Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

TDT4102 Prosedyre- og objektorientert programmering Vår 2023

Øving Inspera

Praktisk:

Det er veldig viktig at du leser det praktiske før du begynner på oppgavene.
Alle oppgavene i del 3 er satt opp slik at de skal besvares i eksisterende funksjoner i .cpp-filene i den utdelte koden. I filene du skal skrive i vil du finne to kommentarer for hver oppgave, som definerer begynnelsen og slutten på koden du skal føre inn. Kommentarene er på formatet

// BEGIN: 1a og // END: 1a

Det er veldig viktig at alle svarene dine er skrevet mellom slike kommentar-par. Kode utenfor et slikt par blir ikke vurdert. BEGIN- og END-kommentarene skal **IKKE** fjernes!

Den utdelte koden inneholder ikke konfigurasjonsfilene som er nødvendig for å kjøre programmet. Du må derfor kjøre "TDT4102: Create Project for TDT4102 Template -> **Configuration only**" fra Comand Palette i VS Code. Dette skal gjøres i mappen du legger den utpakkede koden i, altså i mappen du skal jobbe med øvingen.

Når du vil levere, må du lagre filene, og levere dem som en zippet mappe i Inspera. Husk at hvis du endrer på koden, må du lage zip-filen på nytt.

Intro:

Du har nylig blitt ansatt i et sommerinternship i forsikringsselskapet Motstridige. De ønsker at du skal være med å lage funksjonalitet for å holde kontroll på og automatisere alle forsikringsavtalene de har. For å representere en forsikringsavtale har du fått utdelt klassen `InsuranceContract`. Videre er de ulike typene forsikringer definert i et scoped enum, `InsuranceType`. For å holde orden på alle avtalene er det definert en database, klassen `ContractDataBase`. En database er kort fortalt en organisert samling av strukturert informasjon, i dette tilfellet noe som kan holde `InsuranceContract`-objekter. Databasen skal være sortert med hensyn på id-en til objektene, i stigende rekkefølge ut ifra plasseringen i medlemsvariabelen `ContractDataBase::contracts`.

1 Database-funksjonalitet

Funksjonen under finnes i `InsuranceContract.cpp`

a) *10 poeng*

For å kunne jobbe med `InsuranceContract`-klassen er det viktig å kunne skrive den ut på en fornuftig måte. For å kunne gjøre dette må man kunne konvertere variabler av `InsuranceType` til en streng.

Implementer funksjonen
`string insuranceTypeToString(InsuranceType t)`.
Funksjonen skal konvertere inputargumentet av typen `InsuranceType` til en streng. Strengen skal være på engelsk, med stor forbokstav, slik at for eksempel `InsuranceType::Car` blir `"Car"`.

Du kan teste om funksjonen fungerer med den allerede overlastede «-operatoren til `InsuranceContract`-klassen.

Funksjonene under finnes i `ContractDataBase.cpp`

b) *10 poeng*

Den kanskje viktigste oppgaven til datasystemet er å kunne finne igjen forsikringsavtaler.

Implementer medlemsfunksjonen
`InsuranceContract ContractDataBase::getContract(int id)`.
Funksjonen skal returnere avtalen med den gitte id-en. Du kan anta at id-en finnes i databasen. Du kan teste koden din på databasen `db` i `main()`. Blant annet id `1234` skal returnere et gyldig objekt.

c) *15 poeng*

Selskapet ønsker å holde oversikt over hvor mange avtaler av hver type de har.

Implementer medlemsfunksjonen
`int ContractDataBase::numberOfInsuranceType(InsuranceType type)`.
Funskjonen skal returnere hvor mange kontrakter det er av input-typen i databasen.

d) *15 poeng*

En database er ikke mye verdt uten å kunne legge til nye verdier.

Implementer medlemsfunksjonen
`int ContractDataBase::addContract(string holderName, InsuranceType insType, int value)`.
Funksjonen skal lage et nytt `InsuranceContract`-objekt og legge det til i databasen. Id-en til den nye kontrakten skal være én høyere enn den høyeste nåværende id-en, og "1" hvis databasen er tom. Fuksjonen skal returnere id-en til det nye objektet.

e) *15 poeng*

Implementer medlemsfunksjonen
`void ContractDataBase::saveContracts(string filename)`.
Funksjonen skal lagre databasen til filen med navnet `filename`. Formatet på den strukturerte filen skal være på samme format som i `DataBase.txt`. Her er medlemsvariablene i `InsuranceContract`-klassen lagret i rekkefølgen de er definert i klassen, separert med et `','`.

2 Forsikringsvilkår

Funksjonene under finnes i `Utilities.cpp`

a) *10 poeng*

Forsikringsbrasjen krever vilkår på mange språk, blant annet engelsk, fransk og gresk. Selskapet har dessverre ingen som kan gresk, men tenker at siden du er så flink, klarer du sikkert å finne en løsning. Du oppdager at hvis du forskyver alle bokstavene i et ord to plasser til høyre i alfabetet, ser teksten helt gresk ut. For eksempel blir teksten "alle" til "cnng". Siden ingen leser vilkårene uansett, bestemmer du deg for at dette er en gyldig løsning på problemet.

Implementer funksjonen `string toGreek(string sentence)`.

Funksjonen skal ta inn en streng, og flytte alle bokstavene i strengen to plasser til høyre i alfabetet. Alle mellomrom skal forbli mellomrom. Du trenger ikke å ta hensyn til andre ting, som punktum eller at bokstaver går utenfor alfabetet.

I forsikringsbransjen er en av de viktigste jobbene å skrive uforståelige vilkår. Dette synes alle er grøsselig kjedelig å gjøre, og selskapet håper derfor at du kan automatisere prosessen. Du bestemmer deg derfor for å lage en svadagenerator med forsikringstema. En svadagenerator setter tilfeldig sammen uttrykk som høres relevante ut, og lager slik meningsløse sentninger, selv om de ved første øyekast ser meningsfulle og fornuftige ut.

For å implementere generatoren har du fått tak i filen `SvadaWords.txt`. Filen inneholder relevante uttrykk for forsikringssvada, fordelt på 7 grupper. Utrykkene ligger på hver sin linje, og de ulike gruppene er adskilt med en linje med `"||"` i `txt`-filen. Setninger kan lages ved å plukke ut ett uttrykk fra hver gruppe, og sette dem sammen i rekkefølgen til gruppene.

b) *15 poeng*

For å kunne generere svada må vi først laste inn innholdet.

Implementer funksjonen `vector<vector<string» loadSvada()`.

Funksjonen skal gå gjennom hver av de 7 gruppene i `SvadaWords.txt`, og legge uttrykkene i **én** `vector<string>` for **hver** gruppe. Videre skal `vector<string>`-en fra hver gruppe legges i en `vector<vector<string»` som skal returneres. Funksjonen skal altså returnere en vektor med 7 vektorer med drøye 20 strenger i hver.

c) *15 poeng*

Nå skal du bruke innholdet du har lastet inn til å generere svada.

Variabelen `testSvadaGenerationVec` i `Utilities.h` er av samme format som output-en til `loadSvada()`, og kan brukes til å teste denne funksjonen hvis du ikke har løst 2b).

1. **Implementer funksjonen `string svadaGenerator(vector<vector<string» svadaVec)`.**
   Funksjonen skal gå gjennom hver av de 7 vectorene fra inputen `svadaVec`, og tilfeldig hente **én** streng fra hver. Strengene skal settes sammen til en setning i samme rekkefølge som vektorene de hentes fra. Funksjonen skal returnere denne setningen. Merk at det ikke er mellomrom etter uttrykkene, og at alle ordene i siste gruppe kommer med et punktum.
2. Bruk funksjonene du har implementert i b) og c) til å generere forsikringsvilkår i konstruktøren til `InsuraceContract`. Her skal hvert objekt få 10 setninger med unikt svada som forsikrinsgvilkår.