Del 3: Programmering (60%)

I denne delen skal du skrive kode selv. Du vil få utdelt en .zip-fil. Her finner du en del utdelt kode. Det er tydelig markert hvor du skal skrive inn svaret på de ulike oppgavene:

```
//Begin task <tasknumber>

//Du skal skrive din kode mellom begin og end kommentarene

//End task <tasknumber>
```

I main()-funksjonen i main.cpp-filen kan du teste koden du har skrevet. Merk at dette *ikke* teller som en del av selve øvingen.

Dersom du mener at opplysninger mangler i en oppgaveformulering, gjør kort rede for de antagelser og forutsetninger du finner nødvendig. Skriv dette som kommentarer i koden din.

Husk at prøven er lang, og at det *ikke* er meningen at alle skal rekke alt. Oppgavene er plassert i en logisk rekkefølge innenfor hvert tema, men det betyr ikke at det nødvendigvis er økende vanskelighetsgrad utover i deloppgavene.

Merk: dersom du ikke greier å løse en oppgave kan du i senere oppgaver likevel anta at den koden du skulle ha skrevet fungerer som ønsket. Vær da oppmerksom på at koden din ikke nødvendigvis vil kompilere og kjøre.

Person (5%)

Etter en uke fylt med påskekrim har du bestemt deg for å endelig gjøre drømmen din til virkelighet: du skal starte et detektivbyrå! Basert på det du har sett og lest av krim er du rimelig sikker på at mye av jobben kan gjøres enklere dersom man programmerer litt. Noe som passer bra for deg, for du tar jo TDT4102 dette semesteret!

Det første du kommer på er at man bør lage en egendefinert datatype for å representere personene du støter på i sakene dine. Etter litt grubling kommer du fram til at du derfor vil ha en klasse kalt Person.

Dere har fått klassedeklarasjonen til klassen Person (se utdelt kode: person.h). Dere skal nå fylle inn medlemsfunksjonene etc. slik at disse oppfører seg som ønsket.

3.1 P1: Konstruktør til Person-klassen (2%)

Fyll inn konstruktøren (i person.cpp) slik at medlemsvariablene får korrekte og gyldige startverdier.

3.2 P2: get-funksjoner (1%)

Endre på de tre get-funksjonene i klassen Person (i person.cpp) slik at:

a) getFirstName() returnere personens fornavn
b) getLastName() returnere personens etternavn
c) getAge() returnerer personens alder.

3.3 P3: Overlast utskriftsoperatoren for Person (2%)

Overlast utskriftsoperatoren (operator<<) for objekter av klassen Person (fyll inn i person.cpp) slik at utskriften blir på dette formatet dersom personen heter Segenet Kelemu og er 63 år gammel:

```
Name: Segenet Kelemu
Age: 63
```

Suspect (7%)

Du blir sittende å gruble. Selv om alle folka du møter i forbindelse med sakene dine er personer og derfor kan beskrives med klassen Person, så er det en del som har informasjon og egenskaper som du ikke får dekket med Person. Det er spesielt alle de mistenkte i sakene dine du gjerne skulle fått inn mer informasjon om.

Du bestemmer deg for å lage en klasse kalt Suspect som arver fra klassen Person. Hvert trinn er beskrevet i deloppgavene under**. Alle medlemsfunksjonene skal skrives inne i klassedeklarasjonen til Suspect i person.h-filen** (dette går fint siden alle medlemsfunksjonene er forholdsvis korte).

3.4 S1: Klassen Suspect (2%)

Opprett en klasse Suspect som arver fra klassen Person. Klassedeklarasjonen skal du skrive under klassen Person i filen person.h.

3.5 S2: Medlemsvariabler (1%)

Suspect skal ha 1 privat medlemsvariabel:

- bool guilty;
  denne er true dersom man er sikker på at personen er skyldig (med andre ord false inntil det motsatte er bevist).

3.6 S3: Konstruktøren til Suspect (2%)

Lag konstruktøren til Suspect. Denne skal ta inn fornavn, etternavn og alder. Og gi verdier til:

- fornavn, etternavn og alder.
- guilty – denne skal settes til false ved start.

3.7 S4: get- og set-funksjoner (2%)

Lag get- og set-funksjoner for medlemsvariabelen guilty. Bør noen av disse være const?

TypeOfCase (2%)

Du er rimelig sikker på at det vil være lurt å ha en egendefinert datatype til å beskrive hva slags type sak det er snakk om. Det er jo tross alt en stor forskjell på mordsaker og stjeling av påskeegg. En enum (scoped enum, så enum class) er nok perfekt for dette.

Øverst i filen case.h finner du enum class TypeOfCase. Denne inneholder 4 ulike enum-verdier/symboler som representerer hva slags saker detektivfirmaet kan ha: murder (mord), robbery (tyveri/ran), kidnapping (kidnapping), og other (annet).

3.8 T1: Overlast utskriftsoperatoren for TypeOfCase (2%)

Overlast operator<< for TypeOfCase (fyll ut i case.cpp-filen). Dersom TypeOfCase er murder så skal tekstrengen «murder» skrives til ostreamen, dersom TypeOfCase er robbery skal tekstrengen «robbery» skrives til ostreamen osv.

Case (14%)

Nå er det på tide å lage en datatype som kan representere kriminalsakene detektivbyrået tar på seg. Klassedeklarasjonen til en slik klasse, klassen Case, ligger i filen case.h. Det kan være lurt å sette seg inn i hvilke private medlemsvariable klassen har.

3.9 C1: Konstruktøren til Case (2%)

Start med å fylle inn konstruktøren til klassen Case, du finner den i case.cpp-filen. Konstruktøren tar inn navnet på saken (name), typen sak (type), lønnen din (salary) og en Person-peker til klienten din (aClient), disse fire skal du bruke til å gi verdi til fire av klassens medlemsvariabler. Konstruktøren skal gi verdier til:

- caseName
- caseType
- caseCount – Denne representerer det totale antallet saker du har startet. Denne må derfor økes med 1 hver gang en sak startes (altså må den økes med 1 hver gang konstruktøren brukes). Den representerer med andre ord det totale antallet saker du noen gang har hatt.
- caseNumber – Dette er sakens ID-nummer og forteller hvilket nummer saken har. Med andre ord: Den skal være 1 dersom dette er din første sak og 42 dersom det er din 42. sak etc.
- salary
- client
- solved – denne skal være false ved start.
- set-et cases inneholder pekere til alle Case-objektene som er laget. Legg til en peker i cases til det nye Case-objektet som konstruktøren har opprettet.

3.10 C2: Destruktøren til Case (2%)

Fyll ut destruktøren (denne ligger også i case.cpp-filen). Sørg for at destruktøren rydder opp i det som eventuelt må ryddes opp i. Du kan sammenligne med konstruktøren din for å sjekke hva som eventuelt trenger opprydning i destruktøren.

3.11 C3: get- og set-funksjoner (4%)

Endre funksjonskroppen til følgende medlemsfunksjoner slik at de returnerer/endrer ønsket medlemsvariabler på korrekt måte (alle disse funksjonene ligger i case.cpp-filen):

a) getCaseName() – returnerer navnet på saken.
b) getCaseType() – returnerer typen kriminalsak
c) getCaseCount() – returnerer det totale antallet saker.
d) getCaseNumber() – returnerer ID-nummeret til saken.
e) getCases() – returnerer set-et med pekere til alle Case-objektene
f) getSalary() – returnerer lønnen du får for denne saken.
g) getClient() – returnerer en Person-peker til klienten din.
h) getDescription() – returnerer saksbeskrivelsen.
i) setDescription(string description) – oppdaterer medlemsvariabelen som beskriver kriminalsaken (caseDescription).
j) getSolved() – returnerer variabelen som viser om saken er løst eller ikke.
k) setSolved(bool solved) – oppdaterer medlemsvariabelen solved.
l) getClues() – returnerer vectoren med ledetråder (clues)
m) addClue(string clue) – legger til stringen clue i vectoren med ledetråder (clues).

3.12 C4: Overlaste operator< («mindre enn»-operatoren) for Case (2%)

Du skulle ønske det ikke var sånn, men dessverre er du, som de fleste andre, påvirket av det kapitalistiske samfunnet du lever i. Du vil derfor ha en måte å sammenligne sakene dine på basert på hvor mye du tjener på å løse hver sak.

Overlast operator< for klassen Case (denne ligger i case.cpp-filen) slik at true returneres dersom lønnen (salary) for saken på venstresiden av operatoren er mindre enn lønnen for saken på høyresiden, og false ellers.

3.13 C5: Oversikt over mistenkte (1%)

Du er rimelig fornøyd med Case-klassen din, men det er en viktig ting som mangler: en oversikt over alle de mistenkte.

Legg til en privat medlemsvariabel:
set<Suspect*> suspects; // dette set-et inneholder Suspect-pekere

3.14 C6: getSuspects() og addSuspect() (1%)

Lag deretter to public medlemsfunksjoner, **begge funksjonene skal skrives inne i klassedeklarasjonen til Case i case.h-filen,** for set-et suspects:

- set<Suspect*> getSuspects() const;
  Denne skal returnere set-et med Suspect-pekere.
- void addSuspect(Suspect* s);
  Denne skal ta inn en Suspect-peker kalt s og legge den til i set-et suspects.

3.15 C7: Overlast utskriftsoperatoren for Case (2%)

Overlast utskriftsoperatoren (operator<<) for objekter av klassen Case (fyll inn i case.cpp) slik at utskriften blir på dette formatet (gitt at saken er nummer 1 av totalt 12 saker, et ran, uløst og at informasjonen ellers er det som er gitt til objektet. Dersom saken er løst skal ordet «UNSOLVED» byttes ut med «SOLVED»):

```
Case 1 of 12: Pirat-mysteriet

Type of case: robbery

This case is UNSOLVED.

Client:
Name: Kaptein Undass
Age: 314

Description: Kaptein Undass var ute med skuta si da den ble overfalt
av pirater. Piratene stjal kaffe og kanelboller.

Suspects:
Name: Hakon Haugann
Age: 24

Name: Fanny Skirbekk
Age: 24

Clues:
En istykkerskutt skute.
En halv kanelbolle pa dekk som piratene trolig mistet.
Hakon drakk en kopp kaffe da jeg traff ham.
Piratene pratet om C++ da de robbet skipet.
```

Inntekt og lønnsomhet (6%)

Å starte en ny bedrift viser seg å være økonomisk krevende, selv om du er en utmerket detektiv. Du bestemmer deg derfor for å skrive noen funksjoner for å få bedre oversikt over økonomien.

3.16 L1: Totalinntekt (2%)

Skriv funksjonskroppen til medlemsfunksjonen int totalSalary() const. Den ligger i case.cpp-filen. Funksjonen skal returnere et heltall som representerer hvor mye du tjener på alle sakene dine til sammen.

3.17 L2: Gjennomsnittlig inntekt (2%)

Skriv funksjonskroppen til medlemsfunksjonen double averageSalary() const. Den ligger i case.cpp-filen. Funksjonen skal returnere et flyttall som representerer hvor mye du i gjennomsnitt tjener per sak.

3.18 L3: Kriminalsak med høyest inntekt (2%)

Skriv funksjonskroppen til medlemsfunksjonen const Case& caseWLargestSalary() const. Den ligger i case.cpp-filen. Funksjonen skal returnere en const-referanse til den saken der du tjener mest, sånn at du kan prioritere å jobbe med denne.

Filer (4%)

Du innser raskt at det kan være greit å arkivere opplysninger (både opplysninger om saker og om personer), slik at du har dem tilgjengelig også etter at objektet går ut av skop. Du vil derfor lage en funksjon som kan skrive et objekt av en vilkårlig datatype til en valgfri fil (så lenge datatypen har overlastet operator<<).

3.19 F1: Skriv til fil (2%)

I filen file.h finner du template-funksjonen void writeToFile(string fileName, const T& object). Denne funksjonen tar inn filnavnet til filen som du skal skrive til og et objekt (med variabelnavn object) av en vilkårlig datatype (T).

- Fyll ut funksjonskroppen og sørg for at objektet object blir skrevet til filen.
- Dersom filen ikke lar seg åpne skal du kaste et unntak.
- Du kan anta at operator<< er overlastet for datatypen til object.

3.20 F2: Unntakshåndtering (2%)

Lag en try-catch-blokk i funksjonen testWriteToFile() (denne ligger i file.cpp-filen) der du kaller på writeToFile()-funksjonen i try-blokken. Dersom filen ikke kan åpnes skal det kastes et unntak inne i funksjonen (som beskrevet i oppgave F1). Sørg for at catch-blokken fanger og håndterer dette unntaket. En fornuftig feilmelding til skjerm er en grei håndtering.

GUI - grafisk bruker-grensesnitt (14%)

I starten har du god oversikt over alle som er involvert i de ulike sakene, men etter hvert som du får flere og flere saker merker du at det er vanskelig å holde oversikt. Du støter stadig borti folk med navn du føler at du kjenner igjen, men du greier liksom ikke helt å plassere dem. Du bestemmer deg derfor for å lage en klasse kalt SearchWindow (som arver fra Window) som skal gjøre det mulig å søke opp folk knyttet til sakene dine. Selve klassedeklarasjonen ligger i case_gui.h, implementasjonen av funksjonene ligger i case_gui.cpp.

3.21 G1: Konstruktøren til SearchWindow (1%)

Konstruktøren er nesten ferdigskrevet, og alle medlemsvariablene har fått fornuftige startverdier. Fyll ut funksjonskroppen til konstruktør (den ligger i case_gui.cpp). Det eneste som mangler er å feste de to knappene, de to In_box-objektene, og Multiline_out_box-objektet til vinduet.

3.22 G2: Callback-funksjon til quit-knappen (2%)

Implementer callback-funksjonen til quit-knappen: cb_quit(). Den ligger I case_gui.cpp-filen. Callback-funksjonen skal kalle på medlemsfunksjonen quit(). Det er i funksjonen quit() at vinduet skal lukkes.

3.23 G3: Callback-funksjon til search-knappen (2%)

Implementer callback-funksjonen til search-knappen: cb_search(). Den ligger I case_gui.cpp-filen. Callback-funksjonen skal kalle på medlemsfunksjonen search().

3.24 G4: quit()-funksjonen (1%)

Implementer medlemsfunksjonen void quit(). Du finner den i case_gui.cpp-filen. I denne funksjonen skal vinduet lukkes.

3.25 G5: search()-funksjonen (8%)

Skriv medlemsfunksjonen void search() (den ligger i case_gui.cpp). Denne funksjonen skal lese inn innholdet fra de to In_box-ene firstName og lastName, for deretter å finne alle personene, både klienter og mistenkte, i Case-objektene som ligger i set-et caseSet og som har samme fornavn og/eller samme etternavn som det som ble skrevet i In_box-feltene. set-et caseSet er en medlemsvariabel i klassen SearchWindow som inneholder Case-pekere.

Disse personene skal deretter skrives til Multiline_out_box-objektet som heter searchResult.

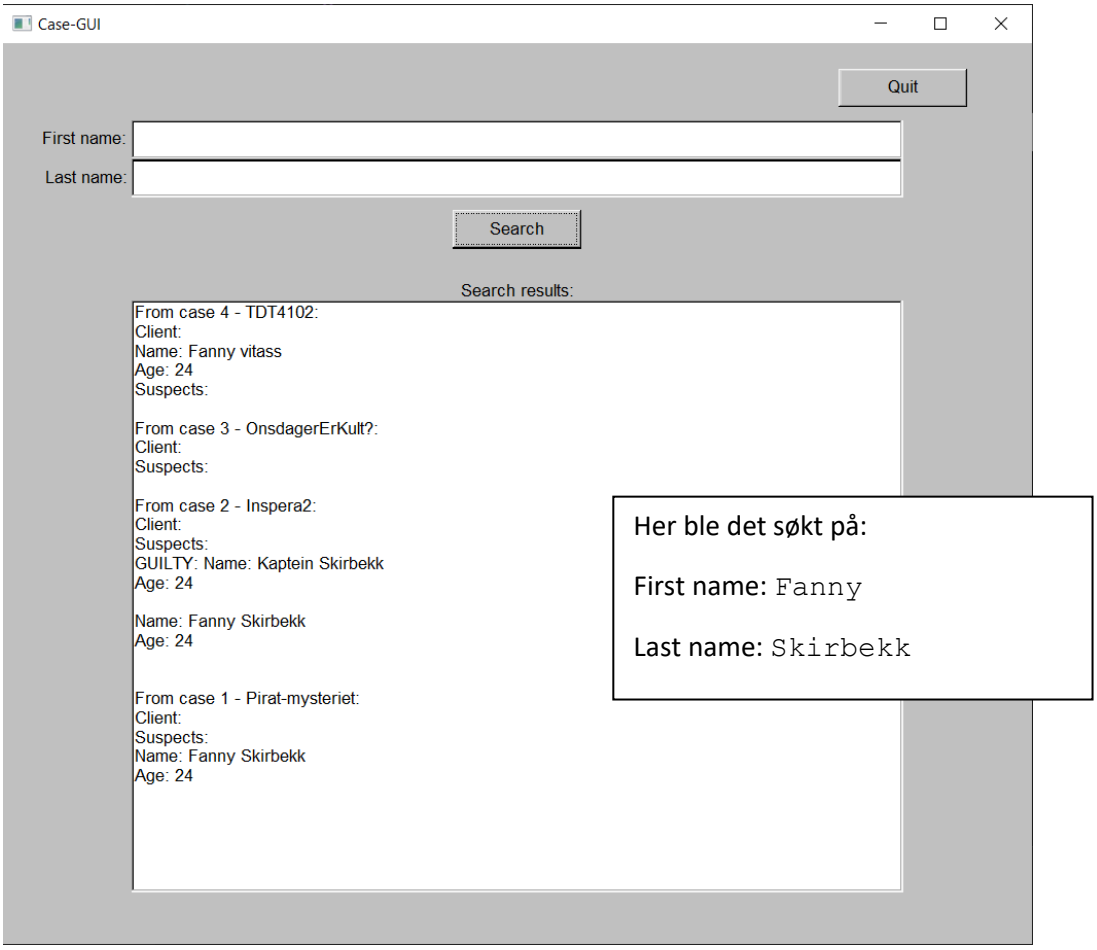
Tips:

- Man kan kun sende én string til Multiline_out_box-en. Det kan derfor lønne seg å skrive til en stringstream som man deretter gjør om til en string og sender til searchResult.
- Man sender strenger til Multiline_out_box-objekter ved å bruke medlemsfunksjonen put(). Denne fjerner det som allerede står i boksen.
- Husk å fjerne innholdet fra In_box-objektene etter at du har lest inn det som står der. Funksjonen clear_value() kan brukes til dette.

Outputen skal være på følgende format:

1. Start med en linje som forteller hvilken sak det er snakk om (både ID-nummer og navn på saken). Dette skal med selv hvis man ikke finner noen matchende personer i denne saken.
2. Neste linje skal inneholde strenge «Client:» for å gjøre det tydelig at den eventuelle personen under var klienten i saken.
3. Deretter skal klienten skrives til outputen (gitt at den matcher på fornavn og/eller etternavn)
4. På linja etter skal strengen «Suspects:» skrives.
5. Deretter skal alle de mistenkte i saken som matcher med fornavn og/eller etternavn skrives til outputen. Ha to linjeskift (altså en tom linje) mellom hver mistenkt.
6. Dersom en (eller flere) av de mistenkte i en sak er skyldige skal ordet GUILTY: skrives med store bokstaver før den skyldige personen skrives til outputen.
7. Dette skal gjentas for alle sakene som ligger i set-et caseSet.

Eksempel-output er gitt i figuren under. Merk at casene kommer i en vilkårlig rekkefølge.



Kodeknekking - dekryptering av hemmelige beskjeder (8%)

Som detektiv er det viktig å kunne dekode hemmelige meldinger på en rask og effektiv måte. Du bestemmer deg derfor for å skrive en funksjon for å gjøre nettopp dette.

3.26 K1: Dekryptering (8%)

Du skal nå skrive funksjonen void decoding(string inFileName, string outFileName). Denne er deklarert i decoding.h og definert i decoding.cpp. Du skal fylle inn funksjonskroppen til funksjonen i decoding.cpp-filen.

- Funksjonen skal lese fra filen med navn inFileName, dekryptere innholdet i filen og skrive den dekrypterte meldingen (altså meldingen på et vanlig/leselig format) til filen outFileName.
- Dersom minst én av filene ikke kan åpnes skal du kaste et unntak.
- I den krypterte meldingen har alle bokstavene blitt byttet ut med bokstaver som er et gitt antall steg lenger ut i alfabetet. Dersom steget er på 3 har alle A-er blitt byttet ut med D-er alle B-er blitt byttet ut med E-er og alle Z-er blitt byttet ut med C-er osv.
- Det er kun bokstavene A-Z og a-z som er med her. De andre tegnene skal ikke endres på.
- inFileName har følgende format:
    1. Den første linjen i filen representerer skiftet. Du kan anta at den «minste» bokstaven alltid vil stå først på denne linja, og at begge bokstavene er store (upper case).
    2. Resten av filen inneholder meldingen som skal dekodes.
    3. Du kan anta at filen alltid vil ha korrekt format.

Eksempel-fil (denne ligger også i den utdelte koden og heter secret_message.txt):

```
B -> G

Bjgg itsj ts ymj Nsxujwf-fxxnlsrjsy, dtz bnqq gj f lwjfy uwtlwfrrjw
fsi ijyjhynaj!

Gjxy wjlfwix, ymj unwfyjx.
```

Output-filen skal da inneholde:

```
Well done on the Inspera-assignment, you will be a great programmer
and detective!

Best regards, the pirates.
```

---------- **Slutt på Inperaøving 2** ----------