TDT4102
Prosedyre- og objektorientert programmering, Øvingsforelesning veke 8, 22. februar

Sivert Krøvel
sivertkr@stud.ntnu.no

Agenda

- Meir om operatoroverlasting, const og referanser
- Meir om std::vector
- Øving 6 – del 1
- Pause
- Øving 6 – del 2
- Dynamisk minnehandtering
- Grafikkeksempel

Operatorar- repetisjon

- 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

Call-by-reference

Tommelfingerregel:
Dersom du ikkje har ein god grunn til å la vere, bør operatoren din ta inn argumenta som const referanser

Kvifor?

Call-by-reference

Tommelfingerregel:
Dersom du ikkje har ein god grunn til å la vere, bør operatoren din ta inn argumenta som const referanser

Då unngår du unødig kopiering (call-by-value) og utilsikta endring av parametrane

Return-by-reference

Enkelte operatorar bør også returnere ein referanse. Dette gjeld til dømes om resultatet skal kunne brukast på venstre side i ei tilordning

```
int a = 1, b = 2, c = 3;
a += c += b;
```

Dette bør vere lov også for dine eigne datatypar

Return-by-reference

Enkelte operatorar bør også returnere ein referanse. Dette gjeld til dømes om resultatet skal kunne brukast på venstre side i ei tilordning

Dette gjeld også for ostream-<< operatoren. Dette er fordi ein ostream ikkje kan kopierast

Vanlige prototypar

Cheatsheet kjem snart

Demonstrasjon

Nokre operatorer til ComplexNumber (litt repetisjon)

const i forskjellige kontekstar

```
class Person {
private:
    string name;
    const Person* bestfriend;
public:
    string getName() const;
    void setName();
    void addFriend(const Person* newFriend);
    const Person* getFriend() const;

};
```

Kva betyr const i dei ulike tilfella?

const i forskjellige kontekstar

```
class Person {
private:
    string name;
    const Person* bestfriend;
public:
    string getName() const;
    void setName();
    void addFriend(const Person* newFriend);
    const Person* getFriend() const;

};
```

Kva betyr const i dei ulike tilfella?

std::vector

- Kan innehalde «kva som helst», Person, const Person*, std::string, const Meeting* osv osv.
- Datatypen til innhaldet spesifiserer vi i deklarasjonen inne i <>
- Kan endre storleik og legge til og fjerne element undervegs, også midt inne i lista

Korleis iterere gjennom std::vector

Demonstrerer tre ulike måtar, indeksoperator, range-based for og iteratorar (ikkje forelest enno)

```
vector<Person> vec;
```

Dette er vektoren vi skal iterere gjennom.
Personklassen er den same som på tidligare slide,
den har ein medlemsfunksjon `string getName()`

Iterasjon vha indeks

```
for (int i = 0; i < vec.size(); i++) {
    cout << vec[i].getName() << endl;
}
```

Range-based for

```
for (Person p : vec) {
    cout << p.getName() << endl;
}
```

Iterasjon vha iterator

```
vector<Person>::iterator it;
//også mulig: auto it = vec.begin();
for (it = vec.begin(); it != vec.end(); ++it) {
    cout << it->getName() << endl;
}
```

Iteratorar er nyttige

Brukast i medlemsfunksjonane erase() og insert() til å fjerne og legge til ting ein vilkårlig plass i vektoren

Iteratorar er nyttige

erase() og insert() kan gjere iteratoren ugyldig!

Dei returnerer ein ny iterator. Det kan vere lurt å avslutte itereringa etter ein slik operasjon, eventuelt finne på ei anna smart løysing.

Døme på bruk av erase

```
//fjerne ein person med eit gitt navn frå lista:
void removePerson(Person toRemove, vector<Person> vec) {
    for (auto it = vec.begin(); it != vec.end(); ++it) {
        if (it->getName() == toRemove.getName()) {
            vec.erase(it);
            //erase-funksjonen kan ugyldiggjere iteratoren,
            //så vi bryt løkka
            break;
        }
    }
}
```

Agenda

- Meir om operatoroverlasting, const og referanser
- Meir om std::vector
- Øving 6 – del 1
- Pause
- Øving 6 – del 2
- Dynamisk minnehandtering
- Grafikkeksempel

Øving 6 – Oversikt del 1

- Tre klasser: Car, Person og Meeting
- Ein enum: Campus
- Klassene inneheld peikarar til kvarandre
- Brukar av std::vector
- const-peikarar => hugs å markere medlemsfunksjonar som const!

Car-klassa

- Veit kor mange ledige sete som er ledige, og eit ledig sete kan reserverast

Public-interface:

```
bool hasFreeSeats()
void reserveFreeSeat()
```

+konstruktørar

Person-klassa

- Har eit navn, epostadresse, og ein peikar til ein Car
- Korleis representere ein person utan bil?

Public-interface:
get/set-funksjonar, `hasAvailableSeats()`

operatorar (<, == og <<) og konstruktørar

Person-klassa

- Har eit navn, epostadresse, og ein peikar til ein Car
- Korleis representere ein person utan bil? *nullptr*

Public-interface:

get/set-funksjonar, `hasAvailableSeats()`

operatorar (<, == og <<) og konstruktørar

Meeting-klassa

- Skildrar eit møte, med tidspunkt, stad, emne, møteleiar og deltakarliste
- Leiaren og deltakarlista består av (konstante) Person-peikarar (const Person*)

Meeting-klassa frå øving 6

- Koplar saman dei andre klassene de implementerer
- Arbeider med peikarar til andre objekt (ikkje kopiar, kvifor?)
- Utstrakt bruk av std::vector

Meeting-klassa

Public-interface:

`addParticipant()`:
legg til ein deltakar

`getParticipantList()`:
returnerer ei liste med *navna* til deltakarane

`findPotentialCoDriving()`:
returnerer ei liste med peikarar til deltakarar som har ledig plass i bilen sin
+ konstruktør, og operator for utskrift

static std::vector<const Meeting*> Meeting::meetings

- Ein statisk medlemsvariabel (eller funksjon) er felles for alle instansar av klassen
- Må definerast ein stad
- Her brukt for at alle instansar av klassen skal «vite om» kvarandre, dvs alle Meeting-instansar kan finne peikarar til alle andre Meeting-instansar

Destruktøren (repetisjon)

- Ein destruktør vert kalla automatisk når eit objekt blir tilintetgjort (går ut av scope)
- Skal «rydde opp» etter objektet, dersom det er nødvendig
- Dersom ingenting må ryddast, er det ikkje behov for å lage ein eigen destruktør

Destruktøren

Klassa treng ein destruktør dersom ein instans set igjen spor etter seg *utanfor seg sjølv* (sine eigne medlemsvariablar)

Til dømes ved bruk av dynamisk allokert minne, filbehandling...

Destruktøren

... eller ved at den opprettar peikarar til seg sjølv hos andre objekt

-> I øving 6 er destruktøren ansvarlig for å fjerne peikaren til objektet som destruerast i meetings-vectoren

```
Meeting::~Meeting()
{
    //rydd opp
}
```

Agenda

- Meir om operatoroverlasting, const og referanser
- Meir om std::vector
- Øving 6 – del 1
- Pause
- Øving 6 – del 2
- Dynamisk minnehandtering
- Grafikkeksempel

Øving 6 – oversikt del 2

- Tre vedlagte filer: game.cpp og gameobjects.cpp/.h
- main-funksjonen litt i game.cpp, klassene som representerer dei ulike tinga som skal tegnast fins i gameobjects.cpp/.h
- Brukar også enkelte funksjonar frå øving 3 og 4

Førebuing

1) Få SFML til å fungere, guide på it's learning
2) Legg til eksisterande filer i prosjektet (på rett måte!). Filene bør leggast i den same mappa prosjektet sjølv oppretter filene sine i

Gjennomføring

- Implementer dei ulike klassene i gameobjects.cpp/.h
- Opprett objekta og test dei undervegs i main.
- Når du er ferdig skal du kunne endre vinkelen og krafta til kanona med piltastane, og skyte med space-tasten

Gjennomgang av game.cpp

Tre hovuddelar:

Oppstart:
Vi opprettar først vindauget og alle objekta som trengs til spelet

Main-loop:
Kjører heile levetida til vindauget. Her oppdaterer vi objekta og teiknar dei på nytt

Event-loop:
Del av main-loop'en. Handterar events, som til dømes tastetrykk.

Gjennomgang av game.cpp

Hovudløkka (main-loop): handterar events (t.d. tastetrykk), oppdaterer og teiknar grafikken. Kjører ca 30 gongar per sekund

```
// main loop
while (window.isOpen()) {

    ...

}
```

Gjennomgang av game.cpp

Tastetrykk vert sendt til vindauget gjennom ein sf::Event. Desse handterast i *event-loop*en (inne i hovudløkka)

```
sf::Event event;
// while there are still unhandled events
while (window.pollEvent(event)) {
    switch (event.type) {
    case sf::Event::Closed:
        window.close();
        break;
    case sf::Event::KeyPressed:
        switch (event.key.code) {
        case sf::Keyboard::Escape:
        case sf::Keyboard::Q:
            window.close();
            break;
    }
}
```

Gjennomgang av game.cpp

For kvar iterasjon vert alle *shapes* oppdaterte, og deretter teikna på nytt. Slik får vi dei til å bevege seg på skjermen. Alt dette skjer i hovudløkka

```
if (!gameOver) {
    // update objects here
}
window.clear();
// draw objects here
window.display();
```

GameObjects

- Alle objekta som skal teiknast på skjermen i denne øvinga har ein medlemsvariabel kalt *shape*
- Datatypen til denne er enten sf::RectangleShape eller sf::CircleShape
- Desse kan teiknast i eit sf::RenderWindow, og endrast vha medlemsfunksjonar, som t.d. move() og rotate()

Demonstrasjon

update()- og draw()-funksjonane til Target

Agenda

- Meir om operatoroverlasting, const og referanser
- Meir om std::vector
- Øving 6 – del 1
- Pause
- Øving 6 – del 2
- Dynamisk minnehandtering
- Grafikkeksempel

Dynamisk minnehandtering

To nye operasjonar

- `new`:
  allokerer plass til ein ny variabel i heap-minnet
- `delete`:
  deallokerer ein tidligare allokert variabel på heapen

Dynamisk minnehandtering

Ta vare på peikarane!

- Utan peikaren veit du ikkje lenger kvar det allokerte minnet er. Då vert det vanskelig å bruke og deallokere

Bruksområde

1) Tabell med dynamisk storleik
2) To-dimensjonal tabell med dynamisk storleik

Syntaks:

```
int* dynamiskTabell = new int[lengde];
```

Bruksområde

1) Tabell med dynamisk storleik
2) To-dimensjonal tabell med dynamisk storleik

Syntaks:

```
int lengde;
cin >> lengde;
int* dynamiskTabell = new int[lengde];
```

Lengda treng ikkje vere ein konstant!

Huskeliste

- Kvar `new` treng ein `delete`
- `new[]` treng `delete[]`
- God praksis:
  Set uinitialiserte og deallokerte peikarar til `nullptr` (talverdi 0)

Agenda

- Meir om operatoroverlasting, const og referanser
- Meir om std::vector
- Øving 6 – del 1
- Pause
- Øving 6 – del 2
- Dynamisk minnehandtering
- Grafikkeksempel

Oppgåve

Vi har vist korleis vi teiknar målskiva (Target). Klarer du å få den til å bevege seg fram og tilbake?