

Del 2 – Klasser, medlemmar och arv

Ämnesområden denna föreläsning:

- Klasser, åtkomst
- Medlemmar, medlemsfunktioner, inline
- Konstruktörer
- Destruktörer
- `this`-pekaren
- Arv, åtkomst
- Multipelt arv, virtuell basklass
- Konstanta funktioner och data, `mutable`

Slide 1

Klasser

Klasser används för att

- strukturera koden
- kapsla in data
- knyta metoder till data
- skapa återanvändbara komponenter

Slide 2

Medlemmar och åtkomst

Slide 3

- Alla kommer åt `public`
- Endast subclasser kommer åt `protected`
- Endast klassen själv kommer åt `private`
- Medlemsfunktioner som definieras i klassdeklarationen är implicit deklarerade som `inline`

Slide 4

```
/* myfile.h -- headerfil med deklARATIONER */
class A
{
    int i;          // i är private
public:           // foo är inline
    int foo(char) { return 7; }
    int bar();     // bar, baz är inline om
    char baz();   // definitionen anger det
protected:
    double d;
};
inline int A::bar() { return 2; } // inline i .h-fil

/* myfile.cpp -- källkodsfil med definitioner */
char A::baz() { return 'x'; } // ej inline i .cpp-fil
```

Statiska medlemmar

Statiska medlemmar är gemensamma för alla instanser av samma klass. De kan t.ex. användas till att räkna instanser av klassen.

Slide 5

```
class A { // i headerfil myfile.h
    static const int fixed_ = 4711; // statisk medlem
    static int count_;           // statisk medlem
    int array[fixed_];          // vanlig medlem
public:
    static int count() {return count_;} // statisk funktion
};
// i implementationsfil myfile.cpp
const int A::fixed_; // statiska medl. måste definieras
int A::count_ = 0;   // statisk medlemsinitiering
...
int i = A::count();
```

Konstruktörer

Slide 6

- När ett objekt instansieras anropas dess **konstruktor**
- Konstruktörer saknar returtyp
- En **implicit konstruktor** tillhandahålls av kompilatorn “vid behov”
- Konstruktörer kan överlagras
- Använd **explicit** då konstruktorn har ett argument av enkel typ
- Konstruktor som är **protected** ger klass som bara kan skapas av subklasser
- Konstruktor som är **private** ger möjlighet att förhindra användandet av t.ex. defaultkonstruktorn

Slide 7

```
// date.h -- headerfil
class Date
{
    Date();                // privat defaultkonstr.
public:
    Date(const Date &);    // kopieringskonstruktor
    Date(int y, int m, int d); // ÅÅMMDD
    explicit Date(int c);   // dagar från 1900-01-01
    ...
    int year, month, date;
};
// date.cpp -- källkodsfil
Date::Date(const Date &d) { /* kopiering */ }
Date::Date(int y, int m, int d)
    : year(y), month(m), date(d) // initiering av medl.
{ }
```

Slide 8

```
class A
{
public:
    A(int i) : number(i) {} // konstruktor
    int number;
};

class B : public A
{
public:
    B(double d, int i)
        : A(i), value(d) {} // initiera basen
    double value;
};
```

Destruktorer

Slide 9

- När ett objekt förstörs (t.ex. med `delete` eller då det hamnar utanför räckvidd) anropas dess destruktör
- Destruktorn i en basclass bör vara deklarerad `virtual`

```

class String { // ligger i string.h -- headerfil
public:
    String() : str(0) {}           // defaultkonstruktör
    String(char *s);              // kopierar s
    ~String();                    // destruktör
protected:
    char *str;
};
// string.cpp -- källkodsfil
String::String(char *s) : str(0)
{
    str = new char[strlen(s) + 1]; // glöm inte '\0'
    strcpy(str, s);
}
String::~String() { delete [] str; }

```

Slide 10

this-pekaren

I en medlemsfunktion kan man komma åt objektet genom att använda den fördefinierade pekaren `this`. `this` går inte att tilldela.

Slide 11

```
class MyInt
{
    int i;
public:
    MyInt &increase(int);
};

inline MyInt &MyInt::increase(int j)
{
    i += j;
    return *this; // ger referens till objektet självt
}
```

Arv och aggregation

För att återanvända befintliga objekt finns olika möjligheter:

- Aggregat (innehållande), “har en”
- Arv, “är en”
- Olika typer av arv är `public`, `protected` och `private`

Slide 12

```
class Animal
{
    String species; // Animal innehåller (har en) sträng
};
class Bear : public Animal // Bear ärver (är en) Animal
{
    ...
};
```

Slide 13

```

class Vehicle
{
public:
    Vehicle(int w) : weight(w) {}
    int weight;
};
class Car : public Vehicle
{
public:
    Car(int p, int w) : Vehicle(w), persons(p) {}
    int persons;
};
...
Car volvo(5, 1700);
int i = volvo.weight;

```

Åtkomst vid arv

Klasser som är till hjälp för implementationen bör ärvas **protected** eller **private**. Man vill inte ha implementationsdetaljer i sitt gränssnitt.

Slide 14

		Åtkomst på data/funktion		
		public	protected	private
Typ av arv	public	public	protected	private
	protected	protected	protected	private
	private	private	private	private

Multipelt arv

C++ stöder multipelt arv av godtyckligt antal klasser.

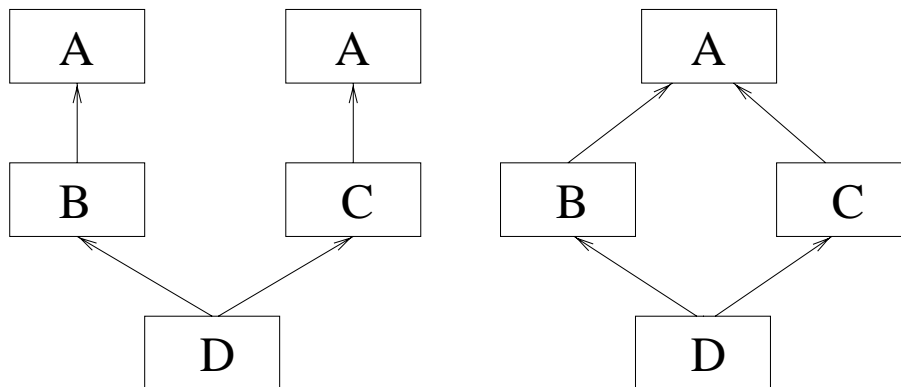
Slide 15

```
class Boat
{
public:
    Boat(bool s) : has_sail(s) {}
    bool has_sail;
};
class Amphibian : public Car, public Boat
{
public:
    Amphibian(int pers, int wt, bool sail = false)
        : Car(pers, wt), Boat(sail) {}
};
```

Virtuell basklass

För att använda gemensamma data i en delad basklass använder man *virtuella* basklasser genom nyckelordet *virtual*.

Slide 16



```

class A { public: int i; };
class B : public A {};
class C : public A {};
class D : virtual public B, virtual public C {};
class E : public B, public C {};

```

Slide 17

```

D d;
d.A::i = 7;      // alla ändrar samma data
d.B::i = 8;
d.C::i = 9;

E e;
e.A::i = 7;      // fel: tvetydigt
e.B::i = 8;      // ok: ändrar i
e.C::i = 9;      // ändrar inte samma som ovan

```

Konstanta funktioner och variabler

Deklarationer med `const` ger skrivskydd.

Slide 18

```

class A
{
    const int ci;           // ci är skrivskyddad (ssk)
    const int *p_ci;       // pekare till ssk int
    int *const cp_i;       // ssk pekare till int
    const int *const cp_ci; // ssk pekare till ssk int

    int get() const;       // objektet är ssk i get
    int i;
};

// källkodsfil
int A::get() const { return i; } // ok: förändrar inte
int A::get() const { return i = 2; } // fel: förändrar obj.

```

mutable – ändrar i objekt som är `const`

Funktioner deklarerade med `const` gör objektet read-only. Med `mutable` inför man ett undantag.

Slide 19

```
class A {
    int get() const;    // objektet är read-only i get
    int i;
    mutable int access_count;
};

inline
int A::get() const
{
    access_count++;    // ok: får ändras trots const
    return i;
}
```