

# Arv (Inheritance) i C++

Genbrug af kode med  
objektorienteret arv

10. april 2025

# Først et citat

In many ways, programming in C++ is like driving without a road.

Sure, you can go anywhere you want, but there are no lines or traffic lights to keep you from injuring yourself. C++, like the C language, has a hands-off approach toward its programmers.

The language assumes that you know what you're doing. It allows you to do things that are likely to cause problems because C++ is incredibly flexible and sacrifices safety in favor of performance.

Memory allocation and management is a particularly error-prone area of C++ programming.

# Kilde

*Professional C++*

(3rd Edition)

Marc Gregoire

2014





# Hvad er arv?

- Arv er et grundprincip i objektorienteret programmering.
- En klasse kan 'arve' egenskaber og funktioner fra en anden.
- Bruges til genbrug og strukturering af kode.

# Eksempel fra virkeligheden

- Klasse: Dyr
- Klasse: Hund
- En Hund er et Dyr, men har egne funktioner som fx at gø.
- Dyr: superklasse eller baseklasse.
- Hund: subklasse eller afledt klasse.
- Validering af arv: en hund er et dyr.
- Subklasse-objekt **er** et superklasse-objekt.

# Fordele ved arv

-  Genbrug af kode.
-  Organisering af komplekse systemer.
-  Mulighed for polymorfi.
-  Risiko for kompleksitet ved uhensigtsmæssig anvendelse.

# Arv i C++ – Syntax

```
class Dyr {  
public:  
    void spis() {  
        cout << "Dyr spiser";  
    }  
};
```

```
class Hund : public Dyr {  
public:  
    void gø() {  
        cout << "Vuf!";  
    }  
};
```

# Brug af arv i praksis

```
int main() {  
    Hund minHund;  
    minHund.spis(); // arvet  
    minHund.gø();  // egen metode  
}
```



# Adgangsniveauer i arv

- public: Alle har adgang
- protected: Tilgængelig i underklasser
- private: Skjult for underklasser

# Overskrivning med override

```
class Dyr {  
public:  
    virtual void lyd() {  
        std::cout << "Dyr laver lyd\n";  
    }  
};  
  
class Kat : public Dyr {  
public:  
    void lyd() override {  
        std::cout << "Miauw\n";  
    }  
};
```

# Hvad gør 'virtual' og 'override'?

- 'virtual' i basisklassen gør det muligt at kalde den rette funktion via en pointer/reference
- 'override' i underklassen sikrer at vi faktisk overskriver noget
- Dette kaldes dynamisk binding eller polymorfi

# Eksempel på polymorfi

```
Dyr* d = new Kat();  
d->lyd(); // kalder Kat::lyd()  
på trods af typen Dyr*
```

# Multipel arv/Multiple inheritance

- C++ understøtter, at en klasse kan arve mere en én anden klasse.
- Det gør Python også.
- Men Java og C# gør *ikke*.
- Multipel arv bør anvendes med den største forsigtighed.
- Rødderne til multipel arv skal findes i tidlige forsøg på kunstig intelligens.

# Opsummering

- Arv tillader genbrug og struktur.
- Husk at bruge `protected`.
- Brug `virtual/override` for fleksibilitet.
- Pas på overdreven brug af arv.