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 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 i Inspera 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.

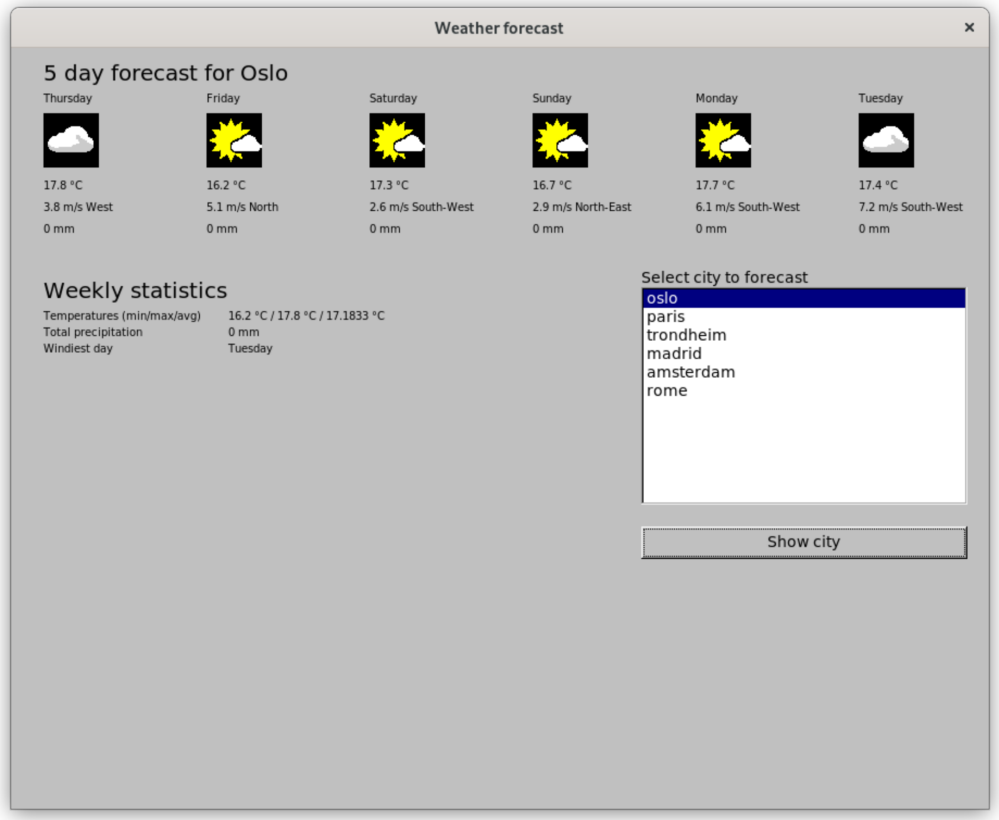
3.2 Application overview

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 gjennomsnitts- 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.



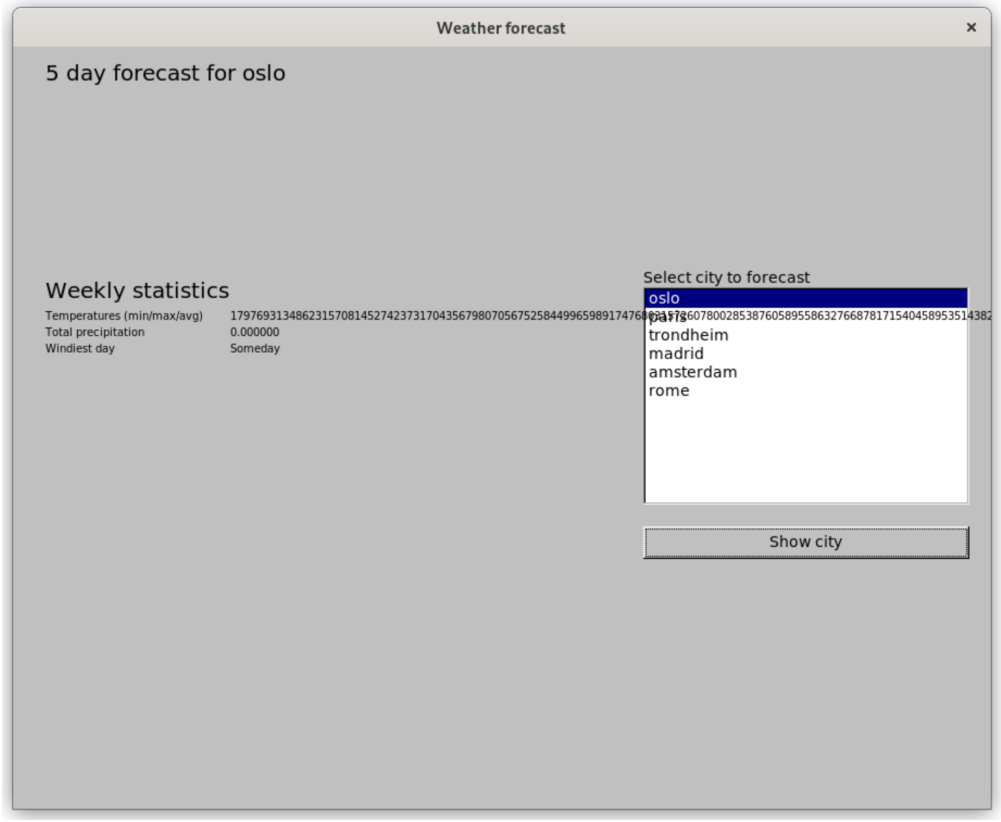
Figur 1: The complete application after selecting "Oslo" and clicking "Show city".

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 sensurmekanikken 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`



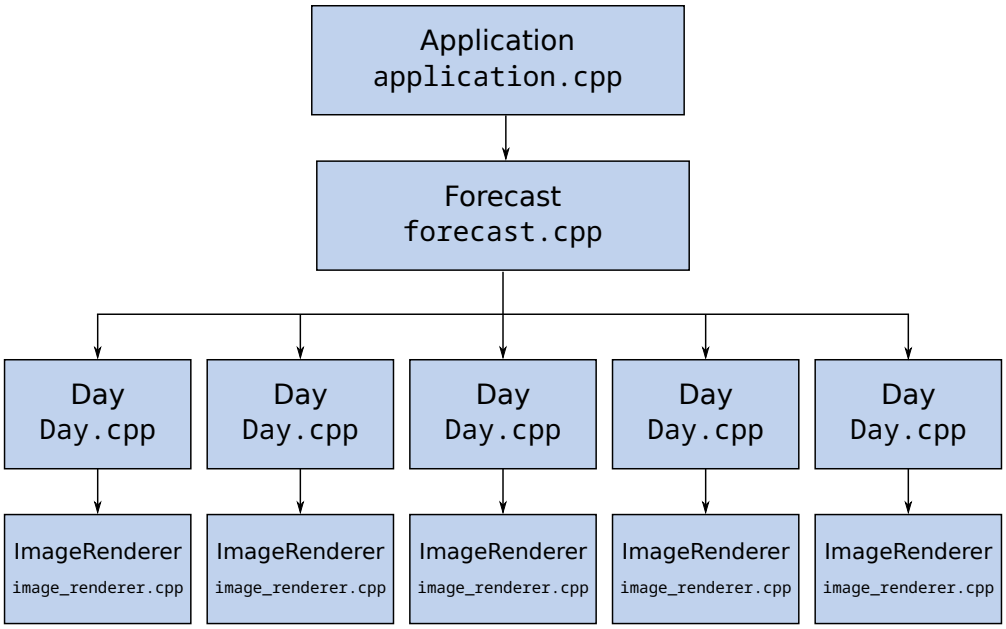
Figur 2: The handout application after selecting "Oslo" and clicking "Show city".

```
int Forecast::get_day_placement(int day)
{
  // BEGIN: F1
  //
  // Write your answer to assignment F1 here, between the // BEGIN: F1
  // and // END: F1 comments. You should remove any code that is
  // already there and replace it with your own.

  return 0;

  // END: F1
}
```

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



Figur 3: Klassediagrammet til WeatherForecast-applikasjonen.

```
int Forecast::get_day_placement(int day)
{
  // BEGIN: F1
  //
  // Write your answer to assignment F1 here, between the // BEGIN: F1
  // and // END: F1 comments. You should remove any code that is
  // already there and replace it with your own.

  /* Din kode her */

  // END: F1
}
```

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 an 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 rendere 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: `vind_speed`, `wind_dir`
   - Nedbørsmengde: 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.

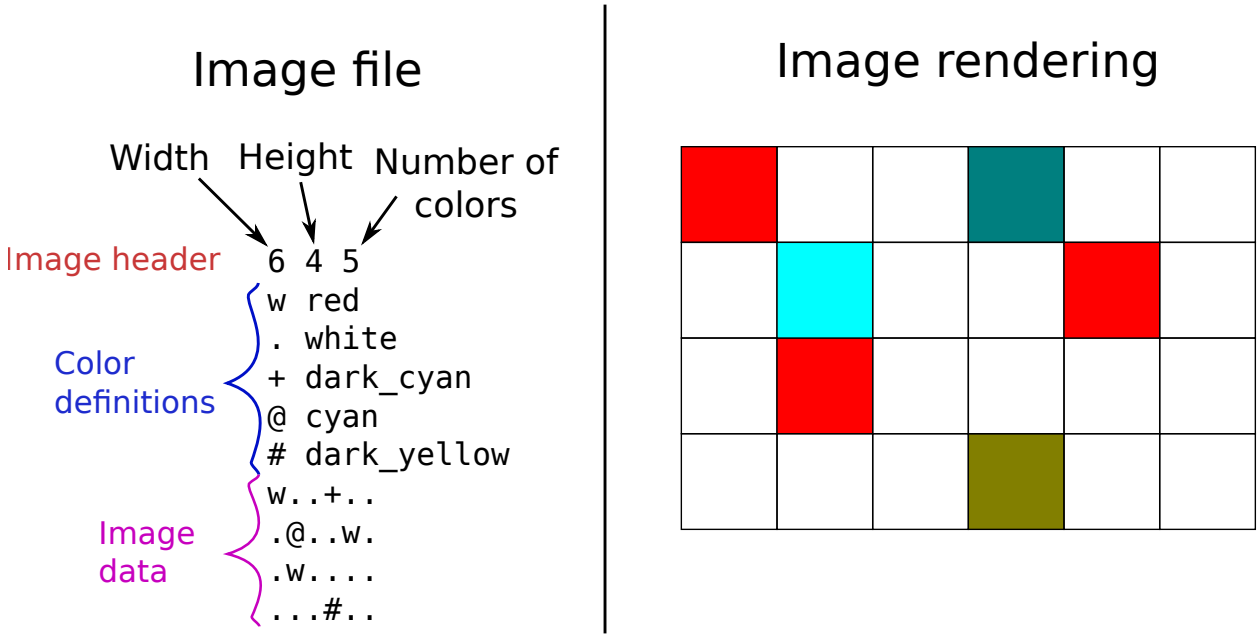
10. (10 points) **F5: Wind statistics**

    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 *fargereferense* (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 bildeformatdelen 4 linjer med 6 tegn hver. Hvert tegn tilsvarer en av de definerte fargene. For eksempel

[1] For 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 bildedataene. 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 tilsvarer 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 bildedataene leses linje for linje, skal pikselen altid 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 parametere: 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.