Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

TDT4102 Prosedyre- og objektorientert programmering Vår 2021

Øving 8

Frist: 2021-03-12

Mål for denne øvinga:

- Lære om operatorar og interaksjon mellom klasser av ulike typar.
- Lære å implementere og å bruke klasser.
- Lære å bruke enkle klasser for eit enkelt grafisk brukargrensesnitt (GUI).
- Lære å bruke peikarar
- Lære å bruke `std::set`

Generelle krav:

- Bruk dei eksakte namn og spesifikasjonar gjeve i oppgåva.
- Teorioppgåver svarar du på med kommentarar i kildekoden slik at læringsassistenten enkelt finn svaret ved godkjenning.
- 70% av øvinga må godkjennast for at den skal vurderast som bestått.
- Øvinga skal godkjennast av stud.ass. på sal.
- Det anbefalast å nytte ein programmeringsomgjevnad(IDE) slik som Visual Studio Code.

Tilrådd lesestoff:

- Kapittel 16 og 17 i PPP

DEL 1: NTNU-samkøyring (60%)

Du har fått sommarjobb i det nye studentføretaket «EcoTrans» som er starta av nokre miljømedvitne NTNU-studentar. Omorganiseringa til nye NTNU har skapt eit behov for koordinert transport mellom byane Trondheim, Ålesund og Gjøvik. Kvar veke reiser mange tilsette og studentar mellom desse byane, ofte i privatbilar med ledige sete. EcoTrans vil lage eit enkelt datasystem for å stimulere til miljøvenleg samkøyring, og i denne oppgåva skal du skrive nokre kodebitar for eit slikt system.

1 Car-klassa (10%)

a) Deklarer ei klasse `Car`.

`Car` skal ha eit heiltal `freeSeats` som privat medlemsvariabel som indikerer kor mange ledige sete det er i bilen. `Car` skal også ha to `public` medlemsfunksjonar `hasFreeSeats` og `reserveFreeSeat`. `hasFreeSeats` returnerer `true` om bilen har ledige seter, og `false` elles. `reserveFreeSeat` «reserverer» eit ledig sete ved å dekrementere `freeSeats`-variabelen (du kan gå ut frå at funksjonen berre verte kalla på om det er ledige sete).

Deklarasjonar for medlemsfunksjonane:

```
bool hasFreeSeats() const;
void reserveFreeSeat();
```

Nyttig å vite: Const correctness

Det er god praksis å markere medlemsfunksjonar som ikkje endrar objektet med `const`. Dette gjer det enklare å finne feil i koden, og let oss bruke medlemsfunksjonane sjølv om objektet er konstant.

```
class NumberClass {
    int number;
    public:
    NumberClass(int number)
        : number{number} {}

    // markert const
    int getNumber() const {
        return number;
    }

    // ikkje markert const
    void setNumber(int newNumber) {
        number = newNumber;
    }
}
```

```
int main () {
    NumberClass x{3};

    int i = x.getNumber(); // OK
    x.setNumber(i+1); // OK

    const NumberClass y(4);
    int j = y.getNumber(); // OK

    // IKKJE OK!
    // Kompileringsfeil:
    // kan ikkje kalle ein funksjon
    // som ikkje er markert const,
    // på eit const objekt
    y.setNumber(j+1);
}
```

b) Deklarer og implementer ein konstruktør som tek inn kor mange ledige sete bilen har.

c) Implementer `hasFreeSeats()` og `reserveFreeSeat()`.

Hugs at deklarasjonen skal vere i `Car.h`, og definisjonen (implementasjonen) skal vere i `Car.cpp`.

2 Person-klassa (20%)

a) Deklarer ei klasse `Person`.

Denne skal ha dei private medlemsvariablane `name` og `email`, begge av typen `string`. I tillegg skal klassa ha ein privat medlemsvariabel `car`, som er ein peikar til ein `Car`. Legg merke til at vi ønskjer å bruke ein peikar og ikkje referanse til `Car`-objektet.

Grunnen til at vi ønskjer å bruke ein peikar er fordi ein peikar kan ha verdien `nullptr` og det passar fint for å representere at ein person *ikkje* har bil. Om vi nytta referanse i staden for peikar ville det vorte vanskelegare å representere dette. Dette er godt forklart i læreboka §17.9.1, og mot slutten av det avsnittet er det oppsummert når ein anbefaler pass-by-value, peikar, eller referanse-parameter.

Klassa skal ha ein konstruktør som set `name`, `email` og `car` til verdiar gjeve av parameterlista. For `car` nyttar vi `nullptr` som eit såkalla «default argument» (standard-verdi). Det tyder at konstruktøren kan brukast med berre de to første parametrane, og då vil den tredje få denne standard-verdien. Sjå også nyttig-å-vite-boks om dette temaet. Deklarer ein `get`-funksjon både for `name` og `email`. Deklarer også ein `set`-funksjon for `email`.

Nyttig å vite: pass-by-pointer

Når vi ønskjer å nytte pass-by-pointer må vi spesifisere det i funksjonsdeklarasjonen.

```
//funksjon som tar inn ein peikar til ein int
void func(int* pointer);
```

Dersom vi skal gjere eit kall på `func(int*)` må vi gi ein peikar som argument. Hugs at ein peikar eigentleg berre er ein adresse til ein variabel, adressa får vi ved å bruke addresseoperatoren, `&`.

```
int a{0};
func(&a); //&a = adressa til a
```

b) Implementer konstruktøren og get-/set-funksjonane frå førre deloppgåve.

Bruk initialiseringsliste i konstruktøren.

Nyttig å vite: default arguments

For å unngå at ein skal definere fleire ulike funksjonar som gjer det same, men har ulik tal på parametrar i parameterlista, så finnast det *default arguments*.

Til dømes kan ein funksjon som alltid skal leggje saman to tal vere standardisert til å inkrementere det første argumentet med ein, dersom det andre argumentet ikkje er oppgjeve. I staden for å lage to funksjoner som gjer same arbeid eller kallar på ein annan, er det formålstenleg å samle dei. Dette kan ein også bruke med medlemsfunksjonar i klassar.

```
void Adder(int a, int b = 1);
//void Adder(int a = 1, int b); // Gir kompileringsfeil

int main() {
    Adder(42, 42); // a+b=84
    Adder(42); // a+b=43
}

void Adder(int a, int b) {
    cout << "a+b=" << a+b;
}
```

Default argument skrivast i deklarasjonen, men ikkje i definisjonen. Argumenta som har

default-verdi må også skrivast sist i parameterlista.

c) Lag medlemsfunksjonen `hasAvailableSeats()`.
Funksjonen returnerer `true` om personen eig ein bil og bilen har ledige sete.

d) Overlast `operator<<`, som skal skrive ut innhaldet i Person til ein `ostream`. Drøft:

- Kvifor bør denne operatoren deklarerast med `const`-parameter? (t.d. `const Person& p`)
- Når bør vi, og når bør vi ikkje (ev. kan ikkje) nytte `const`-parameter?

e) Skriv testar for `Person` i `main()`.
Opprett fleire personar, og prøv å teste ulike tilfelle (t.d. har personen bil? Kva med når personen ikkje har bil?).

3 Meeting-klassa (30%)

a) Deklarer ein `scoped enum`, med namn `Campus`, som inneheld verdiar for dei ulike byane (`Trondheim`, `Ålesund` og `Gjøvik`). Overlast `operator<<` for `Campus`, som skal skrive campusnamnet til ein `ostream`.
La deklarasjonen av `enum class Campus` liggje i `Meeting.h`.

b) Definer klassa `Meeting`.
Klassa skal ha følgjande private medlemsvariablar:

```
int day;
int startTime;
int endTime;
Campus location;
string subject;
const Person* leader;
set<const Person*> participants;
```

Implementer `get`-funksjonar for `day`, `startTime`, `endTime`, `location`, `subject`, og `leader` som ein del av klassedefinisjonen.

c) Lag medlemsfunksjonen `addParticipant`.
Den skal ta inn ein peikar til eit `Person` objekt og leggje den inn i `participants`.

d) Legg til `meetings` som ein statisk, inline, privat medlemsvariabel.
`meetings` skal innehalde peikarar til alle møta som er oppretta.

```
static inline set<const Meeting*> meetings{};
```

Nyttig å vite: Static

Ein statisk medlemsvariabel er ein variabel som er felles for alle instansane av klassa. Ein kan skriva `static` før medlemsvariabelen for å sikre seg at det berre finnast ein kopi av variabelen når programmet blir køyrt, i staden for ein kopi per objekt som lagast i klassa. Til dømes i meetings-settet vårt betyr det at for alle møta som blir oppretta (altså objekt av Meeting-klassa), vil meetings-settet i kvart objekt romme dei same møta som alt finnast frå før. Difor slepp vi å måtte leggje inn alle møta som alt finnast i meetings-settet kvar gong ein opprettar eit nytt møte.

`inline` trengst for å kunna initialisera statiske variablar i klassedefinisjonen.

e) Lag ein konstruktør for Meeting-klassa som tek inn `day`, `startTime`, `endTime`, `location`, `subject`, og `leader`.
Hugs å leggje til det nye møtet i `meetings`, og at møteleiaren også er ein deltakar.

f) Lag ein destruktør for `Meeting`-klassa.

Her må du fjerne peikaren til objektet frå `meetings`. Du kan lesa om destruktørar i kapittel 17.5 i læreboka.

g) Lag funksjonen `getParticipantList()`.

Dette skal vere ein medlemsfunksjon i `Meeting`. Funksjonen har ingen parametrar og returnerer ein `vector<string>` med namn på deltakarane.

h) Overlast `operator<<` for `Meeting`.

Denne operatoren skal IKKJE vere ein `friend` av `Meeting`. Du står fritt til å velje format sjølv, men du skal skrive ut `subject`, `location`, `startTime`, `endTime`, og namnet på møteleiaren. I tillegg skal han skrive ut ei liste med namna på alle deltakarane.

Test funksjonen din frå `main()`.

i) Skriv funksjonen `findPotentialCoDriving`.

Dette skal vere ein medlemsfunksjon i `Meeting`. Funksjonen skal ikkje ta inn noko, og skal returnere ei vektor med `Person`-peikarar. Vektoren skal bestå av alle personar som:

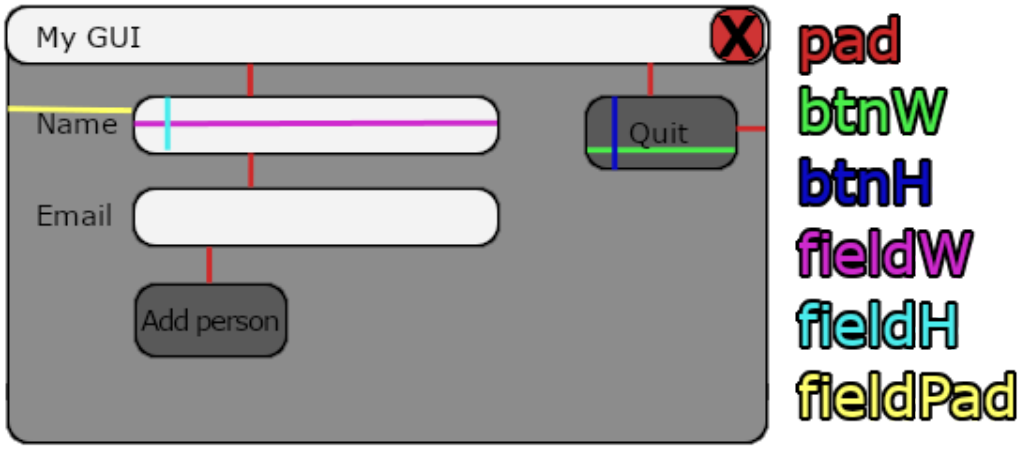
- har ledige plassar i bilen sin og
- skal til eit anna eller same møte, som
  - er på same stad som `this`-møtet,
  - er på same dag som `this`-møtet,
  - har start-tid som er mindre enn ei time forskjellig frå start-tida til `this`-møtet, og
  - har slutt-tid som er mindre enn ei time forskjellig frå slutt-tida til `this`-møtet.

Hint: Funksjonen vil ha følgjande returtype: `vector<const Person*>`.
Hugs `const correctness`.

DEL 2: GUI for samkøyring og møteplanlegging (40%)

For å gjere programmet meir brukarvenleg vil dykk lage eit grafisk brukargrensesnitt (GUI) der passasjerar kan melde seg inn. Den skal ha to tekstfelt, eit for å skrive inn passasjerens namn og eit for e-post. Det skal også vere to knappar: ein for å leggje til personen og den andre for å avslutte programmet.

Før ein går laus på kodinga til eit GUI, er det veldig lurt å danne seg ei skisse for korleis vindauget skal sjå ut. Det er også lurt å namngje alle ulike avstandar i vindauget. Dette er delvis slik ein ikkje treng å sjonglere alle dei ulike verdiane i hovudet, men mest fordi ein då kan endre på alle felles avstandar på ein stad. Under kjem ei mogeleg skisse av GUI-et der kvar farge er kopla til ein variabel. Det er ikkje eit krav at du følgjer denne skissa, men alle elementa skal vere med til slutt.



Det er anbefalt med `using namespace Graph_lib;` i `.h` fila for å unnlate `Graph_lib::`-prefikset.

Alle variablar og funksjonar i resten av oppgåvene skal vere medlem av vindauge-klassa vi lagar.

4 Oppretting av GUI (20%)

Til denne oppgåva skal vi bruke `Window` i staden for den tidlegare `Simple_window`, noko som gjer at vi overlet programstyringa til vindauget sjølv. I praksis tyder det at vi må lage vår eiga vindauge-klasse som arvar frå `Window`, og at klassa skal styre programmet.

Vidare skal vi nytte `In_box` og `Button`, begge desse klassane arvar frå `Widget` og ein må inkludera `GUI.h` for å få tilgang til dei. Sjå gjerne på kapittel 16.4 i PPP før du byrjar på oppgåva, det gir ei god introduksjon til klassane og korleis dei brukast.

a) Lag ei ny klasse `MeetingWindow`, som arvar `public` frå `Window`, og konstruktøren `MeetingWindow(Point xy, int w, int h, const string& title)`.

Plasser klassedeklarasjonen i `MeetingWindow.h`. Sidan `Window` ikkje har ein default-konstruktør, så må du kalle `Window`-konstruktøren i initialiseringslista, der argumenta skal vere dei du tok inn i `MeetingWindow`-konstruktøren. La konstruktørkroppen stå tom inntil vidare.

b) Lag eit `MeetingWindow`-objekt i `main()` og kall `gui_main()`.

Lag eit objekt av typen `MeetingWindow` og legg deretter inn eit kall på `gui_main()` i `main()`. `gui_main()` overlet programstyringa til vindauget du har laga. Prøvekøyr koden, du skal få opp eit blankt vindauge.

c) Før du byrjar på å leggje inn element, er det lurt setje inn nokre heiltal i klassa som definerer oppsettet i vindauget. Sjå skissa over for eit forslag. Sidan desse ikkje skal endrast og er definerte før kompileringa, er det lurt å deklarere dei `static constexpr int` og gi dei verdiar direkte i `.h` fila.

Nyttig å vite: `static constexpr`

Effekten av å deklarera ein medlemsvariabel `static` har blitt forklart tidlegare i oppgåva. Å deklarera ein variabel som `constexpr` gir kompilatoren beskjed om at variabelen skal evaluerast ved kompilering. Det gir oss også moglegheita til å bruke variabelen i `constexpr` funksjonar som kan evaluerast ved kompilering. Ein variabel som er `static constexpr` er difor felles for alle instansane av klassa, og evaluerast ved kompilering. Dette betyr også at den kan (og må) initialiserast direkte i `.h`-fila til klassa. På denne måten oppnår ein raskere køyring av koden, fordi den eine kopien av variabelen allereie er evaluert ved kompileringa og treng aldri å evaluerast igjen.

Nyttig å vite: Kort om callback

Ein callback-funksjon vert kalla av ein knapp i GUI-et når du trykkjer på den og har signaturen `void cb_my_callback(Address, Address pw);`. Den første parameteren bryr vi oss ikkje om, men den andre er adressa til GUI-vindauget. For at du skal kunne bruke vindauget, må du først fortelje funksjonen at den må tolke `pw` som ein `MeetingWindow`-referanse. Dette gjer du med å kalle `reference_to<MeetingWindow>(pw)`, og deretter kan du kalle medlemsfunksjonar med vanleg dot-notasjon, slik som vist under. Ytterlegare informasjon står i §16.3.1 i læreboka.

```
void cb_my_callback(Address, Address pw){
    reference_to<MeetingWindow>(pw).member_func();
}
```

d) Deklarer callback-funksjonen `void cb_quit(Address, Address pw)`.
Ein callback-funksjon må deklarerast som ein `static` medlemsfunksjon til `MeetingWindow`, dette gir GUI-et moglegheita til å kalle funksjonen uavhengig av eit objekt.

e) Implementer `cb_quit()` frå forrige deloppgåve.
Denne callback-funksjonen skal nyttast av avslutnings-knappen, så derfor må den kalle den arva medlemsfunksjonen `hide` som avsluttar vindauget.

f) Legg inn eit `Button`-objekt, `quitBtn`, som privat medlemsvariabel.
`quitBtn` må konstruerast i initialiseringslista til `MeetingWindow`. Det siste argumentet er callback-funksjonen den skal bruke, og for å sende inn ein funksjon som argument skriv du funksjonsnamnet utan parentes (`cb_quit`). For å aktivere `quitBtn` må du også passe på å kalle `MeetingWindow` sin medlemsfunksjon `attach` med `quitBtn` som argument i konstruktørkroppen til `MeetingWindow`-konstruktøren. Prøv å køyre programmet her og sjå om det funkar som forventa.

5 Person-funksjonalitet (20%)

a) Legg til to `In_box`: `personName` og `personEmail`.
Desse er to innskrivingsfelt for parametrane til ein ny `Person`. Sidan `In_box` også arvar frå `Widget` må desse også konstruerast og «attaches» i konstruktøren til `MeetingWindow`.

b) Legg inn `vector<Person*> people` som ein medlemsvariabel, og så definer og implementer ein ny funksjon, `void addPerson()`.

Denne vektoren skal innehalde peikarar til alle personar som vert lagt til gjennom tekstboksane.

`addPerson` skal lese det som er skrive inn i tekstboksane og leggje til ein ny person i vektoren med desse argumenta. Dette skal vere eit anonymt/namnlaust objekt, så her må du bruka `new`:

```
people.push_back(new Person{/*Dine argument*/});
```

For å hente innhaldet i tekstboksane, må du kalle medlemsfunksjonen `get_string()`. Sjekk også om ein av parametrane manglar, slik at det ikkje leggjast til ufullstendige personar. Du kan nytta medlemsfunksjonen `clear_value()` til å tømma tekstboksane.

c) Implementer destruktøren til `MeetingWindow`

Siden vi allokerer personar med `new` må vi frigjøre minnet med `delete`. Signaturen til destruktøren er `~MeetingWindow()`. I destruktøren må `delete` kallast på alle peikarane i `people`.

d) Legg til ein ny `Button`, `personNewBtn` med ein tilhøyrande callback-funksjon, `cb_new_person()`.

Callback-funksjonen skal kalle `addPerson`.

e) Test om programmet fungerer som venta.

Ein enkel måte å gjere dette på er å lage ein `public` funksjon som printar alle personane i vektoren. Denne funksjonen kan du kalle i `main()` etter `gui_main()`.

6 Utviding av GUI (Frivillig)

I denne oppgåva kan du fullføre GUI-et for samkøyringa. Det skal no verte to sider: ei for `Person` og ei for `Meeting`. Ein skal kunne sjå alle møte/personar som er innførte og kunne leggje inn nye. Vindauget skal ha ein knapp for å avslutte, to knappar som respektivt byttar til `Meeting`- og `Person`-sida, eit tekstfelt for informasjon, eit felt for kvar parameter å leggje inn, og to knappar som respektivt legg til ein ny `Person` eller `Meeting`. I tillegg skal der vere to felt der du kan velje ein person å leggje til eit spesifikt møte, og ein knapp som utfører dette.

a) Legg til ein ny `In_box`, `personSeats`, og ein `vector<Car*>`, `cars`.

Denne skal du bruke for å gi personane ein bil i `addPerson`. Dersom `personSeats` har eit tal som er større enn null, noko som du finn ut ved å kalle `get_int()`, lagar du eit `Car`-objekt i `cars` og gir peikaren til denne til `Person`-konstruktøren. Sjåføren må først ha plass, så du må også "reservere" eit sete i `Car`-objektet!

Hugs å kalla `delete` på alle peikarane i `cars` i destruktøren til `MeetingWindow`.

b) Legg til ein `Multiline_out_box`, `data`, som privat medlem.

Denne typen er definert i `Graph_lib.h` og er ein `Out_box` som kan vise fleire linjer. Denne skal fungere som display i vårt GUI.

c) Deklarer og definer medlemsfunksjonen `void displayPeople()`

I denne funksjonen skal du leggje inn alle personane til displayet `data`, og dette gjer du med å kalle medlemsfunksjonen `put()` som tek inn ein streng. Denne vil tolke `\n` som ny linje. Sett eit kall av denne funksjonen ved slutten av `addPerson`. Køyr programmet og sjå om personane du legg inn kjem opp på displayet.

Hint: `stringstream` har medlemsfunksjonen `str()` som returnerer innhaldet sitt som ein `string`. Du kan utnytte dette og gjenbruke tidlegare kode.

d) Vi har lyst å leggje til tilsvarande funksjonalitet for ei `Meeting`-side.

Lag to nye funksjonar, `void showPersonPage()` og `void showMeetingPage()`.

Desse skal vi bruke til å bytte mellom `Person` og `Meeting` sidene. For å gjere dette, må `showPersonPage()` kalle medlemsfunksjonen `show()` på alle element som er knytta til den sida, medan `showMeetingPage()` må kalle `hide()`. Det omvende gjeld for alle komande element som vert knytta til `Meeting`-sida. `data` kan vere felles, men då må du kalle `displayPeople()` i `showPersonPage()`.

e) Legg ein `Menu`, `pageMenu`, og to tilhøyrande callback-funksjonar, `cb_persons()` og `cb_meetings()`.

Callback-funksjonane skal respektivt kalle `showPersonPage()` og `showMeetingPage()`. Ein av parametrane til `Menu` er ein `enum` av typen `Menu::Kind` som fortel om menyen er vertikal eller horisontal. I konstruktøren til `MeetingWindow` skal du leggje til to `Buttons` i `pageMenu` som får sin respektive callback-funksjon.

Obs! I `Graph_lib` er det ein feil som gjer at alle `Button`-objekt som leggast inn i ein `Menu` må være dynamisk allokerte med `new`. Når menyen går ut av scope vil den automatisk rydde opp etter seg, så du trengjer ikkje sjølv å kalle `delete` på knappane.

f) Legg inn fire nye `In_box` til `Meeting`-sida: `meetingSubject`, `meetingDay`, `meetingStart`, og `meetingEnd`.

Desse skal vi seinare bruke for å leggje inn `Meeting`-objekt. Nå er det lurt å kalle `showPersonPage()` på slutten av `MeetingWindow`-konstruktøren.

g) Legg inn to `Choice` til `Meeting`-sida: `meetingLocation` og `meetingLeader`.

`Choice` er definert i `Graph_lib.h` og lagar ei rullegardinliste. For å leggje til eit val må du kalle funksjonen `add` med ein strengparameter. Legg inn dei tre relevante stadane til `location` i `MeetingWindow`-konstruktøren. Utvid `addPerson` til at namnet til den nye personen leggast til som eit val i `meetingLeader`.

h) Legg inn ein ny funksjon, `void addMeeting()`, ein callback som kallar denne, `cb_new_meeting()`, og ein knapp med denne callbacken, `meetingNewBtn`.

`addMeeting` skal lese parametrar frå felta og konstruere eit `Meeting` i ein ny `vector<Meeting*>`, `meetings`. `Choice` har medlemsfunksjonen `value` som returnerer posisjonen til valet i lista, så bruk dette for å finne rett stad eller rett møteleiar i vektoren over personar. Hugs å sjekke om parametrane er gyldige.

Hugs også å kalla `delete` på alle peikarane i `meetings` i destruktøren til `MeetingWindow`.

i) Lag ein funksjon som legg inn alle møta til displayet `data`.

Kall på denne i `addMeeting()` og `showMeetingPage()`.