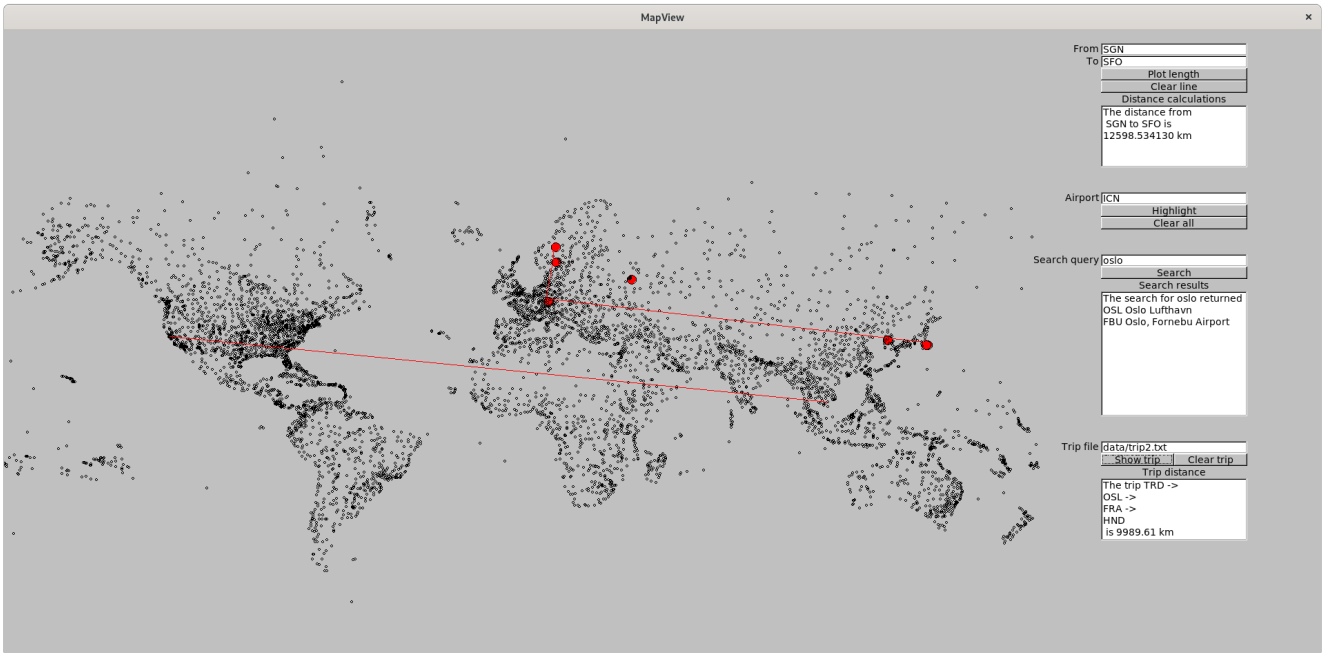

Figur 1: Den komplette applikasjonen

Del 3: Airports of the world

Denne del 3 av eksamenen ber deg om å fullføre en liten grafikk-applikasjon for interaktivt å utforske flyplasser rundt om i verden og fly-rutene mellom dem. Del 3 består av 12 del-oppgaver.

Oppgaven bygger på et datasett som inneholder alle verdens flyplasser, levert av openflights.org[1]. Datasettet, i en litt forenklet form, er inkludert i denne oppgaven (det ligger i `data/airports.csv`). Koden for å lese inn fra denne filen er allerede implementert for deg, slik at du ikke trenger å lage egen kode for det selv.

Siden posisjonen til flyplassene i datasettet er gitt som geografiske koordinater på den sfæriske jorden (en kule), må vi projisere dem over på et plan (2D) ved hjelp av en kartprojeksjonsmetode. Et eksempel på en slik metode er den mye brukte Mercator-projeksjonen[2]. I denne oppgaven bruker vi en variant av Mercator-projeksjonen kalt Web Mercator[3], som vanligvis brukes av online karttjenester.

Når koden er fullført skal programmet ha følgende funksjoner. De oppgavene som må fullføres for å implementere hver enkelt funksjonalitet er oppført i parentes.

- Plotting av flyplassplasser i et 2D-koordinatsystem/kart (Oppgave U1, A1)
- Utheving av en bestemt flyplass på kartet (Oppgave U1, A1, A2, E1, E3)
- Søk etter flyplasser basert på deres navn (Oppgave A3, E1, E4)
- Beregne avstanden mellom to flyplasser (Oppgave U1, U2, A1, E1, E5)
- Plotte og beregne den totale lengden på turer (Oppgave U1, U2, A1, A2, E1, E6)
- Validering av flyplasskoder (Oppgave E2)

Et skjermbilde av den komplette applikasjonen kan sees i Figur 1.

[1] `https://openflights.org/data.html#airport`
[2] `https://en.wikipedia.org/wiki/Mercator_projection`
[3] `https://en.wikipedia.org/wiki/Web_Mercator_projection`

Del U: Modell-implementering

I denne U-delen av eksamen (`util.cpp`) implementerer vi de essensielle delene av den matematiske modellen som støtter hoved-applikasjonen: Først Mercator-ligningen, som projiserer sfæriske koordinater (jordkloden) til et 2D-plan (kart). Og deretter Havesine-ligningen, som bestemmer stor-sirkel-avstanden (langs jordoverflaten) mellom to punkter på en kule (jordkloden).

Merk at du kan løse påfølgende oppgaver uten å løse oppgavene i denne U-delen, så hvis du blir sittende fast, anbefaler vi deg å gå videre.

U1: Web Mercator-projeksjon.

Implementer Web Mercator-projeksjonen som vist i formlene under og returner de resulterende x og y koordinatene som et `Point`.

Web Mercator-projeksjonen er gitt som to formler som beregner diskrete 2D x og y verdier (koordinater) henholdsvis, gitt sfæriske lengde- og breddegrader.

$$\begin{aligned} x &= \left\lfloor \frac{w}{2\pi}(\lambda + \pi) \right\rfloor \text{ pixels} \\ y &= \left\lfloor \frac{h}{2\pi} 2^{0.2} \left(\pi - \ln \left[\tan \left(\frac{\pi}{4} + \frac{\varphi}{2} \right) \right] \right) \right\rfloor \text{ pixels} \end{aligned} \tag{1}$$

der w og h er bredden og høyden på området kartet skal projiseres på (i piksler) og λ og φ er hhv. lengde- og breddegrader (`longtitude` og `latitude` i koden) til punktet som skal projiseres, oppgitt i radianer. x og y er pikselkoordinatene til punktene som skal plottes.

For enkelhets skyld leverer vi ut en `deg_to_rad` funksjon som kan konvertere grader til radianer. Dette er nødvendig siden formlene forventer at koordinatene er oppgitt som radianer og ikke i grader. Bruk konstanten `M_PI` for å få verdien av π. I tillegg er alle de matematiske funksjonene som brukes i formelen tilgjengelige fra C++ standardbiblioteket som `floor` ($\lfloor x \rfloor$), `log` ($\ln x$), `pow` (b^n) og `tan`.

U2: Avstandsberegning

Havesine-formelen er oppgitt som følger [4]

$$\begin{aligned} a &= \sin^2 \left(\frac{\Delta\varphi}{2} \right) + \cos \varphi_1 \cdot \cos \varphi_2 \cdot \sin^2 \left(\frac{\Delta\lambda}{2} \right) \\ c &= 2 \cdot \text{atan2} \left(\sqrt{a}, \sqrt{1-a} \right) \\ d &= R \cdot c \end{aligned} \tag{2}$$

der φ og λ er breddegrad og lengdegrad (hhv. `latitude` og `longtitude` i koden) i radianer og R er den gjennomsnittlige jordradius (6371 km). d er da avstanden mellom to koordinater i km langs overfalten av kulen.

I `distance`-funksjonen i `util.cpp`, implementer Havesine-formelen for beregning av stor-sirkels avstand mellom to geografiske koordinater.

For å hjelpe deg har vi funksjonen `deg_to_rad` som konverterer grader til radianer. Dette er nødvendig siden formlene forventer at koordinatene er oppgitt som radianer og ikke i grader. I tillegg er alle de matematiske funksjonene som brukes i formelen tilgjengelige fra C++ standardbiblioteket som `sin`, `cos`, `sqrt` og `atan2`.

Merk at

$\sin^2 x = \sin x \cdot \sin x$

[4] `http://www.movable-type.co.uk/scripts/latlong.html`

Del A: Datarepresentasjon

Denne gruppen oppgaver (A) omhandler representasjon og behandling av flyplass-dataene i programmet. Alle disse A-oppgavene skal implementeres i filen `airports.cpp`

A1: Koordinat-oversetting

Implementer funksjonen `get_map_coord` slik at den bruker static-funksjonen `to_map_coord` deklarert i klassen `Util` til å returnere et `Point` som representerer de projiserte koordinatene for gjeldende flyplassforekomst. Husk at de relevante parametrene finnes i følgende klassemedlemsvariabler `map_w` (x), `map_h` (y), `latitude` (breddegrad) og `longtitude` (lengdegrad).

A2: Flyplass-markering

Implementer funksjonen `highlight` som fremhever denne flyplassen (`this`) på kartet ved å

- sette fyllfargen til synlig rød
- sette radius på prikken til 7

Husk at klassen (`Airport`) arver fra `Circle`-klassen i `Graph_lib`, så du har tilgang til alle funksjonene fra den klassen i koden du lager nå. Se gjerne på den tilsvarende `restore`-funksjonen (implementert rett etter) for inspirasjon.

A3: Flyplass-søk

Implementer funksjonen `search` som returnerer en vektor av pekere til forekomster av `Airport` (dvs. en `vector<shared_ptr<Airport>`). Alle flyplassene i vektoren skal ha navn som inneholder den strengen som oppgis i parameteren `needle`. Husk å gjøre søket slik at det ikke har noe å si om det er store eller små bokstaver i navnet. For enkelhets skyld har vi laget funksjonen `string_to_lower` for å gjøre om en streng til å inneholde bare små bokstaver.

Returner en tom vektor hvis ingen matchende flyplass blir funnet. Husk at det finnes pekere til alle `Airport`-instanser i klassemedlemsvektoren `airport_list`.

Eksempel: hvis søkestrengen `needle` er "Tron" eller "tron", er den eneste matchende flyplassen "Trondheim Værnes". Den returnerte vektoren skal derfor da bare inneholde en enkelt `Airport`-objektinstanspeker.

Du kan bruke den eksisterende deklarasjonen og retur av `result`-vektoren som allerede er lagt inn i den utdelte koden.

Del E: Flyplassutforsker GUI

Følgende E-oppgaver implementerer funksjonaliteten til det grafiske brukergrensesnittet (GUI). Alle disse E-oppgavene skal implementeres i filen `explorer.cpp`.

E1: Flyplassoppslag

Implementer funksjonen `lookup_airport` som slår opp en flyplass med sin trebokstavs-kode, oppgitt i parameteren `code`. Bruk det Map'et du får tilgang til gjennom `airports->airport_map_by_code` for å gjøre dette. Dette er et normalt C++ `map` som du kan aksessere slik du er vant til, bortsett fra at nøkkeloppslagene er uavhengig av om det brukes store eller små bokstaver. Hvis flyplassen ikke blir funnet, skal du

- kalle funksjonen `alert` for å vise en meningsfull feilmelding
- kaste unntaket `AirportNotFoundException` definert i `explorer.h`

E2: Validering av flyplasskode

Implementer funksjonen `validate_code` som sjekker om flyplasskoden i string-parameteren `code` er gyldig. En gyldig flyplass-kode må

- være nøyaktig tre tegn
- inneholde bare bokstaver fra a til z (både store og små er ok)

For eksempel er BRU og icn gyldige koder, mens br, br0 eller ENGM er ugyldige.

Hvis du finner ut at flyplasskoden er ugyldig

- bruker du funksjonen `alert` for å vise en meningsfull feilmelding, og
- kaster et `InvalidAirportCode`-unntak.

E3: Flyplass-markering

Implementer callback-funksjonen for flyplass-markeringsknappen i det grafisk brukergrensesnitt. Denne skal fremheve flyplassen med den flyplass-koden som legges inn i tekstfeltet med navnet `in_airport` i denne klassen. Funksjonen skal

- få flyplasskoden lagt inn i tekstfeltet `in_airport`
- kalle funksjonen `validate_code` for å sikre at koden er gyldig
- slå opp navnet på flyplassen for å få en peker til det tilsvarende `Airport`-objektet
- kalle `highlight`-funksjonen til den returnerte `Airport`-objektpekeren for å markere flyplassen
- legge til `Airport`-pekeren i klassemedlemsvariabel-vektoren `highlighted` slik at den kan gjenopprettes til normal umarkert tilstand igjen senere.

E4: Flyplassøk

Bruk `search`-funksjonen fra `Airport`- klassen (Oppgave A3), og implementer koden som håndterer det som skal skje når flyplassøk-knappen i det grafiske brukergrensesnittet trykkes. Resultatet av søket skal vises i utskriftsboksen `search_results`.

For eksempel, et søk etter strengen "oslo" returnerer to resultater, OSL (Gardemoen) og FBU (Fornebu)[5]. Et eksempel på en passende utskrift å legge inn boksen `search_results` er da

```
The search for oslo returned
OSL Oslo Lufthavn
FBU Oslo, Fornebu Airport
```

For å generere utskriften ovenfor, husk at du har tilgang til medlemsverdiene (for eksempel kode og navn) for alle `Airport`-objektpekere som returneres fra søket, siden du har "public" tilgang til klassemedlemsvariablene. I dette tilfellet, forutsatt at `a` er en peker til et `Airport`-objekt, vil `a->code` og `a->name` returnere henholdsvis koden og navnet på flyplassen. Se på deklarasjonen av klassen `Airport` i `airports.h` for mer informasjon.

[5] Som, av en eller annen grunn, fremdeles er i databasen

E5: Distanse-beregning

Implementer koden som beregner avstanden mellom to flyplasser og skriver ut resultatet i tekstboksen for avstandsberegning (deklarert som `distance_results`). Tekstfeltene `in_from_airport` og `in_dest_airport` inneholder kodene til de to flyplassene.

Funksjonen din skal bruke avstandsfunksjonen fra `Util`-klassen (oppgave U2). Hent bredde- og lengdegradsverdiene fra Airport-objektpekerne returnert fra `lookup_airport`-funksjonen.

For eksempel, etter spørring om avstanden mellom OSL (Gardemoen) og SFO (San Francisco) kan følgende skrives ut i avstandsberegningsboksen:

```
The distance from
OSL to SFO is
8344.854 km
```

E6: Tur-filer

I denne oppgaven skal du implementere en del av en funksjonaliteten som tegner og beregner den samlede lengden på en reise som går innom flere forskjellige flyplasser.

En tur er definert i en fil som inneholder en enkelt linje med en sekvens av flyplasskoder adskilt med mellomrom. For eksempel:

```
TRD OSL EWR
```

spesifiserer en tur fra Trondheim (TRD) til Newark (EWR) via Oslo (OSL).

Vi ber deg om å implementere koden for åpning og lesing/parsing av slike filer inn i en vektor av strenger som inneholder flyplasskodene. Avstandskalkyle og plotting er allerede implementert i funksjonen `calculate_trip` som tar en slik vektor av flyplasskoder som input-parameter. Navnet (med path-informasjon) til turfilen legges inn i tekstfeltet `in_trip_file`. Du kan gjenbruke deklarasjonen av `codes`-vektoren og kallet til `calculate_trip` i implementeringen din.

Flere eksempler på turfiler finnes i den utleverte koden, i `data`-mappen, og de heter `tripX.txt`, der X er et tall.

Funksjonen skal bruke `alert`-funksjonen for å vise en passende feilmelding og kaste et `FileReadError`-unntak hvis noe går galt. Spesielt bør dette skje når

- Åpning av filen mislykkes
- Filen inneholder mindre enn to flyplasskoder

——————————— Slutt på del 3