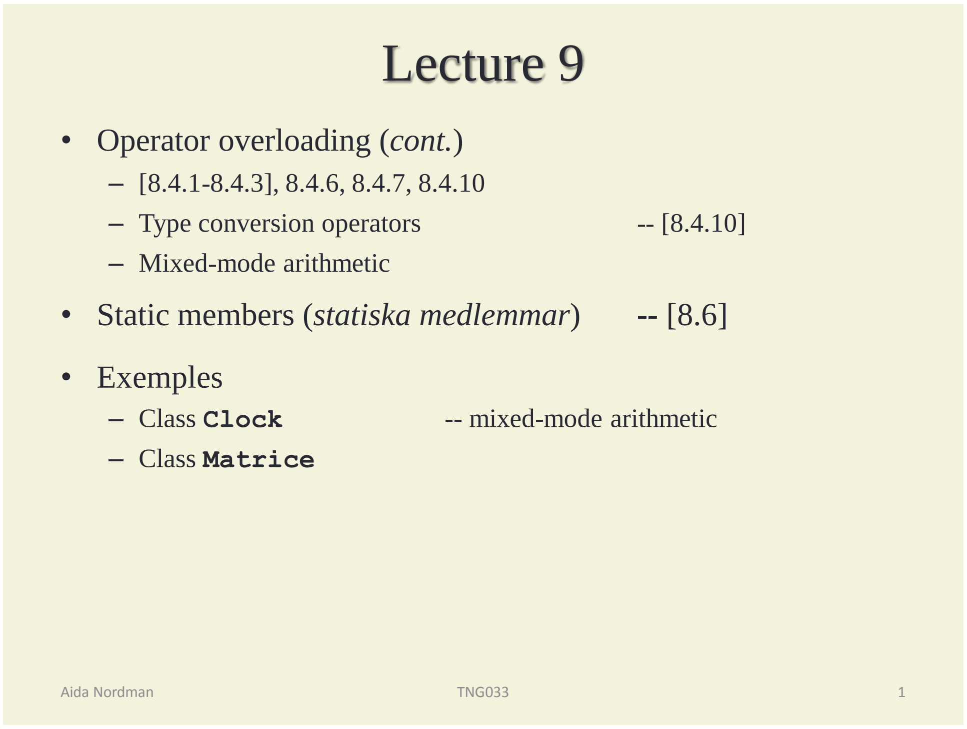
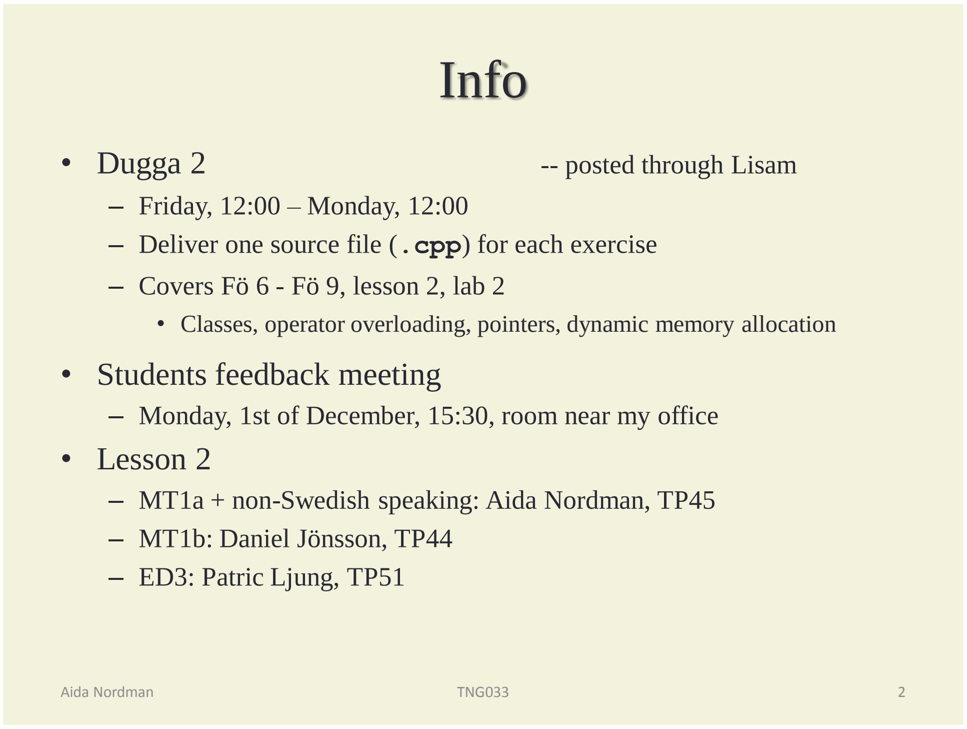


Lecture 9

- Operator overloading (*cont.*)
  - [8.4.1-8.4.3], 8.4.6, 8.4.7, 8.4.10
  - Type conversion operators -- [8.4.10]
  - Mixed-mode arithmetic
- Static members (*statiska medlemmar*) -- [8.6]
- Exemples
  - Class **`Clock`** -- mixed-mode arithmetic
  - Class **`Matrice`**



Info

- Dugga 2 -- posted through Lisam
  - Friday, 12:00 – Monday, 12:00
  - Deliver one source file (**`.cpp`**) for each exercise
  - Covers Fö 6 - Fö 9, lesson 2, lab 2
    - Classes, operator overloading, pointers, dynamic memory allocation
- Students feedback meeting
  - Monday, 1st of December, 15:30, room near my office
- Lesson 2
  - MT1a + non-Swedish speaking: Aida Nordman, TP45
  - MT1b: Daniel Jönsson, TP44
  - ED3: Patric Ljung, TP51

Class `Clock`

```
class Clock
{
  public:
    //constructors
    Clock(int n = 0);
    Clock(int h, int m, int s);
    …

    friend ostream& operator<<(ostream& os, const Clock& c);

  private:
    //represent time as hh:mm:ss
    int hh, mm, ss;
} ;
```

Conversion constructor
Convert `int` to `Clock`

Read sec. 8.4.10

```
Clock K1;      //00:00:00
Clock K(60);   //00:01:00

K1 = 120;      //00:02:00
```

The conversion constructor is automatically called to create an object of class `Clock`

```
Clock temp(120);
K1 = temp;
```

Operators

- Operators that can be overloaded in **C++**

`+`	`-`	`*`	`/`	`%`	`^`	`&`	`|`	`~`
`!`	`=`	`<`	`>`	`+=`	`-=`	`*=`	`/=`	`%=`
`^=`	`&=`	`|=`	`<<`	`>>`	`<<=`	`>>=`	`==`	`!=`
`<=`	`>=`	`&&`	`||`	`++`	`--`	`,`	`->*`	`->`
`( )`	`[ ]`	`new`	`delete`	`new[]`	`delete[]`			

- How to overload the operators?
  - Number of operands cannot be changed
  - Priority cannot be changed

Read sec. 8.4

Operator overloading

```
Clock K1(10,30,0), K2(5,0,0);
Clock K3;
++K1; //inc. 1s
K3 = K1 + K2;
K3 = ++K1 + K2;
if (K1 == K2) …
if (K1 != K2) …
```

Define for class **Clock**

```
operator++    operator+
operator==    operator!=
```

Operator overloading

```
Clock K1(10,30,0), K2(5,0,0);
if (K1 == K2) …;
```

```
K1.operator==(K2);
```

operator== is a member function of class **Clock**

```
class Clock
{
  public:
    …
    bool operator==(const Clock &c) const;

  private:
    //represent time as hh:mm:ss
    …
} ;
```

Function can be called for *const* **Clock** objects

Operator overloading

Relational operators

```
#include <utility>
using namespace std::rel_ops;
class Clock
{
  public:
    …
    bool operator==(const Clock &c) const;
    bool operator<(const Clock &c) const;
  private:
    //represent time as hh:mm:ss
    …
} ;
```

Library that provides **operator!=**, **operator>**, **operator<=**, *etc*

Operator overloading

```
Clock K1(10,30,0);
++K1;  //10:30:01
```

```
K1.operator++();
```

Read section 8.4.3 of course book for pos-increment (pag. 238)
K1++;

operator++ as a member function of class **Clock**

```
class Clock
{
  public:
    …
    Clock& operator++(); //pre-increment
  private:
    //represent time as hh:mm:ss
    …
} ;
```

Allow cascading
++(++K1);

Operator overloading

```
Clock K1(10,30,0);
K1++;   //pos-increment
```

```
K1.operator++(int);
```

this

`operator++` as a member function of class `Clock`

```
class Clock
{
  public:
    …
    const Clock operator++(int);
  private:
    //represent time as hh:mm:ss
    …
} ;
```

No cascading allowed
`(K1++)++;`

Operator overloading

```
Clock K1(10,30,0), K2(5,0,0), K3;
K1 += K2;
```

```
K1.operator+=(K2);
```

this

`operator+=` as a member function of class `Clock`

```
class Clock
{
  public:
    …
    const Clock& operator+= (const Clock &c);
private:
    //represent time as hh:mm:ss
    …
} ;
```

Assignment operators associate right to left

Allow for cascading
`K3 = (K1 += K2);`
`K3 += K1 += K2;`

But, disallow
`(K3 += K1) += K2;`

Operator overloading

```
Clock K1(10,30,0);

K1 += 10;
```

Add 10 seconds to the clock ***K1***

```
K1.operator+=(10);
```

Shall we had another **operator+=** to class **Clock**?

```
class Clock
{
  public:
    …
    const Clock& operator+= (int i);
  private:
    //represent time as hh:mm:ss
    …
} ;
```

Operator overloading

```
Clock K1(10,30,0), K2(5,0,0), K3;

K3 = K1 + K2;
```

```
K1.operator+(K2);
```

operator+ as a member function of class **Clock**

```
class Clock
{
  public:
    …
    const Clock operator+(const Clock &c);
  private:
    //represent time as hh:mm:ss
    …
} ;
```

What are the consequences of this design?

Operator overloading

```
class Clock
{
  public:
    …
    const Clock operator+(const Clock &c);

  private:
    //represent time as hh:mm:ss
    …
} ;
```

```
if  (K1 + K2  =  K3)
     … ;
```

Compiler gives a compilation error ☺

Did we want to write this?

`if  (K1 + K2  ==  K3)…`

Mixed-mode arithmetic

```
Clock K1(10,30,0), K3;

//add 10 seconds with K1
K3 = K1 + 10;

K3 = 10 + K1;  //does not compile
```

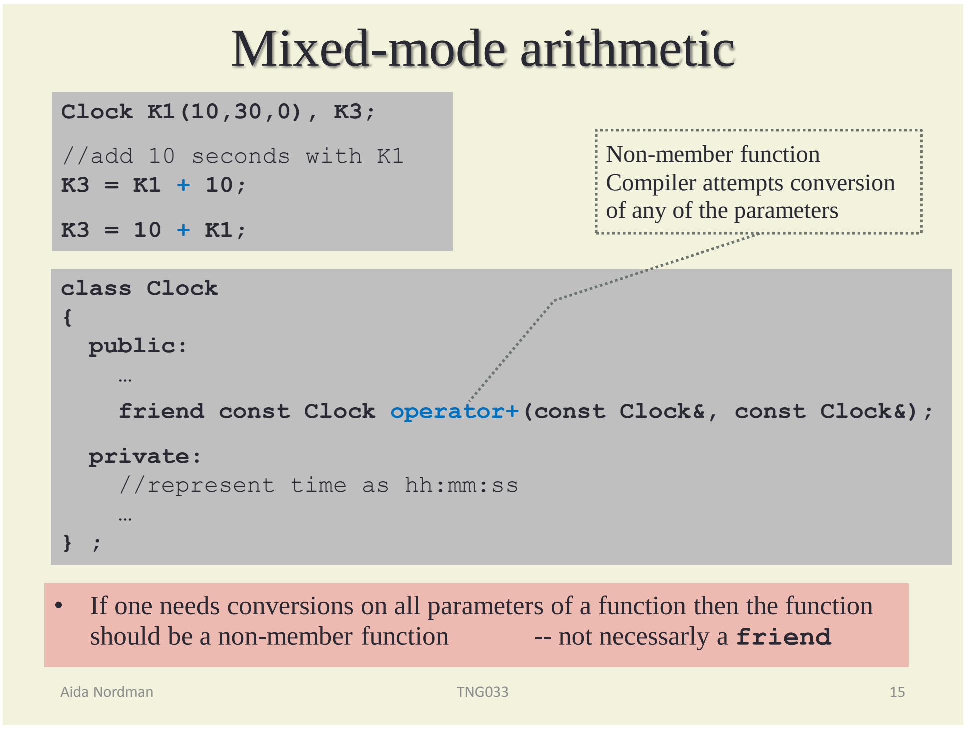
```
K1.operator+(10);
```

```
operator+(10, K1);
```

```
const Clock temp(10);
K3 = K1 + temp;
```

Addition is no longer commutative!!

- Parameters are considered for implicit type conversion only if they are listed in the parameters list
- The object pointed by **`this`** is never eligible for implicit conversion



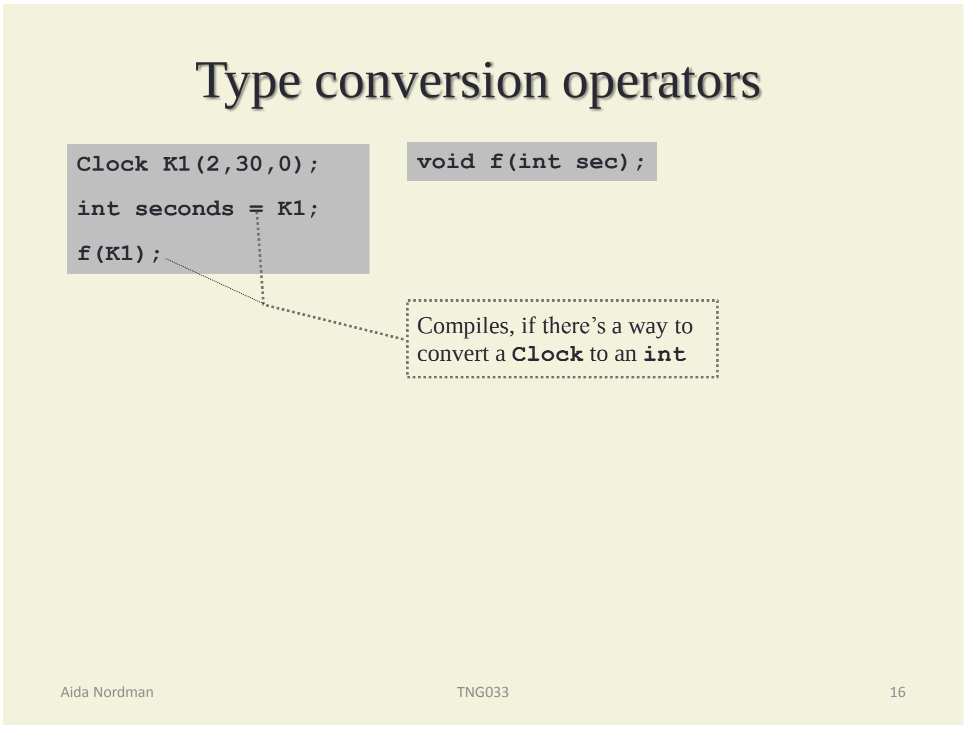
Mixed-mode arithmetic

```
Clock K1(10,30,0), K3;

//add 10 seconds with K1
K3 = K1 + 10;

K3 = 10 + K1;
```

Non-member function
Compiler attempts conversion of any of the parameters

```
class Clock
{
  public:
    …
    friend const Clock operator+(const Clock&, const Clock&);

  private:
    //represent time as hh:mm:ss
    …
} ;
```

- If one needs conversions on all parameters of a function then the function should be a non-member function -- not necessarily a **`friend`**



Type conversion operators

```
Clock K1(2,30,0);

int seconds = K1;

f(K1);
```

```
void f(int sec);
```

Compiles, if there's a way to convert a **`Clock`** to an **`int`**

Type conversion operator: convert **`Clock`** to **`int`**

```
class Clock
{
  public:
    …
    operator int() const;
    …
  private:
    //represent time as hh:mm:ss
    int hh, mm, ss;
} ;
```

`operator` *`type`*`() const;`

- Member function
- No arguments
- No return type
- To convert objects of the class to *`type`*
- Call **automatically** when a conversion is needed

```
Clock::operator int() const
{
   return (hh*60*60 + mm*60 + ss);
}
```

```
Clock K1(2,30,0);

int seconds = K1;
```

Read sec. 8.4.10

`Matrice` class

Read sec. 8.4.6 and sec 8.4.7

- How to define a subscript operator? -- **`M[i,j]`**
  - **`operator[]`** can only have one argument
- Solution
  - Overload function **`operator()`** -- **`M(i,j)`**
  - Can have any number of arguments

line column

```
Matrice M(5, 5, -1);

//modify diagonal
for(int i = 0; i < 5; i++)
        M0(i,i) = 8.8;
```

Function operator

```
class AA
{
  public:
    …
    returnType operator() (parameters);
  private:
    …
}
```

```
AA aa;
…
aa(p1, p2, p3, …);
```

Function object

Bad programming practice

There is something counterintuitive here?

```
class Matrice
{
  public:
    …
    double& operator()(int l, int c) const;
  private:
    …
}
```

Important

- Read advised book sections
- Study the code for classes **`Clock`** and **`Matrice`** given with this Fö
- Do exercises for Lesson 2

- Recall class **`Clock`**
- Consider that we want to keep track of how many **`Clock`** objects exist in a program
- How can this been done?

```
#include "clock.h"

int fun2(Clock &c);

Clock fun3(…);

int main()
{
   Clock K(10,30,0);
   …
   return 0;
}
```

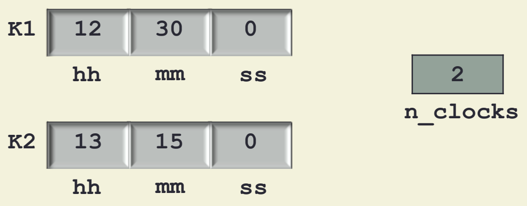
Static members

- Functions and data shared by all objects of the class

```
class Clock {
  public:
    …
  private:
    //represent time as hh:mm:ss
    int hh, mm, ss;
    static int n_clocks;
} ;
```

Counter of the number of existing clocks (i.e. **Clock** objects)

There is only one var. **n_clocks** shared by all objects



Static data members

```
class Clock {
  public:
    Clock(); //default constructor
    Clock(int h, int m, int s);
    //copy constructor
    Clock(const Clock& K)
   ~Clock(); //destructor
    …
  private:
    //represent time as hh:mm:ss
    int hh, mm, ss;
    static int n_clocks;
} ;
```

Constructors should increment **n_clocks**

Destructor should decrement **n_clocks**

Needs to be initialized to zero. How?

See class **Clock** in the folder **Clock_static** Fö 9

Static member functions

```
class Clock {
  public:
    …
    static int number_of_clocks()
    { return n_clocks; };
  private:
    //represent time as hh:mm:ss
    int hh, mm, ss;
    static int n_clocks;
} ;
```

Static member functions can only handle static data members

Static member functions can also be defined in the source file (**.cpp**)

Read sec. 8.6

```
Clock K1(12,30,0);
Clock *ptr_k = new Clock(13,15,0);
cout << "Number of existing clocks = "
     << Clock::number_of_clocks() << endl;
```

Call a static member function

Aida Nordman TNG033 25

Next…

- Inheritance (*arv*) [sec. 9.1 - 9.5]
  - base class and derived class
  - constructors an destructors
  - `public`, `private`, and `protected` data members

Aida Nordman TNG033 26