka bir sınıftır.
- Dış sınıf, iç sınıfın **kapsamını** belirler.
- İç sınıflar genellikle dış sınıfın verilerine ve fonksiyonlarına erişmek için kullanılır.

İç İçe Sınıfların Özellikleri

1. **Kapsam**:
   - İç sınıflar, yalnızca dış sınıfın bir parçası olarak tanımlanır.
2. **Erişim**:
   - İç sınıf, dış sınıfın **private**, **protected**, ve **public** üyelerine doğrudan erişebilir.
   - Ancak, dış sınıfın iç sınıfa doğrudan erişimi yoktur; iç sınıfın bir nesnesi üzerinden erişim yapılır.
3. **Kullanım Alanları**:
   - Kompleks yapıları daha iyi organize etmek.
   - Dış sınıfın verilerine özel erişim sağlamak.

İç İçe Sınıfların Kullanımı

Örnek:

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

class DisSinif {
private:
    int veri;

public:
    DisSinif(int deger) : veri(deger) {}

    // İç sınıf
    class IcSinif {
    public:
        void yazdir(const DisSinif& disObj) {
            // Dış sınıfın private verisine erişim
            cout << "Dış sınıf verisi: " << disObj.veri << endl;
        } };
};
int main() {
    DisSinif disObj(42);              // Dış sınıf nesnesi
    DisSinif::IcSinif icObj;          // İç sınıf nesnesi
    icObj.yazdir(disObj);             // İç sınıf üzerinden veri erişimi
```

```cpp
#include <iostream>
using namespace std;
class Araba {
private:
    string marka;
public:
    Araba(string m) : marka(m) {}
    // İç sınıf
    class Motor {
    public:
        void calistir() {
            cout << "Motor çalışıyor." << endl;
        }
        void arabaMarkasi(const Araba& araba) {
            // Dış sınıfın private üyesine erişim
            cout << "Arabanın markası: " << araba.marka << endl;
        }
    };
};
int main() {
    Araba araba("Toyota");            // Dış sınıf nesnesi
    Araba::Motor motor;               // İç sınıf nesnesi
    motor.calistir();                 // Motor çalıştırma
    motor.arabaMarkasi(araba);        // Araba markasına erişim
    return 0;
}
```

İç İçe Sınıfların Avantajları

1. **Kapsülleme**:
   - İç içe sınıflar, dış sınıfa özgü işlevsellikleri bir arada tutarak kodun kapsülleme prensibini güçlendirir.
2. **Daha İyi Organizasyon**:
   - Karmaşık yapıları daha okunabilir ve yönetilebilir hale getirir.
3. **Veri Güvenliği**:
   - İç sınıf, dış sınıfın verilerine doğrudan erişebilirken, bu erişim kontrollü bir şekilde sağlanır.

İç İçe Sınıfların Dezavantajları

1. **Bağımlılık**:
   - İç sınıf, dış sınıfa bağımlı olduğu için, bağımsız olarak kullanılamaz.
2. **Kod Karmaşıklığı**:
   - Fazla sayıda iç içe sınıf kullanımı, kodun anlaşılmasını zorlaştırabilir.

Özetle

- **İç içe sınıflar**, dış sınıfa özel işlevsellikleri bir sınıf içinde organize etmenin bir yoludur.
- **Dış sınıfın verilerine erişim** sağlarken, dış sınıfın iç sınıfa erişimi nesne üzerinden gerçekleştirilir.
- Dikkatli kullanıldığında, kodun daha iyi organize edilmesine ve veri güvenliğinin artırılmasına yardımcı olur.

`inline` Fonksiyonlar

`inline` Nedir?

- `inline` anahtar kelimesi, bir fonksiyonun çağrıldığında programın akışını kesmek yerine, kodun çağrıldığı yere yerleştirilmesini önerir.
- Bu, **çağrı sırasında oluşan ek yükü** azaltabilir ve **yürütme hızını artırabilir**.
- Derleyici, `inline` önerisini dikkate alabilir veya almayabilir.

`inline` Fonksiyonların Kullanımı

Temel Örnek:

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

// inline fonksiyon
inline int kare(int sayi) {
    return sayi * sayi;
}

int main() {
    int x = 5;
    cout << "Kare: " << kare(x) << endl;

    return 0;
}
```

`inline` Fonksiyonların Özellikleri

1. **Kod Tekrarı**:
   - `inline`, çağrıyı doğrudan kodla değiştirir ve böylece fonksiyon çağrısı sırasında oluşan zaman maliyetini azaltır.
2. **Kısa Fonksiyonlar için Uygun**:
   - Genellikle **küçük ve sık kullanılan** fonksiyonlarda kullanılır.
3. **Derleyiciye Bağlıdır**:
   - `inline`, yalnızca bir öneridir. Derleyici, fonksiyonu `inline` olarak işlemeyebilir.

`inline` ve Geleneksel Fonksiyonlar Arasındaki Farklar

`inline` **Fonksiyonlar**	**Geleneksel Fonksiyonlar**
Kod, çağrının yapıldığı yere yerleştirilir.	Fonksiyon çağrısı sırasında adrese bir sıçrama yapılır.
Daha hızlı çalışabilir (özellikle küçük fonksiyonlarda).	Fonksiyon çağrısında zaman maliyeti oluşur.
Kod boyutunu artırabilir.	Kod boyutu sabit kalır.

`inline` Fonksiyonlar ve Makrolar Arasındaki Farklar

`inline` **Fonksiyonlar**	**Makrolar**
Derleyici tarafından kontrol edilir.	Preprocessor (ön işlemci) tarafından işlenir.
Tür kontrolü yapılır.	Tür kontrolü yapılmaz.
Hata ayıklama (debugging) kolaydır.	Hata ayıklama zordur.

Örnek: `inline` ve Makro Karşılaştırması

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

#define KARE(x) ((x) * (x)) // Makro tanımı
inline int kare(int x) {
    return x * x;
}

int main() {
    cout << "Makro: " << KARE(5) << endl; // Makro kullanımı
    cout << "Inline: " << kare(5) << endl; // Inline fonksiyon kullanımı

    return 0;
}
```

`inline` Kullanımına Uygun Durumlar

1. **Kısa Fonksiyonlar**:
   - Tek satırlık veya birkaç satırlık basit fonksiyonlar.
2. **Sık Kullanılan Fonksiyonlar**:
   - Çok sayıda çağrılan fonksiyonlar.
3. **Performans İyileştirme**:
   - Çağrı maliyetini düşürmek için kullanılır.

`inline` Kullanımına Uygun Olmayan Durumlar

1. **Uzun Fonksiyonlar**:
   - Büyük fonksiyonlarda kod tekrarı program boyutunu artırabilir.
2. **Döngüler veya Karmaşık Yapılar**:
   - `inline` fonksiyonlar, içlerinde döngüler veya karmaşık kod yapıları içeriyorsa verimsiz olabilir.

Özetle

- `inline`, fonksiyonların çağrıldığı yerde kodun yerleştirilmesini önerir.
- Küçük ve sık kullanılan fonksiyonlarda performans artırıcı bir etkisi olabilir.
- Ancak, derleyicinin `inline` önerisini kabul etme zorunluluğu yoktur.

`this` Pointer

`this` Pointer Nedir?

- `this` pointer, C++ dilinde her sınıfın otomatik olarak sahip olduğu özel bir işaretçidir.
- Bir sınıfın üyesi olan fonksiyonlar içinde kullanılabilir.
- `this`, fonksiyonun çağrıldığı **mevcut nesneyi** işaret eder.
- Bu işaretçi, sınıf üyeleri ile aynı ada sahip yerel değişkenleri ayırt etmek için kullanılır.

`this` Pointer'ın Özellikleri

1. **Her Nesne için Benzersizdir**:
   - Her nesne, kendi `this` pointer'ına sahiptir ve bu pointer o nesneyi işaret eder.
2. **Mevcut Nesneye Erişim Sağlar**:
   - Sınıf üyesi fonksiyonlar içinde, çağrıyı yapan nesneye erişmek için kullanılır.
3. **Kapsülleme için Kullanılır**:
   - Üye değişkenlerle aynı ada sahip yerel değişkenleri ayırt etmek için kullanılır.

`this` Pointer Kullanımı

Örnek 1: Yerel Değişkenler ile Çakışma

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

class Kisi {
private:
    string isim;
public:
    // Parametre ismi ile üye değişken çakışması
    void setIsim(string isim) {
        this->isim = isim; // `this` ile sınıfın üyesine erişim
    }
    void yazdir() {
        cout << "İsim: " << isim << endl;
    }
};

int main() {
    Kisi kisi;
    kisi.setIsim("Ali");
    kisi.yazdir();

    return 0;
}
```

Örnek 2: Zincirleme (Chaining) Fonksiyon Çağrıları

- `this` pointer, aynı nesneye referans döndürmek için kullanılabilir. Bu sayede zincirleme fonksiyon çağrıları yapılabilir.

```cpp
#include <iostream>
using namespace std;
class Kisi {
private:
    string isim;
    int yas;
public:
    // Zincirleme çağrılar için `this` kullanımı
    Kisi& setIsim(string isim) {
        this->isim = isim;
        return *this;
    }

    Kisi& setYas(int yas) {
        this->yas = yas;
        return *this;
    }

    void yazdir() {
        cout << "İsim: " << isim << ", Yaş: " << yas << endl;
    }
};
int main() {
    Kisi kisi;
    kisi.setIsim("Ayşe").setYas(25).yazdir();

```

Örnek 3: Nesne Karşılaştırması

- `this` pointer, bir nesneyi başka bir nesne ile karşılaştırmak için kullanılabilir.

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

class Sayac {
private:
    int deger;
public:
    Sayac(int deger) : deger(deger) {}

    bool esittir(const Sayac& diger) {
        return this->deger == diger.deger; // `this` mevcut nesneyi işaret eder
    }
};

int main() {
    Sayac sayac1(10);
    Sayac sayac2(10);
    Sayac sayac3(20);

    cout << "sayac1 ve sayac2 eşit mi? " << (sayac1.esittir(sayac2) ? "Evet" : "Hayır") << endl;
    cout << "sayac1 ve sayac3 eşit mi? " << (sayac1.esittir(sayac3) ? "Evet" : "Hayır") << endl;

```

`this` Pointer'ın Avantajları

1. **Üye Değişken ve Yerel Değişken Çakışmalarını Çözme**:
   - `this` pointer, çakışmaları çözmek için net bir yol sağlar.
2. **Zincirleme Fonksiyon Çağrıları**:
   - Aynı nesne üzerinde birden fazla işlemi tek bir satırda yapmayı mümkün kılar.
3. **Mevcut Nesneye Doğrudan Erişim**:
   - Çağrıyı yapan nesneye referans sağlar.

Özetle

- `this` pointer, bir sınıfın mevcut nesnesine referans verir.
- Çakışmaları önlemek, zincirleme çağrılar yapmak ve nesneler üzerinde işlem yapmak için kullanılır.
- Kodun daha okunabilir ve güvenli hale gelmesine yardımcı olur.

Statik Üyeler (Static Members)

Statik Değişkenler (Static Variables)

- Bir sınıfa ait **statik değişkenler** sınıfın **tüm nesneleri tarafından paylaşılır**.
- Statik değişkenler sınıf düzeyinde tanımlanır ve **tek bir kopya** oluşturulur.
- Anahtar kelime: `static`.