

i **Kopi av Forside**

Institutt for datateknologi og informatikk

Eksamensoppgave (kont) i TDT4102 - Prosedyre- og objektorientert programmering

Eksamensdato : Tirsdag 9. august 2022.

Eksamenstid (fra-til) : 09:00-13:00

Hjelpemiddelkode/Tillatte hjelpemiddel: C / Bestemt, enkel kalkulator og spesifiserte trykte og håndskrevne hjelpemidler:

"Alle trykte og skrevne hjelpemidler, inkludert egen PC med alt innhold, er tillatt. Se Juks/Plagiat for hva som IKKE er tillatt. Ingen ekstra skjerm, iPad, Mobiltelefon, etc er tillatt."

Faglig kontakt under eksamen : Rune Sætre (faglærer), Marte Hoff Hagen (stipendiat).

Tlf : 452 18 103 (Rune), 416 33 507 (Marte)

ANNEN INFORMASJON

Skaff deg overblikk over oppgavesettet før du begynner på besvarelsen din.

Les oppgavene nøye, gjør dine egne antagelser og presiser i besvarelsen hvilke forutsetninger du har lagt til grunn i tolkning/avgrensing av oppgaven. Faglig kontaktperson skal kun kontaktes dersom det er direkte feil eller mangler i oppgavesettet. Henvend deg til en eksamensvakt hvis du ønsker å kontakte faglærer. Noter gjerne spørsmålet ditt på forhånd.

På flervalgsspørsmål får du positive poeng for riktige svar og negative poeng for feil svar. Summen vil aldri være mindre enn null poeng for et spørsmål, selv om du svarer mer feil enn riktig på det spørsmålet.

Juks/plagiat:

Eksamen skal være et individuelt, selvstendig arbeid. Det er tillatt å bruke hjelpemidler, men vær obs på at du må følge eventuelle anvisninger om kildehenvisninger under. Under eksamen er det ikke tillatt å kommunisere med andre personer om oppgaven eller å distribuere utkast til svar. Slik kommunikasjon er å anse som juks, og det gjelder både den som hjelper, og den som blir hjulpet. Alle besvarelser blir kontrollert for plagiat. Du kan lese mer om juks og plagiering på eksamen her: <https://innsida.ntnu.no/wiki/-/wiki/Norsk/Juks+på+eksamen>

Kildehenvisninger:

Selv om "Alle hjelpemiddel er tillatt", er det ikke tillatt å kopiere andres kode og levere den som din egen. Du kan se på andre åpent tilgjengelige ressurser, og deretter skrive din egen versjon av det du så, i henhold til copyright-forskrifter.

Varslinger:

Hvis det oppstår behov for å gi beskjeder til kandidatene underveis i eksamen (for eksempel ved feil i oppgavesettet), vil dette bli gjort via varslinger i Inspira. Et varsel vil dukke opp som en dialogboks på skjermen i Inspira. Du kan finne igjen varselet ved å klikke på bjella øverst i høyre hjørne på skjermen.

Vekting av oppgavene:

Del 1 teller ca. 20% av totalen, Del 2 teller ca. 20% av totalen, og Del 3 teller ca. 60% av totalen på denne eksamen. Du vil få F på eksamen hvis du svarer blankt på en av de tre delene.

Slik svarer du på oppgavene:

Alle oppgaver (unntatt Del 3, som er av typen filopplasting), skal besvares direkte i Inspira. I Inspira lagres svarene dine automatisk hvert 15. sekund.

NB! Klipp og lim fra andre programmer frarådes også da dette kan medføre at formatering og elementer (bilder, tabeller etc.) vil kunne gå tapt.

Filopplasting:

Når du jobber i andre programmer fordi hele eller deler av besvarelsen din skal leveres som filvedlegg – husk å lagre besvarelsen din med jevne mellomrom.

Merk at alle filer må være lastet opp i besvarelsen før eksamenstida går ut.

Det framgår av filopplastingsoppgaven hvilket filformat som er tillatt (**zip**).

Det er lagt til **30 minutter** til ordinær eksamenstid for eventuell digitalisering av håndtegnninger og opplasting av filer. Tilleggstida er forbeholdt innlevering og inngår i gjenstående eksamenstid som vises øverst til venstre på skjermen.

NB! Det er ditt eget ansvar å påse at du laster opp riktige og intakte filer. Kontroller zip-filen du har lastet opp ved å klikke “Last ned” når du står i filopplastingsoppgaven. Alle filer kan fjernes og byttes ut så lenge prøven er åpen.

Pass på at det ikke finnes noe forfatterinformasjon i filen(e) du skal levere.

I siste del av eksamen skal du bruke samme program som du satte opp i øving 0. Du må også vite hvordan du laster ned, pakker ut, og setter opp mapper fra en zip-fil som VS Code (eller tilsvarende) C++-prosjekter på maskinen din. Til slutt må du være i stand til pakke alle slike mapper sammen i en zip-fil igjen for å levere det du har kodet, innen tidsfristen, for å kunne bestå denne eksamen.

Eksamen - step by step

Denne listen vil lede deg trinn for trinn igjennom hva du skal gjøre på denne eksamen.

1. Les nøye igjennom disse **introduksjons-sidene**
2. **Last ned** medfølgende zip-fil med en gang eksamen starter. I zip-filen finner du .cpp- og .h-filer samt oppgaveteksten til del 3.
3. Les **forklaringen** på problemet gitt i begynnelse av hver seksjon.
4. Du kan bruke VS Code (eller et hvilket som helst annet utviklingsmiljø du foretrekker) til å åpne og jobbe med den oppgitte koden (som i Øving 0). All kode skal være C++.
5. Du kan bruke boken eller andre online/offline ressurser, men du kan **IKKE** samarbeide med andre på noen måte, eller direkte kopiere og lime inn online kode som om det er din egen.
6. Etter å ha fullført hver enkelt kode-oppgave bør du huske å **lagre**.
7. Send inn koden din selv om den ikke kan kompileres og / eller ikke fungerer riktig.
Fungerende kode er **IKKE** et krav for at du skal stå, men det er en fordel.
 - Last opp all den komplette koden som en .zip-fil. Ikke endre den opprinnelige mappestrukturen. For å få bestått på denne eksamen er det **HELT AVGJØRENDE AT DU LASTER OPP ZIP-FILEN**. Etter eksamen-slutt (13:00) har du 30 minutter til rådighet til dette.
 - Vi anbefaler likevel at du prøver å laste opp filen en gang underveis i eksamenstiden også, for å sjekke at du klarer det.
 - Det er mulig å oppdatere både enkelt-svarene og fil-opplastningen flere ganger, i tilfelle du retter på noe etter første innlevering.
8. Husk at funksjonene du lager i en deloppgave ofte er ment å skulle brukes i andre deloppgaver. Selv om du står helt fast på en deloppgave bør du likevel prøve å løse alle eller noen av de etterfølgende oppgavene ved å anta at funksjoner fra tidligere deloppgave er riktig implementert.
9. Før manuell sensureringen av eksamen din, vil vi foreta automatisk testing og plagiat-kontroll av all koden du har levert. Basert på resultatene kan det hende vi ber deg om en

forklaring, og dette kan påvirke eksamensresultatet ditt. Du må huske å legge til kommentarlinjer i koden din, siden det også kan hjelpe oss å forstå koden din bedre.

Nedlastning av start-fil (zip)

Vi gir dere en .zip-fil med noen forhånds-programmerte deler av det ferdige programmet.

- Last ned og lagre .zip-filen (lenken er på Inspera). Gjør dette med en gang eksamen starter... i tilfelle noe er galt, eller internett detter ut av og til underveis.
- Husk å lagre den på et sted på datamaskinen du husker og kan finne igjen.
- Pakk ut (unzip) filen.
- Begynn å jobbe med oppgavene i VS Code (eller et hvilket som helst annet utviklingsmiljø du foretrekker). Vi forventer at du vet hvordan du sammenstiller og kjører kode, slik det ble forklart i øving 0 på starten av semesteret.
- Filene vi leverer ut kompilerer til et kjørende program, men du må selv skrive resten av koden for alle oppgavene og prøve å få programmet til å kjøre som et eget prosjekt. Det er ikke et krav at all den innleverte koden kan kjøres, men det er en fordel.
- Etter at du er ferdig med kodingen av alle del-spørsmål, må du laste opp alle kodefilene dine, etter at de på nytt er pakket sammen til en lignende .zip-fil som den du begynte med (**ikke 7z, rar eller andre**). Ikke endre noe på de opprinnelige mappe/fil-navnene før du zipper filen og laster den opp til Inspera igjen. Last gjerne opp på nytt hver time!

Automatisk innlevering

Besvarelsen din leveres automatisk når eksamenstida er ute og prøven stenger, forutsatt at minst en oppgave er besvart (og at zip-fila er riktig lasta opp). Dette skjer selv om du ikke har klikket «Lever og gå tilbake til Dashboard» på siste side i oppgavesettet. Du kan gjenåpne og redigere besvarelsen din så lenge prøven er åpen. Dersom ingen oppgaver er besvart ved prøveslutt, blir ikke besvarelsen din levert. Dette vil anses som "ikke møtt" til eksamen.

Trekk/avbrutt eksamen

Blir du syk under eksamen, eller av andre grunner ønsker å levere blankt/avbryte eksamen, gå til "hamburger"-menyen i øvre høyre hjørne og velg «Lever blankt». Dette kan ikke angres selv om prøven fremdeles er åpen.

Tilgang til besvarelse

Du finner besvarelsen din i Arkiv etter at sluttida for hele eksamen er passert.

☑ **Kopi av Regler og samtykker**

REGLER OG SAMTYKKER

Dette er en **individuell** eksamen. Du har ikke lov til å kommunisere (gjennom web-forum, chat, telefon, hverken i skriftlig, muntlig eller annen form), ei heller samarbeide med noen andre under eksamen.

Før du kan fortsette til selve eksamenen må du forstå og SAMTYKKE i følgende:

Under eksamenen:

Jeg skal IKKE motta hjelp fra andre.

☐ Aksepter

Jeg skal IKKE hjelpe andre eller dele løsningen min med noen.

☐ Aksepter

Jeg skal IKKE copy-paste noe kode fra noen eksisterende online/offline kilder. (Du kan se, og deretter skrive din EGEN versjon av koden).

☐ Aksepter

Jeg er klar over at eksamenen kan bli underkjent uavhengig av hvor korrekt svarene mine er, hvis jeg ikke følger reglene og/eller IKKE aksepterer disse utsagnene.

☐ Aksepter

✓ Kopi av Samtykke til forskning

Jeg gir samtykke til at mine eksamensdetaljer kan brukes til forskningsformål av forskere ved NTNU for å forbedre undervisningen i faget.

- ☐ Aksepter
- ☐ Nei, ikke bruk resultatene mine til forskning

i Kopi av Informasjon om del 1

Du kommer nå til **del 1** av eksamen som består av flervalgsoppgaver. Denne delen inneholder **10 oppgaver** som totalt kan gi **60 poeng** og teller ca **20 %** av eksamen.

Informasjon om poenggiving: For hver oppgave på denne delen får du en *maksimal poengsum* på 5 eller 10 poeng avhengig av vanskelighetsgrad og arbeidsmengde. Det er minst et riktig svaralternativ per oppgave, men *antall riktige svaralternativ* per oppgave kan være mer enn en. For hvert riktige svaralternativ du velger får du $+\frac{\text{maksimal poengsum}}{\text{antall riktige svaralternativ}}$ poeng. For hvert uriktige svaralternativ du velger får du $-\frac{\text{maksimal poengsum}}{\text{antall riktige svaralternativ}}$ poeng, med mindre oppgaven har stor overvekt av gale alternativer.

Uansett er det slik at det *ikke* er mulig å få en negativ poengsum for en oppgave; du får minimum 0 poeng per oppgave.

¹ Kopi av 1.1 - Funksjonsdeklarasjon

Hva må minimum presiseres ved funksjonsdeklarasjon?

Velg ett eller flere alternativer

- ☐ Parameterliste
- ☐ Verdi
- ☐ Funksjonsnavn
- ☐ Returtype

Maks poeng: 5

2 Kopi av 1.2 - Iteratorer

Hvilke (en eller flere) av følgende utsagn er korrekt?

Velg ett eller flere alternativer

- ☐ Medlemsfunksjonen `rend()` til `std::list` returnerer en iterator.
- ☐ Medlemsfunksjonene `rbegin()` og `end()` til `std::list` returnerer samme element.
- ☐ Medlemsfunksjonen `rbegin()` til en STL-beholder peker til det første elementet i beholderen.
- ☐ Medlemsfunksjonene `begin()` og `rend()` til `std::list` returnerer ikke same elem

Maks poeng: 5

3 Kopi av 1.3 - Operatoroverlasting

Hvilke (en eller flere) av alternativene er korrekt/viktig i følge læreboken?

Velg ett eller flere alternativer

- ☐ Bruk returtype og parametre som samsvarer med operatortypen og (resten av) språket.
- ☐ Deklarer operatoroverlastingen som en muterende medlemsfunksjon.
- ☐ Ikke overlast en operator med mindre nytten er stor.
- ☐ En overlastet operator må ha minst to brukerdefinerte typer som operander.

Maks poeng: 5

4 Kopi av 1.4 - Grunn kopi

Se på klassedeklarasjonen.

```

1  class Object {
2      int number;
3      Object * theOtherObject;
4
5  public:
6      Object(int number, Object * theOtherObject = nullptr) :
7          number{number}, theOtherObject{theOtherObject} {}
8  };

```

Hvilke (en eller flere) av kopikonstruktørene under gjør en grunn kopi av objektet *Object*?

Velg ett eller flere alternativer

- ☐ `Object(const Object &o) = default;`
- ☐ `Object(const Object &o) {}`
- ☐ `Object(const Object &o) { number = o.number;
theOtherObject = new Object(theOtherObject->number, theOtherObject->theOtherObject); }`
- ☐ `Object(const Object &o) = delete;`

Maks poeng: 5

5 Kopi av 1.5 - Templates

Se på klassedeklarasjonene under.

```
1  class Syllabus {
2      string name;
3      string content;
4
5      public:
6          Syllabus(string name, string content) : name {name}, content{content} {}
7
8          void changeContent(string newContent) {
9              content = newContent;
10         }
11
12         string getContent() {
13             return content;
14         }
15     };
```

Hvilke (en eller flere) av følgende utsagn er i følge læreboka vår korrekt om klassen *Syllabus* skal gjøres om til en templateklasse ved å bytte ut *string content* med *Template<T> content*?

Velg ett eller flere alternativer

- ☐ Vi må endre alle medlemsfunksjonene til klassen.
- ☐ Vi må skrive template over hver funksjonsdeklarasjon inne i klassen.
- ☐ Vi bør ha alle funksjonsdefinisjonene i samme fil som klassedeklarasjonen.
- ☐ Vi må ha funksjonsimplementasjonene utenfor klassedeklarasjonen.

Maks poeng: 5

6 Kopi av 1.6 - Klasser I

Se på klassedeklarasjonen under.

```
1 struct Flower {  
2     int numberOfPetals;  
3     string color;  
4     int weight;  
5     Flower(int numberOfPetals, string color, int weight = 3) : numberOfPetals{numberOfPetals},  
6         color{color}, weight{weight} {}  
7     Flower() : Flower{6, "yellow", 2} {}  
8 };
```

Hvilke (en eller flere) av følgende utsagn er korrekt?

Velg ett eller flere alternativer

- ☐ Klassen har en standard (default) konstruktør.
- ☐ Medlemsfunksjoner må lages for å kunne endre på medlemsvariablene.
- ☐ Klassen kan konstrueres med to inputparametere.
- ☐ Klassen har en delegerende konstruktør.

Maks poeng: 5

7 Kopi av 1.7 - Klasser II

Hvilke (en eller flere) av følgende par av klassedeklarasjoner gjør det mulig for brukeren av et *Child*-objekt å endre variabelen *number* direkte?

Velg ett eller flere alternativer

- ☐

```
class Parent {  
    public:  
        int number = 5;  
};
```
- ☐

```
class Child : public Parent {  
};
```
- ☐

```
class Parent {  
    protected:  
        int number = 5;  
};
```
- ☐

```
class Child : public Parent {  
};
```
- ☐

```
class Parent {  
    int number = 5;  
};
```
- ☐

```
class Child : public Parent {  
};
```
- ☐

```
struct Parent {  
    int number = 5;  
};
```
- ☐

```
struct Child : public Parent {  
};
```

Maks poeng: 5

8 Kopi av 1.8 - Kodeforståelse I

Se på klassedeklarasjonen under:

```

1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  enum class FishName{Nemo, Dory, Marlin, Wanda, Cleo, Larry, count};
6  enum class FishType{Anemonefish, Angelfish, Goldfish, Salmon, Sunfish, Surgeonfish, count};
7
8  class Fish {
9      static inline vector<Fish*> otherFish;
10     FishName name;
11     FishType type;
12
13 public:
14     bool operator==(const Fish& rhs) {
15         if(name == rhs.name) return true;
16         return false;
17     }
18
19     Fish() : name{FishName::Cleo}, type{FishType::Goldfish} {
20         bool fishFlag = true;
21         while(fishFlag) {
22             fishFlag = false;
23             for(auto fish : otherFish) {
24                 while(*this == *fish) {
25                     type = static_cast<FishType> (rand() % static_cast<int>(FishType::count));
26                     name = static_cast<FishName> (rand() % static_cast<int>(FishName::count));
27                     fishFlag = true;
28                 }
29             }
30         }
31         otherFish.push_back(this);
32     }
33 };

```

Hvilket (en eller flere) av følgende utsagn er korrekt?

Velg ett eller flere alternativer

- ☐ Vi må uansett ha en fisk som heter Cleo.
- ☐ Koden går i en uendelig løkke når man oppretter den 7. fisken.
- ☐ Vi kan ikke ha to gullfisker.
- ☐ Vi kan bare ha én fisk som heter Dory.

Maks poeng: 10

9 Kopi av 1.9 - Kodeforståelse II

Se på koden under.

```
1  #include <iostream>
2  using namespace std;
3
4  class Animal {
5      string name;
6  public:
7      Animal(string name) : name{name} {}
8  };
9
10 template<typename T>
11 void function(T one, T two) {
12     if(one == two) return;
13     else cout << ( one > two );
14 }
15
16 int main() {
17     Animal dog{};
18     Animal cat{};
19     function(dog, cat);
20     return 0;
21 }
```

Hvilke (en eller flere) av de følgende konseptene må vi implementere i klassen for at koden skal kjøre?

Velg ett eller flere alternativer

- ☐ << operator
- ☐ == operator
- ☐ > operator
- ☐ Move-konstruktør
- ☐ Standard (*default*) konstruktør
- ☐ Kopikonstruktør

Maks poeng: 5

10 Kopi av 1.10 - Program

NB! Legg merke til at denne oppgaven har to deloppgaver (**a** og **b**) som begge skal besvares.

Du har fått i oppgave å programmere sjakk. Kort fortalt består sjakk av 16 svarte og 16 hvite brikker på et 8x8-rutenett. Det finnes 6 ulike typer sjakkbrikker i hver farge, der alle typene ser forskjellige ut og kan bevege seg på forskjellige måter.

a) Du ønsker å lage en klasse *ChessPiece* for å representere sjakkbrikkene. Hvilke (en eller flere) av alternativene under kan man gjøre for å gjøre klassen abstrakt?

Velg ett eller flere alternativer

- ☐ Lage en "helt virtuell" (*pure virtual*) funksjon.
- ☐ Lage alle konstruktører abstrakte.
- ☐ Kun bruke virtuelle funksjoner.
- ☐ Slette kopikonstruktøren og tilordningsoperatoren.

b) Hvilken av disse (en eller flere) funksjonalitetene er det naturlig å ha med i den abstrakte *ChessPiece*-klassen?

Velg ett eller flere alternativer

- ☐ Logikken for hvordan brikken skal bevege seg.
- ☐ Grafikken til brikkene.
- ☐ Fargen på brikken.
- ☐ Posisjonen til brikken.

Maks poeng: 10

i Kopi av Informasjon om del 2

Bra jobbet! Nå kommer du til **del 2** av eksamen som er kortsvarsoppgaver. Denne delen inneholder **9 oppgaver** som til sammen gir en maksimal poengsum på **60 poeng** og teller ca. **20 %** av eksamen.

Informasjon om poenggivning: Hver oppgave gir maksimal poengsum på 5 eller 10 poeng avhengig av vanskelighetsgrad og arbeidsmengde. Du får poeng etter hvor presist svaret ditt er. Det vil si at det er mer uttelling for et kort og presist svar enn et langt og upresist svar.

¹¹ Kopi av 2.1 - Grunnleggende programmering I

Forklar kort hvilke elementer som er nødvendig for å definere en variabel.

Skriv ditt svar her

Maks poeng: 5

12 Kopi av 2.2 - Grunnleggende programmering II

Forklar kort hvilke elementer som er nødvendig for å definere en funksjon.

Skriv ditt svar her

Format

B


I


U


x_2


x^2


I_x













$\frac{1}{2}$


$\frac{1}{2}$

Ω





Σ



Words: 0

Maks poeng: 5

13 Kopi av 2.3 - Beholdere

Hva er hovedfordelen med lenket liste sammenlignet med tabeller/vector?

Skriv ditt svar her

Format

B


I


U


x_2


x^2


I_x













$\frac{1}{2} =$


$\frac{1}{2} =$

Ω





Σ



Words: 0

Maks poeng: 5

14 Kopi av 2.4 - Pekere

Forklar kort de største risikoene ved å bruke pekere.

Skriv ditt svar her

Format

B


I


U


x_2


x^2


I_x
























Σ



Words: 0

Maks poeng: 5

¹⁵ Kopi av 2.5 - Heltallstyper

Forklar kort hvorfor heltallstypene *char* og *int* har ulike intervaller som verdiene de kan inneholde må være innenfor.

Skriv ditt svar her

Maks poeng: 5

16 Kopi av 2.6 - Kodeforståelse I

Forklar med egne ord hva funksjonen under gjør.

```
1 char function(char var) {  
2     if (tolower(var) > 'f' - 1) return var;  
3     return function(var + 1);  
4 }
```

Skriv ditt svar her

Format

B


I


U


x_2


x^2


$\frac{1}{x}$













$\frac{1}{2}$


$\frac{3}{2}$

Ω





Σ



Words: 0

Maks poeng: 5

20/26

¹⁷ **Kopi av 2.7 - Kodeforståelse II**

Forklar med egne ord hva funksjonen *mysteryFunction* gjør.

```

1  class Node {
2      int nodeNumber;
3      int nodeValue;
4      Node* nextNode;
5  public:
6      Node(int nodeNumber = 0, int nodeValue = 0, Node* next=nullptr) :
7          nodeNumber{nodeNumber}, nodeValue{nodeValue}, nextNode{next} {}
8
9      Node* addNode() {
10         Node* oldNextNode = nextNode;
11         return nextNode = new Node(nodeNumber+1, 0, oldNextNode);
12     }
13
14     int value(){return nodeValue;}
15
16     ~Node() {
17         if(nextNode) delete nextNode;
18     }
19
20     Node* mysteryFunction(Node& inputNode) {
21         if(!nextNode) return this;
22         else if(nextNode != &inputNode) return nextNode->mysteryFunction(inputNode);
23         else {
24             delete nextNode;
25             nextNode = nullptr;
26             return this;
27         }
28     }
29 };

```

Skriv ditt svar her

Words: 0

Maks poeng: 10

18 Kopi av 2.8 - Feilsøking I

NB! Denne oppgaven har tre deloppgaver. Skill på deloppgavene i svarfeltet ved å først skrive henholdsvis **a)**, **b)** og **c)**.

Funksjonen *position* skal beregne posisjonen ut ifra startposisjonen, startfarten og akselerasjonen, gitt loven om konstant akselerasjon: $pos = pos_0 + vel_0 * time + \frac{1}{2} * accel * time^2$

```
1 double position(double initialPosition, double initialVelocity, int time, int acceleration) {
2     return initialPosition + initialVelocity*time + acceleration / 2 * time*time;
3 }
```

- a) Hva er problemet med koden?
- b) Hvilke to ulike måter kan man løse problemet på?
- c) Hvorfor gir koden likevel ut riktig resultat i noen tilfeller?

Skriv ditt svar her

Format

B

I

U

x_2

x^2

I_x

Ω

Σ

Words: 0

Maks poeng: 10

¹⁹ Kopi av 2.9 - Feilsøking II

Se på koden under.

```
1  #include <iostream>
2  using namespace std;
3
4  int sumAndSquare(int a, int b) {
5      return sum(a,b)*sum(a,b);
6  }
7
8  int sum(int a, int b) {
9      return a+b;
10 }
11
12 int main(){
13     cout << sumAndSquare(4,1);
14     return 0;
15 }
```

Koden kompilerer ikke. På hvilke *to* ulike måter kan problemet i koden enkelt løses?

Skriv ditt svar her

Maks poeng: 10

i Kopi av Informasjon del 3

Bra jobba! Nå kommer du til **del 3** av eksamen som er programmer-selv-oppgaver. Denne delen inneholder **18 oppgaver** som til sammen gir en maksimal poengsum på **ca 200 poeng** og teller **ca. 60 %** av eksamen.

VIKTIG:

Zip-filen du skal laste ned inneholder kompilerbare (og kjørbare) .cpp- og .h-filer med forhåndskodede deler og en full beskrivelse av oppgavene i del 3 som en PDF-fil. Etter å ha lastet ned zip-filen står du fritt til å bruke et utviklingsmiljø etter eget valg (for eksempel VS Code) for å jobbe med oppgavene.

For å få bestått på denne eksamen er det **HELT AVGJØRENDE AT DU LASTER OPP ZIP-FILEN**. Etter eksamensslutt (13:00) har du 30 minutter til rådighet til dette

De korte svarene i Inspera (del 1 og 2) lagres automatisk hvert 15. sekund, helt til eksamenstiden er slutt. Og zip-filen (i del 3) kan også endres / lastes opp flere ganger. Så hvis du vil gjøre noen endringer etter at du har lastet opp filen, kan du bare endre koden, zippe den sammen på nytt, og laste opp filen igjen. Når prøvetiden er over, vil siste versjon av alt du har skrevet eller lastet opp i Inspera automatisk bli sendt inn som ditt gjeldende svar, så **sørg for at du laster opp minst en gang halvveis, og en gang før tiden går ut.**

På neste side kan du laste ned .zip-filen, og senere laste opp den nye .zip-filen med din egen kode inkludert. Husk at PDF-dokumentet med alle oppgavene er inkludert i zip-filen du laster ned.

20 Kopi av Nedlasting/opplasting av kode

LAST NED\

[tdt4102_summer_2022_exam-handout](#)

LAST OPP

Last opp all den komplette koden som en .zip-fil. Ikke endre den opprinnelige mappestrukturen. For å få bestå prøven er det **HELT AVGJØRENDE AT DU LASTER OPP DEN NYE ZIP-FILEN DIN KORREKT**, minst en gang i løpet av de 4 timene du har til rådighet.

Det er mulig å oppdatere både enkeltsvarene og filopplastningen **flere ganger**, i tilfelle du retter på noe etter første innlevering.


Prøv å last opp en oppdatert zip-fil minst en gang ekstra, midt i prøvetiden, for å se at du klarer det, og for å finne ut hvor lang tid du bruker på det.

Last opp zip-filen med besvarelsen din her. Alt i én zip-fil. **Merk at filen må være av typen .zip (andre format som 7z, rar etc godtas ikke).**



Last opp filen her. Maks én fil.

Alle filtyper er tillatt. Maksimal filstørrelse er **50 GB**.

 Velg fil for opplasting

Maks poeng: 180

Question 25

Attached



Part III: WeatherForecast

Maksimal score for del 3 er 190 poeng.

3.1 Info

Bra jobba! Nå kommer du til del 3 av eksamen som er programmer-selv-oppgaver. Denne delen inneholder 18 oppgaver som til sammen gir en maksimal poengsum rundt 200 poeng og teller ca. 60% av eksamen. VIKTIG:

Zip-filen du skal laste ned inneholder kompilerbare (og kjørbare) .cpp- og .h-filer med forhåndskodede deler og en full beskrivelse av oppgavene i del 3 som en PDF-fil. Etter å ha lastet ned zip-filen står du fritt til å bruke et utviklingsmiljø etter eget valg (for eksempel VS Code) for å jobbe med oppgavene.

For å få bestått på denne eksamen er det HELT AVGJØRENDE AT DU LASTER OPP ZIP-FILEN. Etter eksamensslutt (13:00) har du 30 minutter til rådighet til dette

De korte svarene i Inspira (del 1 og 2) lagres automatisk hvert 15. sekund, helt til eksamenstiden er slutt. Og zip-filen (i del 3) kan også endres / lastes opp flere ganger. Så hvis du vil gjøre noen endringer etter at du har lastet opp filen, kan du bare endre koden, zippe den sammen på nytt, og laste opp filen igjen. Når prøvetiden er over, vil siste versjon av alt du har skrevet eller lastet opp i Inspira automatisk bli sendt inn som ditt gjeldende svar, så sørg for at du laster opp minst en gang halvveis, og en gang før tiden går ut.

På neste side i Inspira kan du laste ned .zip-filen, og senere laste opp den nye .zip-filen med din egen kode inkludert. Husk at PDF-dokumentet med alle oppgavene er inkludert i zip-filen du laster ned.

WeatherForecast er en applikasjon som gjengir en ukes vær-prognose for en by. Stilen på gjengivelsen er i tråd med hvordan det vanligvis gjøres av værvarslingsapper. I den utleverte zip-filen, vil du finne et kjørbart applikasjonsskjelett, men som mangler en del funksjonalitet. Gjennom en rekke små oppgaver veileder vi deg mot å implementere alle de manglende brikkene for å lage applikasjonen fullt funksjonell. Etter at alle oppgavene i denne teksten er løst, vises sluttresultatet som i Figur 1. Når du kjører den utleverte koden før du begynner å gjøre noen endringer, vil du bli møtt av vinduet som vist i Figur 2.

I det følgende vil vi introdusere applikasjonen, dens funksjonalitet og filformatet som brukes til å lagre den grafiske representasjonen av værtyper, filene som inneholder stedsdata og filene som inneholder værmeldingsdata.

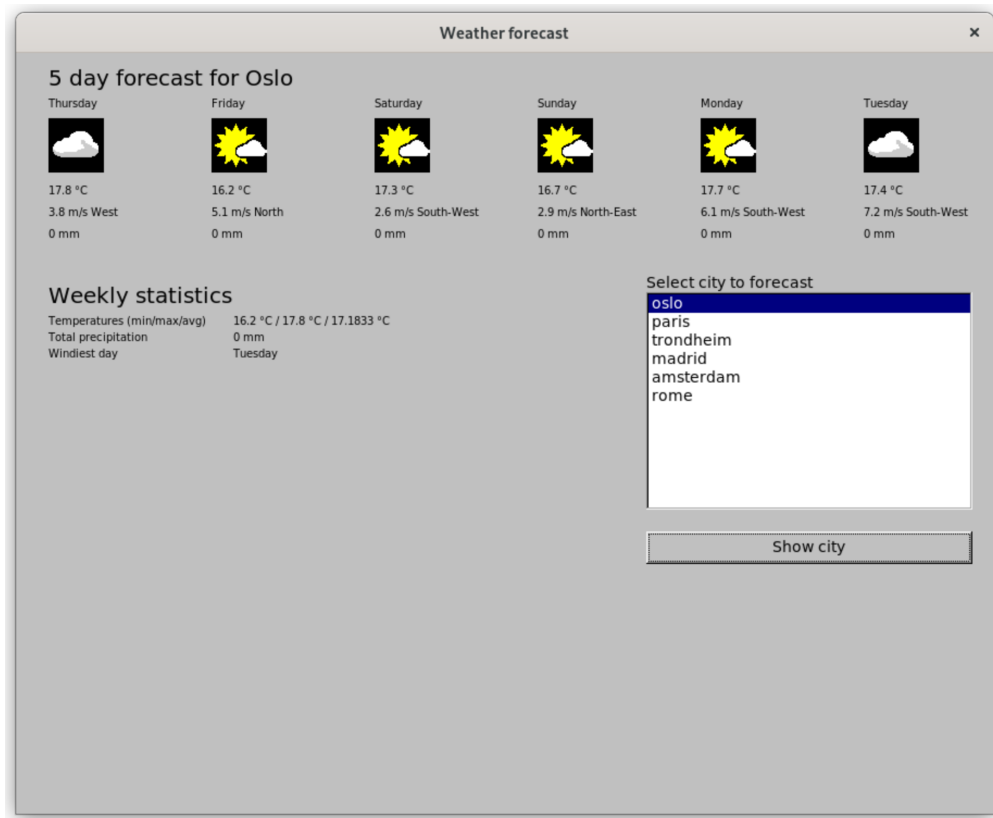
3.2 Application overview

Applikasjonen er implementert i flere klasser som forholder seg til hverandre på en hierarkisk måte. Hierarkiet er avbildet i Figur 3.

Toppnivåklassen til applikasjonen er `Application`. Denne klassen er ansvarlig for å sette opp GUI og instansiere `Forecast`-klassen som leser en CSV-fil som inneholder værmeldingen for en by og gjengir den innleste prognosen. CSV-filene som inneholder prognosene er plassert i katalogen `forecasts`. En prognose som vises av applikasjonen inneholder to ting (Figur 1): Forventet vær i 5 dager, og statistikk for forventet vær som gjennomsnitt- og maksimumstemperaturer.

Statistikkgenerering og gjengivelse skjer innenfor `Forecast`-klassen og implementeres i oppgavene F3-F5. En dag i prognosen er representert av `Day`-klassen. `Forecast`-klassen instansierer fem `Day`-klasser, en for hver dag, med været for den dagen som lest fra CSV-prognosefilen. CSV-prognosefilene finnes i katalogen `forecasts`. Vi ber deg ikke skrive kode for å analysere en CSV-fil eller finne listen over tilgjengelige prognoser siden dette allerede er implementert i den utleverte koden.

Til slutt, for å vise et ikon som representerer været (f.eks. en sky eller en sol for henholdsvis skyet og solfylt vær) instansierer hver `Day`-klasse en `ImageRenderer`-klasse for å laste og vise bildefilen med riktig ikon. Værikonbildefilene finnes i katalogen `symbols` i utdelt zip-fil. Vi ber deg implementere funksjonene for lesing av bildefilene som inneholder værsymboler i oppgavene R1-R3 og vi gir en detaljert beskrivelse av bildeformatet i innledningen til disse oppgavene.



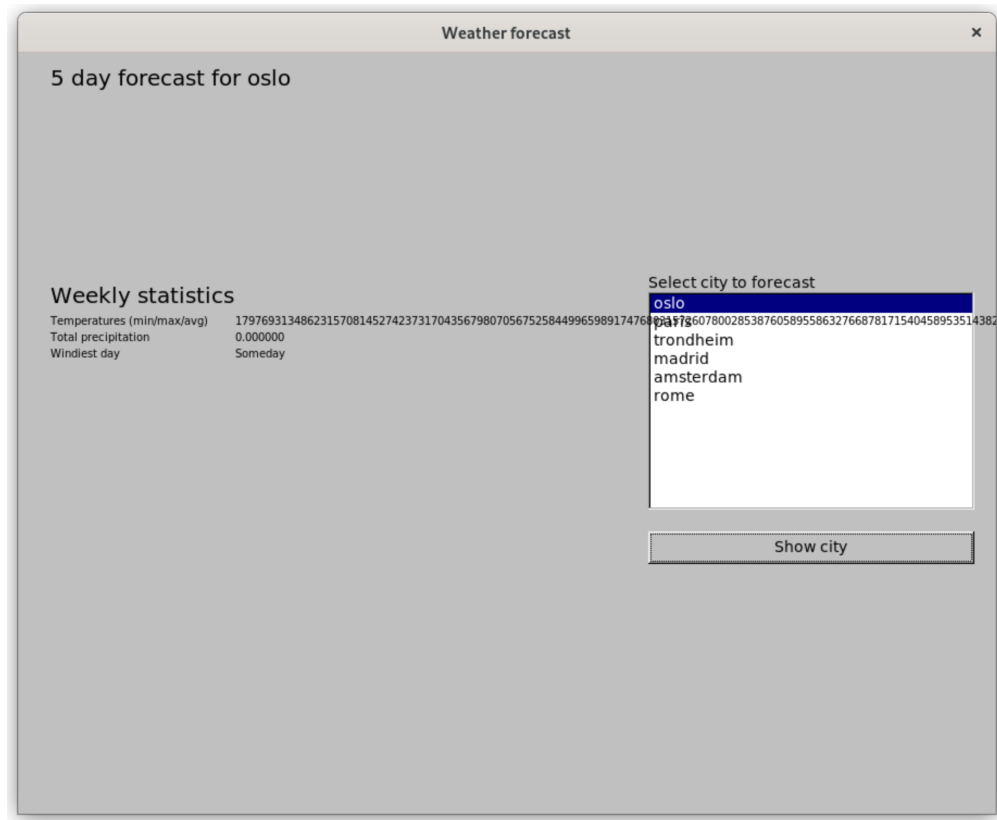
Figur 1: The complete application after selecting "Oslo" and clicking "Show city".

Hvordan besvare del 3?

Hver oppgave i del 3 har en tilhørende unik kode for å gjøre det lettere å finne frem til hvor du skal skrive svaret. Koden er på formatet <tegn><siffer> (TS), eksempelvis F1, F2 og R1. En oversikt over koblingen mellom bokstav og fil er gitt i tabell 1. I forecast.cpp, og de andre filene du skal løse oppgaver i, vil du for hver oppgave finne to kommentarer som definerer henholdsvis begynnelsen og slutten av koden du skal føre inn. Kommentarene er på formatet: // BEGIN: TS og // END: TS.

Det er veldig viktig at alle svarene dine er skrevet mellom slike kommentar-par, for å støtte sensurmekanismen vår. Hvis det allerede er skrevet noen kode *mellom* BEGIN- og END-kommentarene i filene du har fått utdelt, så kan, og ofte bør, du erstatte den koden med din egen implementasjon med mindre annet er spesifisert i oppgavebeskrivelsen. All kode som står *utenfor* BEGIN- og END-kommentarene SKAL dere la stå.

For eksempel, for oppgave G1 ser du følgende kode i utdelte robot_grid.cpp



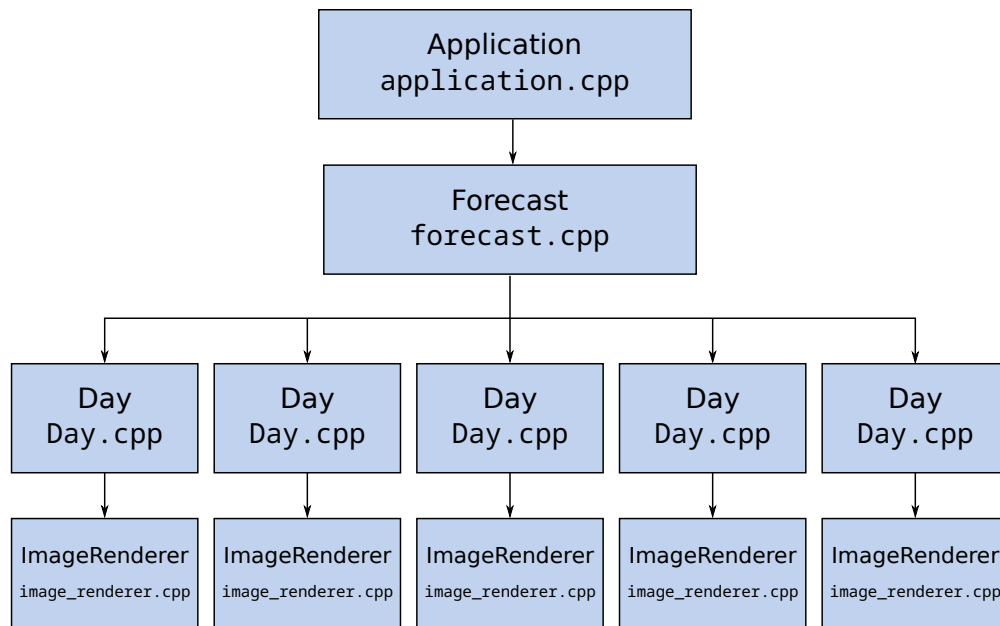
Figur 2: The handout application after selecting “Oslo” and clicking “Show city”.

```

1  int Forecast::get_day_placement(int day)
2  {
3      // BEGIN: F1
4      //
5      // Write your answer to assignment F1 here, between the // BEGIN: F1
6      // and // END: F1 comments. You should remove any code that is
7      // already there and replace it with your own.
8
9      return 0;
10
11     // END: F1
12 }

```

Etter at du har implementert din løsning, bør du ende opp med følgende i stedet:



Figur 3: Klassediagrammet til WeatherForecast-applikasjonen.

```

1  int Forecast::get_day_placement(int day)
2  {
3      // BEGIN: F1
4      //
5      // Write your answer to assignment F1 here, between the // BEGIN: F1
6      // and // END: F1 comments. You should remove any code that is
7      // already there and replace it with your own.
8
9      /* Din kode her */
10
11     // END: F1
12 }

```

Merk at BEGIN- og END-kommentarene **IKKE** skal fjernes.

Til slutt, hvis du synes noen av oppgavene er uklare, oppgi hvordan du tolker dem og de antagelsene du må gjøre som kommentarer i den koden du sender inn.

Før du starter må du sjekke at den (umodifiserte) utdelte koden kjører uten problemer. Du skal se det samme vinduet som i Figur 2. Når du har sjekket at alt fungerer som det skal er du klar til å starte programmering av svarene dine.

3.2.1 Hvor i koden finner du oppgavene?

Hver oppgave har en unik kode bestående av en bokstav etterfulgt av et tall. Bokstaven indikerer hvilken fil oppgaven skal løses i. Se Tabell 1 for en oversikt over bokstaver og filnavn. Eksempelvis finner du oppgave F1 i filen `forecast.cpp`

Tabell 1: Oversikt over koblingen mellom oppgave-bokstav og filnavn.

Bokstav	Fil
F	forecast.cpp
D	day.cpp
R	image_renderer.cpp
U	util.cpp

Oppgavene

Formatting values (40 points)

Enhver værmelding må inneholde pent formaterte numeriske parametere som beskriver forventet vær som temperaturer og vindhastighet. I denne delen av oppgaven skriver vi funksjonene for å formatere disse verdiene.

1. (10 points) **U1: Weekday numbers to names**

Skriv en funksjon som tilordner et heltall mellom 0 og 6 til et ukedagsnavn. Den første dagen i uken (verdi 0) er mandag. Hvis verdien av `weekday_num`-parameteren er utenfor dette området, bør funksjonen gi et unntak.

2. (10 points) **U2: Friendly wind directions**

Skriv en funksjon som samler kompassvinklene (0 til 360 grader) til vindretningene til 8 forskjellige retninger (f.eks. Nord, Nord-Vest, osv.). Hvis verdien av `heading`-argumentet er utenfor det gyldige området, skal funksjonen kaste et unntak.

: Funksjonen skal benytte følgende regler for overgang fra vinkel til retning:

- 337.5 - 22.5: North
- 292.5 - 337.5: North-West
- 247.5 - 292.5: West
- 247.5 - 202.5: South-West
- 157.5 - 202.5: South
- 112.5 - 157.5: South-East
- 67.5 - 112.5: East
- 22.5 - 67.5: North-East

3. (10 points) **U3: Get unit as string**

Betrakt enum `Unit` definert i `util.h` som følger

```
enum Unit {  
    UNIT_MS, // Meters per second, m/s  
    UNIT_MM, // Millimeteres, mm  
    UNIT_DC // Degrees celcius, °C  
};
```

Skriv en funksjon som gitt en `Unit` enum-verdi returnerer den tilsvarende strengrepresentasjonen. For eksempel, `get_unit(UNIT_MM) == "mm"`.

4. (10 points) **U4: Format units**

Skriv en funksjon som formaterer numeriske værparametere for presentasjon ved å avrunde flyttall til en spesifisert antall desimaler og legge til en enhet. Funksjonen tar tre parametere: `value` er verdien som

skal formateres, `decimals` er antallet av desimaler som verdien skal avrundes til og `unit` er enheten som skal legges til den formaterte strengen. Bruk funksjonen `get_unit` implementert i U3 for å konvertere `unit`-verdien til en streng.

For eksempel skal funksjonen returnere verdier som gjør følgende sammenligninger er sanne (forutsatt at `get_unit`-funksjonen er korrekt implementert): `format_value(3.94, 1, UNIT_MM) == "3.9 mm"` og `format_value(3.587, 2, UNIT_MS) == "3.59 m/s"`

5. (10 points) **D1: Rendering a day**

Nå bruker vi hjelpefunksjonene definert tidligere for å render en dag i værmeldingen. For å legge til tekst i prognosen, er funksjonen `draw_text(Point pos, string value)` gitt. Der det er aktuelt, avrund flyttall til én desimal og sørg for at riktig enhet er lagt til. For å plassere teksten, bruk variablene `xpos` og `ypos` som holder koordinatene til øvre høyre hjørne av en dag. Listen nedenfor forteller deg hvilken informasjon som skal være tilstede i prognosen, tekstens relative y-offset og variabelen til `Day`-klassen hvor informasjonen finnes. Bruk variablene `xpos` og `ypos` sammen med de relative y-offsetene for å konstruere `Point`-parameteren `pos` til `draw_text`-funksjonen.

- Ukedagen. Y-posisjon: 20, variabel: `day`
- Temperaturen. Y-posisjon: 100, variabel `temp`
- Vindhastigheten og -retningen: Y-posisjon: 120, variabler: `wind_speed`, `wind_dir`
- Nedbørmengde: Y-posisjon: 140, variabel: `precip_amount`

Showing the forecast (50 points)

6. (10 points) **F1: Forecast layout**

Skriv en funksjon som returnerer den horisontale pikselforskyvningen for hver dag i prognosen. Hver dag er 150 piksler bred. Hvis verdien av parameteren `day` er ugyldig, dvs. den er utenfor området $0 \leq \text{day} \leq 6$, skal funksjonen kaste et unntak.

7. (10 points) **F2: Capitalize a string**

Skriv en funksjon som gjør om det første tegnet i en streng til en stor bokstav slik at `capitalize(tt-rondheim)` == `Trondheim`". Denne funksjonen trenger bare å endre det første tegnet i strengen. Husk at funksjonen `toupper` kan brukes til å endre en `char`-verdi til store bokstaver.

8. (10 points) **F3: Precipitation statistics**

Skriv en funksjon som beregner den totale nedbørmengden forventet på tvers av alle dagene som er dekket i prognosen og legg resultatet i klassevariablen `total_precip`. Når denne funksjonen blir kalt er `total_precip` 0.

For å gjøre dette, iterer over `days`-vektoren i den nåværende `Forecast` klasseinstansen. Husk at vektoren `days` er definert som `vector<unique_ptr<Days>> days` i `forecast.h`. For å få mengden av nedbør som er forventet for en dag, bruk den offentlige `precip_amount`-variabelen i `Days` klasseinstansen.

9. (10 points) **F4: Temperature statistics**

Skriv en funksjon som beregner maksimums-, minimums- og gjennomsnittstemperaturer som forventes for dagene som er inkludert i prognosen, og lagre de resulterende verdiene i henholdsvis klassemedlemsvariablene `max_temp`, `min_temp` og `avg_temp`. Når denne funksjonen blir kalt er `max_temp` og `min_temp` satt til minimums- og maksimumsverdiene som støttes av C++ `double`-typen henholdsvis.

Som i de forrige oppgavene, iterer over `days`-vektoren og bruk `temp`-medlemsvariabelen til `Day`-klasseinstansen for å få temperaturen som forventes for en dag.

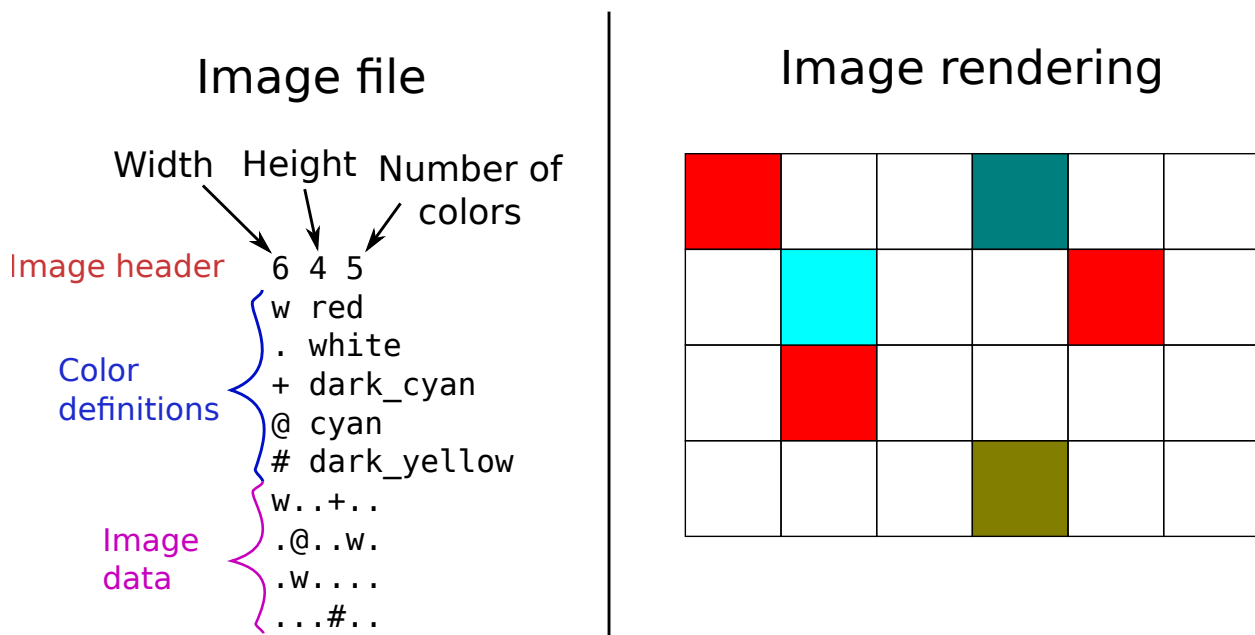
10. (10 points) **F5: Wind statistics**

Skriv en funksjon som finner den mest vindfulle dagen i uken, og legg resultatet i klassemedlemsvariabelen `windiest_day`. Husk at vi er interessert i å identifisere *dagnummeret* (dvs. et heltall mellom 0-6) og ikke den faktiske vindhastigheten.

Vindhastigheten for en dag kan finnes i den offentlige `wind_speed`-medlemsvariabelen i klassen `Day`.

Gjengivelse av bilder (90 poeng)

I denne siste delen av oppgaven implementerer vi funksjonene for lesing og gjengivelse av bildeformatet som brukes for å lagre værsymbolene i værmeldingen. Før vi starter på oppgavene beskriver vi bildeformatet som de påfølgende oppgavene ber deg implementere.¹



Figur 4: Et eksempel på en enkel bildefil (venstre) og dens gjengivelse (høyre).

Bildefilformatet er delt inn i tre deler: Image header, Color definitions og Image data. Delene er definert som følger:

Image header En linje med tre tall atskilt med mellomrom. De to første tallene gir bredden og høyden på bildet og det siste tallet gir antall farger i bildet.

Color definitions Denne delen tilordner en *fargereferanse* (color reference) til en *fargenavn* (color name). Fargereferansen er et enkelt tegn representert av C++-typen `char` og fargenavnet er en streng. Tegnet kan være hva som helst bortsett fra et mellomrom, og hver linje inneholder en enkelt definisjon. Antall linjer, og dermed antall definisjoner, er gitt av det siste tallet i bildeoverskriften. Derfor, hvis det siste tallet i bildeoverskriften er 5, som i Figur 4, bør du forvente at de 5 linjene etter overskriften er fargedefinisjon.

Image data Den siste delen av bildet inneholder de faktiske bildedataene. Hver linje i bildet tilsvarer en linje i bildefilen, og derfor, hvis et bilde har 6x4 piksler (som i Figur 4), inneholder bildefor-matdelen 4 linjer med 6 tegn hver. Hvert tegn tilsvarer en av de definerte fargene. For eksempel

¹For historisk referanse er filformatet som brukes i denne oppgaven en liten forenkling av XPM2-filformatet som ble brukt til å lagre bilder i tidlige versjoner av X11 window system.

er de to første horisontale pikslene i bildet i Figur 4 gitt av tegnene "w.". Fra fargedefinisjonsdelen ser vi at w er tilordnet til fargen red og . er tilordnet fargen white. Derfor er de to første pikslene i bildet henholdsvis røde og hvite. For enkelhets skyld gir vi kartet `color_map` definert i klassen `ImageRenderer` som du kan bruke til å konvertere fargenavnene som brukes i bildefilen til fargeverdiene som brukes i tegnefunksjonene.

Intern representasjon. Når et bilde lastes inn fra en bildefil, transformeres det til den interne representasjonen som brukes i `ImageRenderer`-klassen. Den interne representasjonen består av to datatyper definert i `image_renderer.h` som følger:

```
1 map<char, Graph_lib::Color::Color_type> colors;  
2 vector<unique_ptr<vector<Graph_lib::Color::Color_type>>> image;
```

Map-et `colors` inneholder tilordningene mellom *fargereferanser* og fargekodene som brukes i `Graph_lib`. Vi implementerer en hjelpefunksjon for å legge til oppføringer i dette kartet i oppgave R2 og funksjonen for å lese fargekartlegginger fra bildefilen i oppgave R6. Vektoren `image` inneholder de faktiske bilde-dataene. Merk at det er definert som en vektor av vektorer av farger. Den ytre vektoren inneholder én vektor per rad i bildet og de indre vektorene inneholder fargene til pikslene som utgjør hver rad. Så, etter at bildet i Figur 4 er lest, inneholder `image`-vektoren 4 vektorer (tilsvarer høyden på bildet) som hver inneholder 6 farger (tilsvarer bredden på bildet)

De følgende tre oppgavene ber deg implementere parsere for de tre delene av bildeformatet. Deretter implementerer vi funksjonen for å gjengi den interne representasjonen av et bilde på skjermen.

11. (10 points) **R1: Color lookup**

Skriv en funksjon som returnerer den tilsvarende `Graph_lib` fargetypen som tilsvarende fargenavnet som er gitt i parameteren `color`. For å gjøre dette, kan du bruke map-et `color_map` definert i `image_renderer.h`. Hvis parameteren `color` refererer til en ugyldig farge, skal funksjonen kaste et unntak.

12. (10 points) **R2: Add color references**

Skriv en funksjon som gitt en fargereferanse (parameter `ref`) og et fargenavn (parameter `color_name`) legger til et element i `colors`-vektoren. Bruk funksjonen `get_color_value` for å slå opp fargenavnene.

13. (10 points) **R3: Add image rows**

Skriv en funksjon som legger til en ny vektor til `image`-vektoren. Hensikten er at denne funksjonen kalles for hver ny rad med piksler som blir funnet ved lesing av bildefilen. Husk at den nye vektoren må instansieres som en `unique_pointer`.

14. (10 points) **R4: Add image pixels**

Skriv en funksjon som legger en piksel til bildet. Siden bilde-dataene leses linje for linje, skal pikselen alltid legges til den sist tilføyde vektoren i `image`-vektoren.

15. (10 points) **R5: Reading the header**

Fullfør funksjonen for lesing av image header. Det meste av det harde arbeidet er allerede gjort for deg, men delen for å lese de faktiske verdiene inn i variablene `width`, `height` og `ncolors` mangler. Din oppgave er å legge til dette.

16. (20 points) **R6: Reading the colors**

Implementer funksjonen for å lese fargedefinisjonene til en bilde. Et eksempel på fargedefinisjoner er gitt i Figur 4

Bildefilstrømmen er gitt i parameteren `image_file_stream`, og antall farger som skal leses sendes i parameteren `ncolor`. Når funksjonen kalles, peker `image_file_stream` til begynnelsen av fargedelen av filen (forutsatt at du implementerte forrige oppgave riktig). For hver fargedefinisjon som leses, kall funksjonen `add_color` for å legge til fargen til den interne representasjonen. Funksjonen `add_color` tar to parametre: en `char` som inneholder fargereferansen og en `string` som inneholder fargenavnet.

17. (20 points) **R7: Reading the pixels**

Implementer funksjonen for å lese pikseldelen av bildet. Fra `width` og `height`-variablene initialisert i `read_image`-funksjonen, får du dimensjonene til bildet og dermed antall linjer med bildedata som skal leses og antall tegn hver linje har. For hver linje du leser, gjør et kall til `add_line`-funksjonen uten argumenter. For hver piksel du leser, kall `add_pixel`-funksjonen med navnet på fargen på pikselen. Tilordningene mellom fargereferanser og verdier lagres i map-et `colors`. Dermed kan du bruke fargereferensen du leser fra bildefilen for å utføre et oppslag i `colors` map-et for å få det nødvendige argumentet til `add_pixel`-funksjonen.