TDT4102
Prosedyre- og objektorientert programmering, Øvingsforelesning veke 6, 8. februar

Sivert Krøvel
sivertkr@stud.ntnu.no

Agenda

- Tabellar
- Eigendefinerte typar
- Konvensjonar
- Objektorientering
- Introduksjon til klasser
- Oppgåver

Tabellar - repetisjon

- Ein tabell er ei liste med verdiar av *same datatype*
- Samanhengande blokk i minnet
- Ein tabellvariabel er ein referanse til *det første* elementet
- Storleiken må vere bestemt når programmet *kompilerer*

```
int tabell[10];
```

Deklarere ein tabell

Storleiken må vere *ein konstant*

```
int size = 10;
int tabell[size];
```

Dette går ikkje:
«expression must have a constant value»

Dette går fint

```
const int SIZE = 10;
int tabell[SIZE];
```

Å bruke tabellar

«Variabelen veit ikkje sjølv at den er ein tabell»:

- Vi må sjølve ha kontroll på storleiken
- Vi kan ikkje bruke operatorar direkte (t.d. ==, <, >>)
- Det går, men dei gjer ikkje det vi trur!

Å bruke tabellar i funksjonar

- Når vi tek ein tabell som argument i ein funksjon, må vi også ta inn storleiken!
- To måtar å ta inn ein tabell, peikarsyntaks og tabellsyntaks:

```
void printTable(int *table, int size)

void printTable(int table[], int size)
```

Døme, compareTables

Ein funksjon som tek inn to tabellar og returnerer ein bool. Sant dersom dei er like, usant dersom dei ikkje er det

Agenda

- Tabellar
- Eigendefinerte typar
- Konvensjonar
- Objektorientering
- Introduksjon til klasser
- Oppgåver

Eigendefinerte datatypar

- Verda består av meir enn berre int og bool
- Meir og mindre komplekse objekt
- Ulike objekt påverkar kvarandre
- Forskjellige objekt har likskapar og ulikskapar

Eigendefinerte datatypar

- Vi kan lage eigne datatypar for å representere praktisk talt det vi vil
- Ulike kategoriar
- enum, struct og class

enum

- Enumerated type
- Ein variabel som kan ha ein av ei avgrensa mengde verdiar
- Kvar mulige verdi er ein *konstant* av typen som er definert
- Praktisk for å lage variablar som kan ha ein av nokre få lovlige verdiar

enum

Syntaks

Typenavn, brukast i variabeldeklarasjonar

Eit sett med konstantar

Startar i utgangspunktet med verdien 0, deretter 1, så 2 osv

```
enum Beatle {JOHN, PAUL, GEORGE, RINGO};
```

Nøkkelord,
«Vi vil definere ein ny enum-type»

Ein variabel av typen Beatle

```
Beatle myBeatle = JOHN;
```

struct

- Data structure
- Samlar data som «hører saman» i eitt objekt
- Medlemsvariablar av vilkårlige typar (kan godt vere ulike)
- Medlemsvariablane kan aksesserast med dot-operatoren (.)

struct

Syntaks

Typenavn

Nøkkelord, «Vi vil definere ein ny struct-type»

```
struct Date {
    int day;
    std::string month;
    int year;
};
```

Medlemsvariablar

Ein variabel av typen Date

```
Date today = { 8, "Februar", 2017 };
```

class

- Samlar *data* og *eigenskapar* i eitt objekt
- Medlemsvariablar og medlemsfunksjonar
- Medlemmar kan aksesserast vha dot-operatoren
- myClass.memberFunction();

class

Syntaks

Typenavn

Nøkkelord, «Vi vil definere ein ny class-type»

```
class Person {
private:
    std::string name;
    Date dateOfBirth;
public:
    Person();
    Person(std::string name);
    Person(int day, int month, int year);

    void setDateOfBirth(Date date);
    void setName(std::string name);
};
```

Private medlemmar

Public medlemmar

Tenke- og diskusjonsoppgåve

Tenk ut eit døme på kva dei tre ulike konstruksjonane (enum, struct, class) kan brukast til
(Korleis er dei forskjellige?)

Eigendefinerte datatypar

- Kan brukast som alle andre datatypar, i tabellar og medlemsvariablar, som returtype og parameter i funksjonar
- Det går sjølvsagt også an å lage peikarar og referansar til dei
- MEN: dei fleste operatorane (==, +, -, <, > etc) fungerer *ikkje* utan vidare

Demonstrasjon

Døme på enum, struct og class

Agenda

- Tabellar
- Eigendefinerte typar
- Konvensjonar
- Objektorientering
- Introduksjon til klasser
- Oppgåver

Nokre konvensjonar

- Det er mykje som er lov, men ikkje alt som er lurt
- Oversiktligheit og lesbarheit, både for andre og for deg sjølv (nokre dagar seinare)
- «Best practice», motverkar feil

Navnekonvensjonar

- *Variablar* skriv vi med liten forbokstav, og ordgrenser er markert med stor forbokstav (camelCase)
- *Konstantar* skriv vi med store bokstavar (upper case). Ordgrenser markert med understrek
- *Eigendefinerte typar* har stor forbokstav, elles camelCase

Navnekonvensjonar

```
const int MY_CONSTANT = 10;

int myInteger = 10;

class MyClass {
    int member;
};
MyClass myClass;
```

Inndeling i filer

- Det er vanlig å la kvar klasse ha si eiga fil (.h og .cpp) oppkalt etter seg
- Klassen *Person* er definert i *Person.h* og medlemsfunksjonane er implementert i *Person.cpp*
- Klassedefinisjonen, potensielt saman med enkle funksjonsdefinisjonar, fins i headerfila, elles brukar vi cpp-fila

Agenda

- Tabellar
- Eigendefinerte typar
- Konvensjonar
- Objektorientering
- Introduksjon til klasser
- Oppgåver

Objektorientert programmering

- Arbeider med *objekt*, dvs instansar av klasser
- Samhandling mellom objekt
- Kombinerer data og funksjonalitet

Litt terminologi

- Ein *klasse* er ein eigendefinert datatype
- Ein variabel med denne typen vert kalla ein *instans* av klassen. Når du opprettar ein ny variabel, *instansierer* vi den
- Variablar av klassetypar vert også kalla *objekt*

Terminologi

Klassedeklarasjon

```
class MyClass {
    int member;
};
```

Instansar/objekt

```
MyClass myClass;
MyClass arbitraryVariableName;
```

Innkapsling

«Utanforståande» skal ikkje ha tilgang til meir enn nødvendig

- Forhindrar feilaktige eller uautoriserte endringar av data
- Hindrar ein heil del «brukarfeil»
- Nøkkelorda *public* og *private* (og *protected*, men det kjem vi tilbake til)

Agenda

- Tabellar
- Eigendefinerte typar
- Konvensjonar
- Objektorientering
- Introduksjon til klasser
- Oppgåver

Klasser

- Deklarerast i headerfila <classname>.h(pp)
- Medlemsfunksjonane implementerast i kjeldefila <classname>.cpp
- Har som regel fleire medlems*variablar* og medlems*funksjonar* (også kalt *metoder*)

Døme, personklasse

Headerfil (Person.h)

```
class Person {
private:
    std::string name;
    Date dateOfBirth;
public:
    Person();
    void setDateOfBirth(Date date);
    void setName(std::string name);
};
```

Kjeldefil (Person.cpp)

```
void Person::setDateOfBirth(Date date) {
    dateOfBirth = date;
}

void Person::setName(std::string name) {
    this->name = name;
}
```

Døme, personklasse

I main:

```
int main() {
    Person p1;
    Date dato = { 8, "Februar", 1995 };

    p1.setName("Ola Nordmann");
    p1.setDateOfBirth(dato);
}
```

Legg merke til korleis vi kallar på medlemsfunksjonar!

Medlemsfunksjonar

- I utgangspunktet: tilhøyrer objektet, dvs. ein instans av klassen
- Alle objekt har «sin eigen» versjon av funksjonen (unntak: static, ein medlemsfunksjon som er deklarert som static gjeld for alle instansar av klassen)

Medlemsfunksjonar

- Som alle andre medlemmar: aksesserast vha dot-operatoren (.)

```
int main() {
    Person p1;
    Date dato = { 8, "Februar", 1995 };

    p1.setName("Ola Nordmann");
    p1.setDateOfBirth(dato);
}
```

Konstruktørar

- Instansiering: å opprette eit nytt objekt (t.d. ved å deklarere ein variabel av klassetypen)
- Initialisering: å gi objektet ein fornuftig startverdi
- Nøkkelen er: konstruktørar

Konstruktørar

- Ein funksjon utan returtype, oppkalt etter klassen
- Blir kalt «automatisk» når eit objekt *instansierast*
- Gir fornuftige verdiar til medlemsvariablane, og utfører ev. andre oppgåver som trengs for å bruke objektet vidare

Konstruktørar

- Er alltid til stades, automatisk generert dersom du ikkje lagar ein sjølv
- Fleire forskjellige (med ulike parameterlister)

Kva er ein konstruktør?

Ein funksjon som kjører «automatisk» når du opprettar ein ny variabel. Den initialiserer medlemsvariablane til klassen, mm.

Du treng altså ikkje å kalle på kunstruktøren eksplisitt!
(Dersom du gjer det lagar du faktisk fleire instansar enn du hadde tenkt!)

Konstruktørar i bruk

```
class Person {
private:
    std::string name;
    Date dateOfBirth;
public:
    Person();
    Person(std::string name);
    Person(int day, int month, int year);
};
```

```
int main() {
    Person p1;
    Person p2("Ola Nordmann");
    Person p3(8, 2, 1995);
}
```

Demonstrasjon

Når kjører konstruktøren?

Agenda

- Tabellar
- Eigendefinerte typar
- Konvensjonar
- Objektorientering
- Introduksjon til klasser
- Oppgåver

Oppgåver

- Lag ein klasse for å representere komplekse tal (eit komplekst tal består av ein reell og ein imaginær del)
- Lag to konstruktørar, ein som initialiserer talet til 0 (både reell og imaginær), og ein som let deg definere reell og imaginær del sjølv
- Lag ein medlemsfunksjon som let deg printe innhaldet til skjerm (t.d. «3 + 5i»)