Part III: WeatherForecast

Maksimal skår for del 3 er 190 poeng.

3.1 Info

Bra jobba! No kjem du til del 3 av eksamen som er program- sjølv- oppgåver. Denne delen inneheld 18 oppgåver som til saman gir eit maksimal poengsum rundt 200 poeng og tel ca. 60% av eksamen.
VIKTIG:

Zip-filen du skal lasta ned inneheld .cpp- og .h-filer som allereie kompilerer (og køyrer) med førehandskodede deler og ei full skildring av oppgåvene i del 3 som ein PDF-fil. Etter å ha lasta ned zip-filen står du fritt til å bruka eit utviklingsmiljø etter eige val (til dømes VS Code) for å jobba med oppgåvene.

For å få bestått på denne eksamenen er det HEILT AVGJERANDE AT DU LASTAR OPP ZIP-FILEN. Etter eksamensslutt (13:00) har du 30 minutt til rådvelde til dette

Dei korte svara i Inspera (del 1 og 2) blir lagra automatisk kvart 15. sekund, heilt til eksamenstida er slutt. Og zip-filen (i del 3) kan òg endrast / blir lasta opp fleire gangar. Så viss du vil gjera nokre endringar etter at du har lasta opp fila, kan du berre endra koden, zippe han saman på nytt, og lasta opp fila igjen. Når prøvetida er over, vil siste versjon av alt du har skrive eller lasta opp i Inspera automatisk bli sendt inn som ditt gjeldande svar, så sørg for at du lastar opp minst ein gang halvvegs, og ein gang før tida går ut.

På neste side i Inspera kan du lasta ned .zip-filen, og seinare lasta opp den nye .zip-filen med din eigen kode inkludert. Hugs at PDF-dokumentet med alle oppgåvene er inkluderte i zip-filen du lastar ned.

WeatherForecast er ein applikasjon som attgir ein vekes vêr-prognose for ein by. Stilen på attgjevinga er i tråd med korleis det vanlegvis blir gjort av vêrvarslingssappar. I den utleverte zip-filen, vil du finna eit applikasjonsskjelett som kompilerer, men som manglar ein del funksjonalitet. Gjennom ei rekke små oppgåver rettleier vi deg mot å implementera alle dei manglande brikkene for å laga applikasjonen fullt ut funksjonell. Etter at alle oppgåvene i denne teksten er løyst, blir sluttresultatet som vist i Figur 1. Når du køyrer den utleverte koden før du byrjar å gjera nokre endringar, vil du bli møtt av vindauget som vist i Figur 2.

I det følgjande vil vi introdusera applikasjonen, dens funksjonar og filformatet som blir brukt til å lagra den grafiske representasjonen av vêrtypar, filene som inneheld stadsdata og filene som inneheld vêrmeldingsdata.

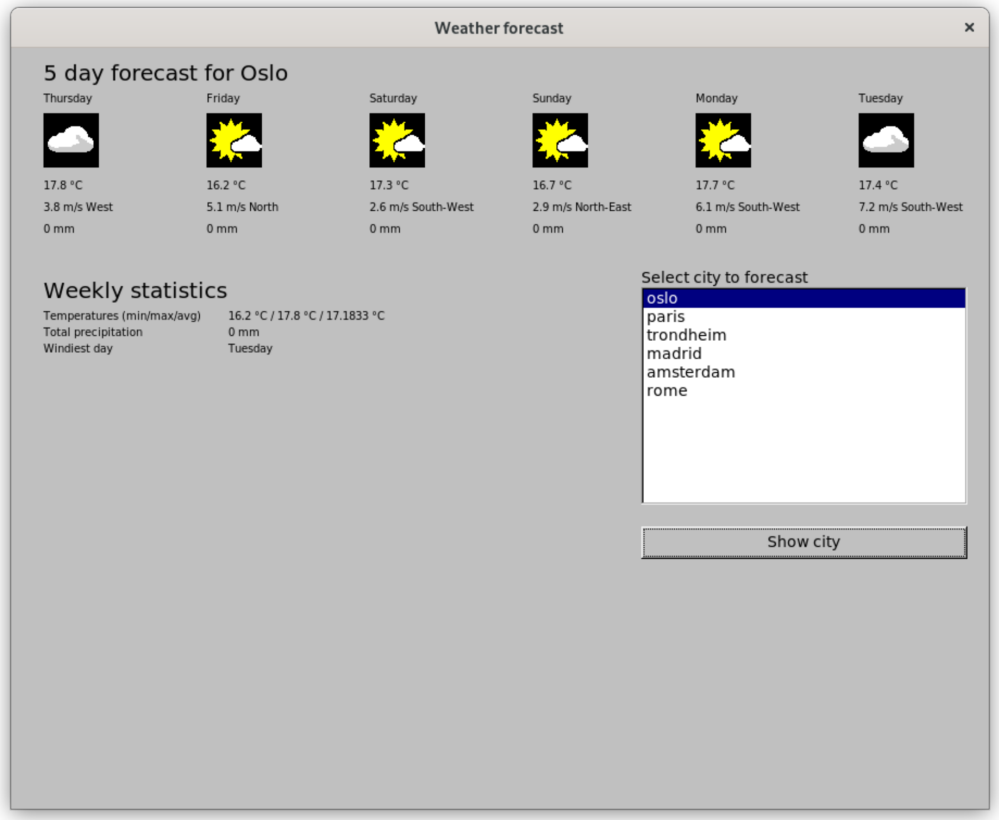
3.2 Application overview

Applikasjonen er implementert i fleire klassar som forheld seg til kvarandre på ein hierarkisk måte. Hierarkiet er avbilda i Figur 3.

Toppnivåklassen til applikasjonen er `Application`. Denne klassen er ansvarleg for å setja opp GUI og instansiere `Forecast`-klassen som les ein CSV-fil som inneheld vêrmeldinga for ein by og attgir den innlesne prognosen. CSV-filene som inneheld prognosane er plasserte i katalogen `forecasts`. Ein prognose som blir vist av applikasjonen inneheld to ting (Figur 1): Forventa vêr i 5 dagar, og statistikk for forventa vêr som gjennomsnitts- og maksimumstemperaturar.

Statistikkgenerering og attgjeving skjer innanfor `Forecast`-klassen og blir implementert i oppgåvene F3-F5. Ein dag i prognosen er representert av `Day`-klassen. `Forecast`-klassen instansierer fem `Day`-klassar, ein for kvar dag, med vêret for den dagen som lese frå CSV-prognosefilen. CSV-prognosefilene finst i katalogen `forecasts`. Vi ber deg ikkje skriva kode for å analysera ein CSV-fil eller finna lista over tilgjengelege prognosar sidan dette allereie er implementert i den utleverte koden.

Til slutt, for å visa eit ikon som representerer vêret (t.d. ein sky eller ei sol for høvesvis skya og solfylt vêr) instansierer kvar `Day`-klasse ein `ImageRenderer`-klasse for å lasta og visa bildefilen med riktig ikon. Værikonbildefilene finst i katalogen `symbols` i utdelt zip-fil. Vi ber deg implementera funksjonane for lesing av bildefilene som inneheld vêrsymbol i oppgåvene R1-R3 og vi gir ei detaljert skildring av bildeformatet i innleiinga til desse oppgåvene.



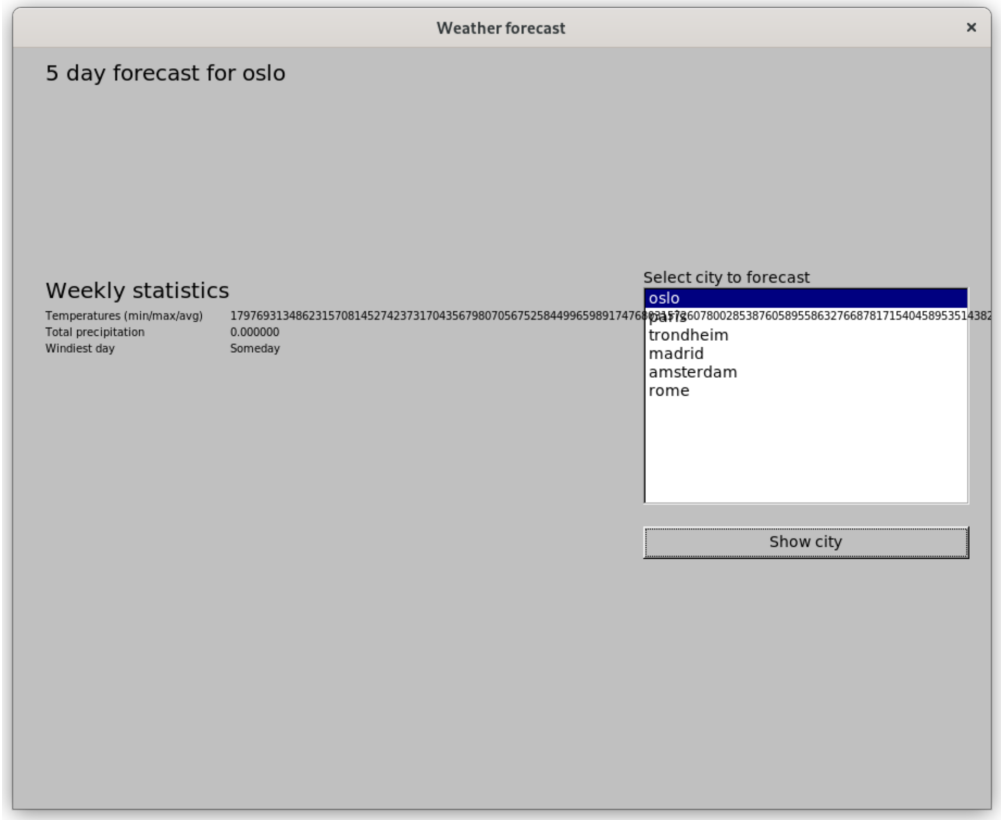
Figur 1: The complete application after selecting "Oslo" and clicking "Show city".

Korleis svara på del 3?

Kvar oppgåve i del 3 har ein tilhøyrande unik kode for å gjera det lettare å finna fram til kor du skal skriva svaret. Koden er på formata `<tegn><siffer>` (TS), eksempelvis F1, F2 og R1. Eit oversyn over koplinga mellom bokstav og fil er gitt i tabell 1. I forecast.cpp, og dei andre filene du skal løyse oppgåver i, vil du for kvar oppgåve finna to kommentarar som definerer høvesvis byrjinga og slutten av koden du skal føra inn. Kommentarane er på formatet: `// BEGIN: TS` og `// END: TS`.

Det er veldig viktig at alle svara dine er skrivne mellom slike kommentar-par, for å støtta sensurmekanikken vår. Viss det allereie er skrive noko kode *mellom* BEGIN- og END-kommentarane i filene du har fått utdelt, så kan, og ofte bør, du erstatta den koden med din eigen implementasjon med mindre anna er spesifisert i oppgåveskildringa. All kode som står *utanfor* BEGIN- og END-kommentarane SKAL de la stå.

Til dømes, for oppgåve G1 ser du følgjande kode i utdelte `robot_grid.cpp`



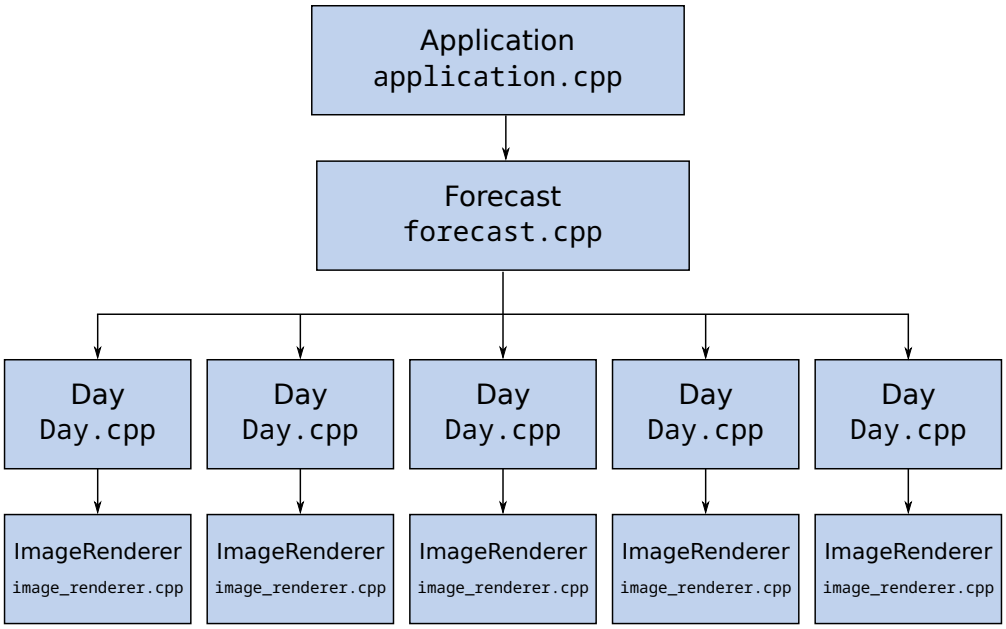
Figur 2: The handout application after selecting "Oslo" and clicking "Show city".

```
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 løysinga di, bør du enda opp med følgjande i staden:



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-kommentarane **IKKJE skal fjernast**.

Til slutt, viss du synest nokon av oppgåvene er uklare, oppgi korleis du tolkar dei og dei antagelsene du må gjera som kommentarar i den koden du sender inn.

Før du startar må du sjekka at den (umodifiserte) utdelte koden køyrer utan problem. Du skal sjå det same vindauget som i Figur 2. Når du har sjekka at alt fungerer som det skal er du klar til å starta programmering av svara dine.

3.2.1 Kor i koden finn du oppgåvene?

Kvar oppgåve har ein unik kode bestående av an bokstav følgd av eit tal. Bokstaven indikerer kva fil oppgåva skal løysast i. Sjå Tabell 1 for eit oversyn over bokstavar og filnamn. Eksempelvis finn du oppgåve F1 i fila `forecast.cpp`

Tabell 1: Oversyn over koplinga mellom oppgåve-bokstav og filnamn.

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

Oppgåvene

Formatting values (40 points)

Kvar vêrmelding må innehalda pent formaterte numeriske parametrar som beskriv forventa vêr som temperaturar og vindfart. I denne delen av oppgåva skriv vi funksjonane for å formatera desse verdiane.

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

   Skriv ein funksjon som tilordnar eit heiltal mellom 0 og 6 til eit vekedagsnamn. Den første dagen i veka (verdi 0) er måndag. Viss verdien av `weekday_num`-parameteren er utanfor dette området, bør funksjonen gi eit unntak.

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

   Skriv ein funksjon som samlar kompasvinklane (0 til 360 grader) til vindretningane til 8 forskjellige retningar (t.d. Nord, Nord-Vest, osb.). Viss verdien av `heading`-argumentet er utanfor det gyldige området, skal funksjonen kasta eit unntak.

   : Funksjonen skal nytta følgjande reglar for overgang frå 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**

   Vurder enum `Unit` definert i `util.h` som følgjer

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

   Skriv ein funksjon som gitt ein `Unit` enum-verdi returnerer den tilsvarande strengrepresentasjonen. Til dømes, `get_unit(UNIT_MM) == "mm"`.

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

   Skriv ein funksjon som formaterer numeriske vêrparametrar for presentasjon ved å avrunda flyttal til eit spesifisert tal desimalar og legga til ei eining. Funksjonen tar tre parametrar: `value` er verdien som skal

formaterast, `decimals` er antallet av desimaler som verdien skal avrundast til og `unit` er eininga som skal leggast til den formaterte strengen. Bruk funksjonen `get_unit` implementert i U3 for å konvertera `unit`-verdien til ein streng.

Til dømes skal funksjonen returnera verdiar som gjer følgjande samanlikningar er sanne (føresett at `get_unit`-funksjonen er korrekt implementerte): `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**

   No bruker vi hjelpefunksjonane definert tidlegare for å rendere ein dag i vêrmeldinga. For å legga til tekst i prognosen, er funksjonen `draw_text(Point pos, string value)` gitt. Der det er aktuelt, avrund flyttal til éin desimal og sørg for at riktig eining er lagt til. For å plassera teksten, bruk variablane `xpos` og `ypos` som held koordinatane til øvre høgre hjørne av ein dag. Lista nedanfor forteljar deg kva informasjon som skal vera tilstades i prognosen, tekstens relative y-offset og variabelen til `Day`-klassen der informasjonen finst. Bruk variablane `xpos` og `ypos` saman med dei relative y-offsetene for å konstruera `Point`-parameteren `pos` til `draw_text`-funksjonen.

   - Vekedagen. Y-posisjon: 20, variabel: `day`
   - Temperaturen. Y-posisjon: 100, variabel `temp`
   - Vindfarten og -retninga: Y-posisjon: 120, variablar: `vind_speed`, `wind_dir`
   - Nedbørsmengde: Y-posisjon: 140, variabel: `precip_amount`

Showing the forecast (50 points)

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

   Skriv ein funksjon som returnerer den horisontale pikselforskyvningen for kvar dag i prognosen. Kvar dag er 150 pikslar brei. Viss verdien av parameteren `day` er ugyldig, dvs. han er utanfor området $0 \leq \texttt{day} \leq 6$, skal funksjonen kasta eit unntak.

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

   Skriv ein funksjon som gjer om det første teiknet i ein streng til ein stor bokstav slik at `capitalize(tt-rondheim") == Trondheim"`. Denne funksjonen treng berre å endra det første teiknet i strengen. Hugs at funksjonen `toupper` kan brukast til å endra ein `char`-verdi til store bokstavar.

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

   Skriv ein funksjon som bereknar den totale nedbørsmengda forventa på tvers av alle dagane som er dekte i prognosen og legg resultatet i klassevariablen `total_precip`. Når denne funksjonen blir kalla er `total_precip` 0.

   For å gjera dette, iterer over `days`-vektoren i den noverande `Forecast` klasseinstansen . Hugs at vektoren `days` er definert som `vector<unique_ptr<Days>> days` i `forecast.h`. For å få mengda av nedbør som er forventa for ein dag, bruk den offentlege `precip_amount`-variabelen i `Days` klasseinstansen.

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

   Skriv ein funksjon som bereknar maksimums-, minimums- og gjennomsnittstemperaturar som blir forventa for dagane som er inkluderte i prognosen, og lager dei resulterande verdiane i høvesvis klassemedlemsvariablane `max_temp`, `min_temp` og `avg_temp`. Når denne funksjonen blir kalla er `max_temp` og `min_temp` sett til minimums- og maksimumsverdiane som blir støtta av C++ `double`-typen høvesvis.

   Som i dei førre oppgåvene, iterer over `days`-vektoren og bruk `temp`-medlemsvariabelen til `Day`-klasseinstansen for å få temperaturen som blir forventa for ein dag.

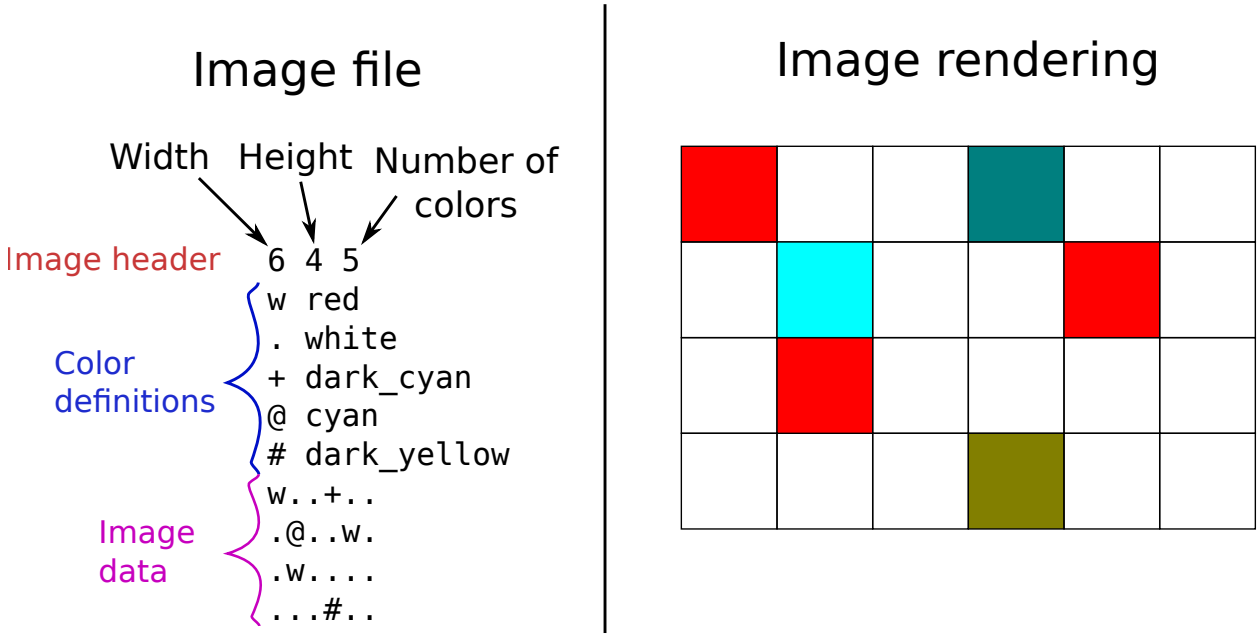
10. (10 points) **F5: Wind statistics**

    Skriv ein funksjon som finn den mest vindfulle dagen i veka, og legg resultatet i klassemedlemsvariabelen `windiest_day`. Hugs at vi er interesserte i å identifisera *dagnummeret* (dvs. eit heiltal mellom 0-6) og ikkje den faktiske vindfarten.

    Vindfarten for ein dag kan finnast i den offentlege `wind_speed`-medlemsvariabelen i klassen `Day`.

Attgjeving av bilde (90 poeng)

I denne siste delen av oppgåva implementerer vi funksjonane for lesing og attgjeving av bildeformatet som blir brukt for å lagra vêrsymbola i vêrmeldinga. Før vi startar på oppgåvene beskriv vi bildeformatet som dei følgjande oppgåvene ber deg implementera. [1]



Figur 4: Eit døme på ein enkel bildefil (venstre) og attgjevinga dens (høgre).

Bildefilformatet er delt inn i tre delar: Image headar, Color definitions og Image data. Delane er definerte som følgjer:

Image header Ei linje med tre tal skilt med mellomrom. Dei to første tala gir breidda og høgda på bildet og det siste talet gir tal fargar i bildet.

Color definitions Denne delen tilordnar ein *fargereferense* (color reference) til ein *fargenamn* (color name). Fargereferansen er eit enkelt teikn representert av C++-typen `char` og fargenamnet er ein `streng`. Teiknet kan vera kva som helst bortsett frå eit mellomrom, og kvar linje inneheld ein enkelt definisjon. Tal linjer, og dermed tal definisjonar, er gitt av det siste talet i bildeoverskrifta. Derfor, viss det siste talet i bildeoverskrifta er 5, som i Figur 4, bør du forventa at dei 5 linjene etter overskrifta er fargedefinisjon.

Image data Den siste delen av bildet inneheld dei faktiske bildedataa. Kvar linje i bildet tilsvarer ei linje i bildefilen, og derfor, viss eit bilde har 6x4 pikslar (som i Figur 4), inneheld bildeformatdelen

[1] For historisk referanse er filformatet som blir brukt i denne oppgåva ein liten forenkling av XPM2-filformatet som vart brukt til å lagra bilde i tidlege versjonar av X11 window system.

4 linjer med 6 teikn kvar. Kvart teikn tilsvarer ein av dei definerte fargane. Til dømes er dei to første horisontale pikslane i bildet i Figur 4 gitt av teikna "`w.`". Frå fargedefinisjonsdelen ser vi at `w` er tilordna til fargen `red` og `.` er tilordna fargen `white`. Derfor er dei to første pikslane i bildet høvesvis raude og kvite. For enkelhets skuld gir vi kartet `color_map` definert i klassen `ImageRenderer` som du kan bruka til å konvertera fargenamna som blir brukte i bildefilen til fargeverdiane som blir brukte i tegnefunksjonene.

Intern representasjon. Når eit bilde blir lasta inn frå ein bildefil, blir det transformert til den interne representasjonen som blir brukt i `ImageRenderer`-klassen. Den interne representasjonen består av to datatypar definerte i `image_renderer.h` som følgjer:

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

Map-et `colors` inneheld tilordningane mellom *fargereferansar* og fargekodane som blir brukte i Graph_Lib. Vi implementerer ein hjelpefunksjon for å legga til oppføringar i dette kartet i oppgåve R2 og funksjonen for å lesa fargekartleggingar frå bildefilen i oppgåve R6. Vektoren `image` inneheld dei faktiske bildedataa. Merk at det er definert som ein vektor av vektorar av fargar. Den ytre vektoren inneheld éin vektor per rad i bildet og dei indre vektorane inneheld fargane til pikslane som utgjer kvar rad. Så, etter at bildet i Figur 4 er lesne, inneheld `image`-vektoren 4 vektorar (tilsvarer høgda på bildet) som kvar inneheld 6 fargar (tilsvarer breidda på bildet)

Dei følgjande tre oppgåvene ber deg implementera parsarar for dei tre delane av bildeformatet. Deretter implementerer vi funksjonen for å attgi den interne representasjonen av eit bilde på skjermen.

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

    Skriv ein funksjon som returnerer den tilsvarande `Graph_lib` fargetypen som tilsvarer fargenamnet som er gitt i parameteren `color`. For å gjera dette, kan du bruka map-et `color_map` definert i `image_renderer.h`. Viss parameteren `color` refererer til ein ugyldig farge, skal funksjonen kasta eit unntak.

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

    Skriv ein funksjon som gitt ein fargereferanse (parameter `ref`) og eit fargenamn (parameter `color_name`) legg til eit element i `colors`-vektoren. Bruk funksjonen `get_color_value` for å slå opp fargenamna.

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

    Skriv ein funksjon som legg til ein ny vektor til `image`-vektoren. Hensikta er at denne funksjonen kallast for kvar ny rad med pikslar som blir funne ved lesing av bildefilen. Hugs at den nye vektoren må instansieres som ein `unique_pointer`.

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

    Skriv ein funksjon som legg ein piksel til bildet. Sidan bildedataa blir lesne linje for linje, skal pikselen altid leggast til han sist føydde til vektoren i `image`-vektoren.

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

    Fullfør funksjonen for lesing av image headar. Det meste av det harde arbeidet er allereie gjort for deg, men delen for å lesa dei faktiske verdiane inn i variablane `width`, `height` og `ncolors` manglar. Oppgåva di er å legga til dette.

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

    Implementer funksjonen for å lesa fargedefinisjonane til eit bilde. Eit døme på fargedefinisjonar er gitt i Figur 4

    Bildefilstrømmen er gitt i parameteren `image_file_stream`, og tal fargar som skal lesast blir send i parameteren `ncolor`. Når funksjonen kallast, peikar `image_file_stream` til byrjinga av fargedelen av

fila (føresett at du implementerte førre oppgåve riktig). For kvar fargedefinisjon som blir lesen, kall funksjonen `add_color` for å legga til fargen til den interne representasjonen. Funksjonen `add_color` tar to parametrar: ein `char` som inneheld fargereferansen og ein `string` som inneheld fargenamnet.

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

    Implementer funksjonen for å lesa pikselsdelen av bildet. Frå `width` og `height`-variablane initialisert i `read_image`-funksjonen, får du dimensjonane til bildet og dermed tal linjer med bildedata som skal lesast og tal teikn kvar linje har. For kvar linje du les, gjer eit kall til `add_line`-funksjonen utan argument. For kvar piksel du les, kall `add_pixel`-funksjonen med namnet på fargen på pikselen. Tilordningane mellom fargereferansar og verdiar blir lagra i map-et `colors`. Dermed kan du bruka fargereferensen du lesar frå bildefilen for å utføra eit oppslag i `colors` map-et for å få det nødvendige argumentet til `add_pixel`-funksjonen.