

TDT4102

Prosedyre- og objektorientert programmering,
Øvingsforelesning veke 7, 15. februar

The image features a large, stylized 'C++' logo in blue. Behind the logo is a snippet of C++ code in a monospaced font, tilted at an angle. The code includes a header, namespace declaration, and a main function that prints 'Hello World!'.

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello World!\n";
    return 0;
}
```

Sivert Krøvel

sivertkr@stud.ntnu.no

Anna Rekdal

annalr@stud.ntnu.no

Agenda

- Litt frå forrige øving
- Konstruktører
- Meir om klasser
- Operatoroverlagring
- const
- vector-klassen
- Oppgåver

Kva er problemet her? (Ø4)

```
int randomWithLimits(int lower, int higher);
```

```
void randomizeCString (char tabell[], int  
antall, char nedre, char ovre);
```

I main:

```
char tab[6];
```

```
randomizeCString(tab, 6, 'F', 'A')
```

Konstruktører

Konstruktører

Instansiering: opprette eit objekt

Initialisering: gje objektet ein fornuftig startverdi

Nøkkelen her: konstruktører!

Konstruktører

- Har ALLTID same namn som klassa
- Har ingen returtype (ikkje ein gong void)
- Kan ha fleire konstruktører, skil mellom dei vha parameterlista
- Blir kalla automatisk når eit objekt instansierast, du treng ikkje å kalle han eksplisitt!

Konstruktøren er ein medlemsvariabel!

```

Class Klasse{
private:
int vilkårligTall;
public:
klasse();
};

Klasse::Klasse(){
vilkårligTall = 0;
}

int main(){
Klasse a;
}

```

Initialiseringsliste i konstruktør

```
Me(string first, string last) :  
  firstName(first),  
  lastName(last), age(22) {}
```


Delegerande konstruktør

```
Me(string first, string last)
```

```
: Me(first, last, 22){}
```

- Dette er ein delegerande konstruktør
- Forhindrar duplisering av kode ved å bruke ein tidligare implementer konstruktør

Operatoren ::

- Heiter scope-operatoren
- Blir brukt til å spesifisere klassenamnet når ein skriv ein funskjonsdefinisjon av ein medlemsfunksjon.
- Døme: `void Card:: drawcard()`
- Generelt IKKJE bruk i kode, berre i definisjon

Operatoren .

- Kallast prikk/dot- operatoren
- Blir brukt til å hente ein medlemsfunksjon til eit objekt
- objektnamn.medlemsfunksjon()
- Det er denne du brukar for å kalle medlemsfunksjonar!

Get og set

Ofte nyttige, spesielt om enkelte verdier er forbodne.

Døme!

Destruktør

- Skal gjere akkurat det motsette av ein konstruktør
- Skal «rydde opp» etter objektet
- Blir automatisk kalla når objektet går ut av scope

Deklarerast som:

```
MyClass::~~MyClass();
```

Døme på destruktør

```
class Useless {
private:
    int* a = new int;
public:
    Useless() { *a = 0; }
    ~Useless() { delete a; }
};
```

Svært ofte nyttig å overlaste <<
for klassa di

Gjer testing enklare, og klassa kan (ofte)
følest meir intuitiv når ho oppfører seg slik
du har blitt vand med.

Agenda

- Litt frå forrige øving
- Konstruktører
- Meir om klasser
- Operatoroverlagring
- const
- vector-klassen
- Oppgåver

Operatorar

- Funksjonar med spesiell syntaks
- `+ - * / -- << % & ...`og mange fleire
- Definert for grunnleggande datatypar (t.d. `int`)
- For eigendefinerte typar må du sjølv definere funksjonaliteten til (og eksistensen av) operatorane

Operatoroverlasting

- Dei fleste operatorar kan *overlastast* (dvs. å lage ein eigen «versjon» av operatoren med andre parameterlister)
- «Syntactic sugar» -> ikkje nødvendig; jobben kan like gjerne gjerast av ein funksjon, men praktisk i mange tilfelle

```
bool operator==(ComplexNumber& left, ComplexNumber& right);
```

Operatoroverlasting

To variantar:

1. Ikkje-Medlemsoperator
2. Medlemsoperator

Du kan i stor grad velge sjølv. Det eksisterer enkelte retningslinjer

Ikkje-medlemsoperator

- Deklarerast utanfor klassen
- Nyttar *public*-grensesnittet (medlemsfunksjonar). Har ikkje tilgong til private medlemmar (unntak: *friend*)

Syntaks

```
ComplexNumber operator+(ComplexNumber& left, ComplexNumber& right);
```

Deklarerast og definerast som ein funksjon, med returtype og parameterliste som vi er vande med. Funksjonsnavnet består av nøkkelordet *operator* og symbolet for operatoren

```
ComplexNumber operator+(ComplexNumber& left, ComplexNumber& right)
{
    return ComplexNumber(left.getReal() + right.getReal(), left.getImag() + right.getImag());
}
```

Den første parameteren vert venstre operand, den andre er venstre operand

Medlemsoperator

- Medlemsfunksjon av klassen
- Har full tilgong til medlemmane
- Venstre operand er alltid objektet som «eig» operatoren
- Nokre få opeatorar må vere medlemmar. t.d. indeksoperatoren []

Syntaks

```
class ComplexNumber{
    /*...*/

    ComplexNumber operator+(ComplexNumber& right);
};
```

Deklarerast og definerast som ein medlemsfunksjon, med returtype og parameterliste som vi er vande med. Funksjonsnavnet består av nøkkelordet *operator* og symbolet for operatoren. Ein eventuell parameter vert høgre operand

```
ComplexNumber ComplexNumber::operator+(ComplexNumber & right)
{
    return ComplexNumber(this->getReal() + right.getReal(), this->getImag() + right.getImag());
}
```

Objektet sjølv, tilgjengelig via *this*-peikaren vert venstre operand. Den eine parameteren vert høgre

Konvensjonar

- Først og fremst: implementer operatorar som gir meining (brøk+brøk gir meining, men kva betyr bil+bil?)
- Overlasta operatorar bør fungere slik brukaren forventar, dvs. på same måte som dei fungerer for enkle datatypar
- Meir detaljar neste veke

<< operatoren for printing

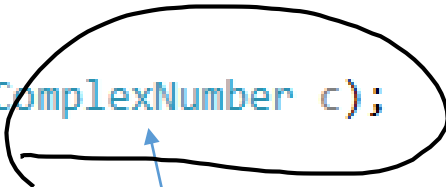
Treng ein *std::ostream&* på venstre side,
bør også returnere denne

```
std::ostream& operator<<(std::ostream& os, ComplexNumber c);
```

<< operatoren for printing

Treng ein *std::ostream&* på venstre side, bør også returnere denne

```
std::ostream& operator<<(std::ostream& os, ComplexNumber c);
```



Kva med denne?
Bør vi endre noko
her?

<< operatoren for printing

Treng ein *std::ostream&* på venstre side, bør også returnere denne

```
std::ostream& operator<<(std::ostream& os, const ComplexNumber& c);
```

const-referanse
Vi skal ikkje endre
på denne i
funksjonen

Demonstrasjon

Overlasting av operatorar for ComplexNumber-klassa

Agenda

- Litt frå forrige øving
- Konstruktører
- Meir om klasser
- Operatoroverlagring
- const
- vector-klassen
- Oppgåver

Kva betyr *const*?

- *Type qualifier*. Gir meir informasjon om objektet.
- Spesifiserer at objektet er *read only*, dvs. det kan ikkje endrast

```
const int P = 90;
P = 40; //fyfy
```

Expression must be a modifiable lvalue

Kvifor bruke *const*?

Forhindre feil (utilsikta endringar etc.)

Avgrensar tilgongen objekt har til å endre på kvarandre

Vanlig i samband med medlemsfunksjonar for å garantere rett oppførsel

Bruk *const* i klasser

- Ein medlemsfunksjon som er deklarert *const* garanterer at han ikkje endrar på objektet
- Ein funksjon som tek inn eit *const*-objekt (eller ein referanse eller peikar til eit slikt) kan *kun* bruke *const*-funksjonar

Nokre retningslinjer

- 1) Ein *const*-funksjon kan kun bruke andre *const*-funksjonar
- 2) Ein «vanlig» funksjon kan bruke alle funksjonar, også *const*
- 3) Eit *const*-objekt kan kun bruke *const*-medlemsfunksjonar
- 4) Eit «vanlig» objekt kan bruke alle

Oppsummering

For medlemsfunksjonar,
bruk *const* viss du kan

Fleire detaljar neste veke,
når vi jobbar meir med
operatoroverlagring

Agenda

- Litt frå forrige øving
- Konstruktører
- Meir om klasser
- Operatoroverlagring
- const
- vector-klassen
- Oppgåver

std::vector

- Container-klasse frå standardbiblioteket
- Ein «smartare» tabell.
- Kan endre storleiken undervegs, slette element frå vilkårlige plassar, etc
- Veit sjølv kor stor den er

std::vector

Typen til innhaldet må spesifiserast i deklarasjonen

```
std::vector<int> myIntVector;  
myIntVector.push_back(13);
```

HUGS: #include<vector>

Nokre nyttige medlemsfunksjonar

- `push_back(const T& e)`: legg elementet til i slutten av lista
- `size()`: returnerer antal element i vektoren
- Indeks-operatoren `[]`: gjer slik at du kan bruke same syntaksen som ved tabellar, t.d. `myIntVector[3] = 13;`

Oppgåver

- ComplexNumber frå forrige veke ligg på it's learning (representerer komplekse tal, har medlemmane *real* og *imag*, samt to konstruktørar)
- Overlast << operatoren for å skrive ut objektet til skjerm
- Overlast nokre aritmetiske (+, -, *, /) og logiske (<, > ==...) operatorar