Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

TDT4102 Prosedyre- og objektorientert programmering Vår 2019

Øving 5

Frist: 2019-02-15

Mål for denne øvingen:

- Lære å bruke enum-typer
- Lære å bruke strukturer (`struct`)
- Lære å implementere og bruke klasser (`class`)

Generelle krav:

- Det er valgfritt om du vil bruke IDE (Visual Studio, Xcode), men koden må være enkel å lese, kompilere og kunne kjøre.
- 70% av øvingen må godkjennes for at den skal vurderes som bestått.
- Øvingen skal godkjennes av stud.ass. på sal.

Anbefalt lesestoff:

- Kapittel 9

Bakgrunn for oppgavene

I denne øvingen skal vi lære å definere egne typer. Til det brukes enum-typer, strukturer og klasser til å definere en kortstokk. Disse typene skal du til slutt bruke til å programmere kortspillet Blackjack. En vanlig kortstokk består av 52 kort, delt inn i fire *farger* (suits): hjerter (hearts), ruter (diamonds), kløver (clubs) og spar (spades). Det finnes 13 kort av hver farge: ess (ace), 9 tallkort med tallene 2 til 10 og de tre bildekortene knekt (jack), dame (queen) og konge (king). Avhengig av hvilket kortspill det er snakk om kan ess være det *mest* verdifulle kortet, eller det *minst* verdifulle kortet. I denne øvingen, før blackjack, skal vi anta at ess er det mest verdifulle kortet, med verdi lik 14.

Hvis vi vil beskrive et individuelt kort i kortstokken med ord skriver vi for eksempel «ruter fem», «hjerter ess» og «spar konge». På engelsk skriver vi for eksempel «ace of spades», «five of diamonds» og «king of hearts». Når kort skal beskrives på denne måten i øvingen er det valgfritt om du vil bruke norsk eller engelsk, men vær konsekvent.

Merk: på norsk brukes ordet «farge» til å beskrive symbolet på kortet, eller det som på engelsk heter «suit». Dette kan være forvirrende, siden kortene også er delt i de *røde* kortene (hjerter og ruter) og de *svarte* kortene (kløver og spar). I denne øvingen brukes ordet «farge» for å skille mellom kløver, ruter, hjerter og spar.

Konvensjoner for klasser

I denne øvingen skal du implementere dine egne typer ved å bruke klasser. Det er konvensjon i C++ at navn på typer (og dermed klasser) starter med stor forbokstav. For å gjøre koden mer leselig skal du følge denne konvensjonen.

Det finnes veldig mange stiler å skrive kode i. Noen liker å dele opp ordene i et navn med understrek og andre liker å starte hvert ord med stor bokstav. Populært kalles det bl.a `snake_case` og `camelCase`. Når du skriver kode, så kan du fint velge den formen du selv liker best, men vært konsekvent. Hvis du deklarerer navn med forskjellige stiler blir koden fort veldig rotete og vanskelig å lese.

Andres kode kan også ha en annen stil enn din egen. For å bruke standardbiblioteket er så å si alle navn med kun små bokstaver, andre bibliotek kan ha store bokstaver og andre kan ha understrek. Så lenge din kode er konsekvent er du på god vei.

Typenavnene dine kan derfor være på formen:

`MyType` eller `My_type`

i motsetning til variabelnavn, som vanligvis er på formen

`myVariable` eller `my_variable`.

I de fleste av øvingene oppgir vi hvor funksjoner og klasser skal deklareres og defineres. Det er en retningslinje du bør følge. Normalt ønsker du å dele opp koden etter logisk sammenhengende komponenter. Det betyr ofte at en klasse opptar to filer: en headerfil og en implementasjonsfil. I andre tilfeller kan det også være hensiktsmessig å samle flere klasser i samme fil, men da gjelder det å være presis i navngivning av headerfil og implementasjonsfil.

1 Enumerasjoner (10%)

Det er ingen grafikkoppgaver i denne øvingen, alt du trenger er et prosjekt med `std_lib_facilities.h`.

I denne oppgaven skal du lage og bruke enumerasjoner. Enumerasjoner er enkle, brukerdefinerte typer med et sett enumeratorer (verdier). Enumeratorene er symbolske konstanter. I denne oppgaven skal vi bruke *scoped enums*, det betyr at en enumerator lever i scopet til en enumerasjon. Det er nyttig, da slipper vi å være veldig forsiktig med navngivning og vi får typesikkerhet med på kjøpet. For å referere til en enumerator må vi først velge enumerasjonen, eller scopet, den er definert i. Det kalles å «oppløse» et scope. Det gjøres ved å skrive navnet på enumerasjonen etterfulgt av `::`. F.eks. `Month::july` eller `Day::sunday`. Legg også merke til at Month::july ≠ Day::sunday, selv om de begge to representerer den syvende måneden og den syvende dagen, det gjør jobben som programmerer mye enklere.

Merk: For at det skal være mulig å inkludere enum-typer i flere kilderfiler, må enum-typer deklareres i en headerfil.

a) Suit.

Lag enum-typen `Suit`. Definisjonen skal ligge i headerfilen `Card.h`.

`Suit` skal representere *fargen* til et kort, og kan ha verdiene `clubs`, `diamonds`, `hearts` og `spades`.

For å opprette en scoped enum er syntaksen:

```
enum class Name { enumerator0, enumerator1 /*, osv.*/};
```

Se også §9.5 i læreboken.

b) `Rank`.

`Rank` skal representere *verdien* til et kort, og kan ha verdiene `two`, `three`, `four`, `five`, `six`, `seven`, `eight`, `nine`, `ten`, `jack`, `queen`, `king` og `ace`.

c) Definer funksjonen `suitToString()`.

Funksjonen skal ta inn en `Suit` og returnere en `string` som representerer `Suit`-en som tekst. Siden tekststrengen skal velges basert på en enum, så bør `switch` brukes til å velge riktig tekst for riktig `Suit`. F.eks: `Suit::spades` skal bli strengen "Spades".

En veldig enkel måte å løse denne, og den neste, oppgaven på er å benytte beholderen `map`: et map for farge og et map for verdi, `map<Suit, string>` og `map<Rank, string>`. For informasjon om `map`, se bokens kapittel 21.6.2 og B.4. Det er også gjennomgått i forelesning hvordan denne oppgaven kan løses. Ved bruk av map er det ikke *nødvendig å bruke switch for å velge streng.*

d) Definer funksjonen `rankToString()`.

Gjør tilsvarende for denne funksjonen som i *c)*, for `Rank`. F.eks: `Rank::three` skal bli strengen "Three".

e) Teori.

I denne oppgaven har vi valgt å represenetere farge og verdi på kort som symboler. Nevn to fordeler ved å bruke symboler framfor f.eks. heltall og strenger i koden.

2 Struktur (10%)

I denne delen skal du lage og bruke strukturer.

I denne oppgaven skal du definere din egen type som skal representere et kort vha. enum-typene fra forrige oppgave. Enum-typene gjør det mulig å begrense hvilke kort som kan representeres.

a) Struct.

Lag strukturen (`struct`) `CardStruct` i `Card.h`

Strukturen skal holde variablene:

- `Suit s`, en variabel av typen `Suit`.
- `Rank r`, en variabel av typen `Rank`.

b) Tekstrepresentasjon av kort.

Definer funksjonen `toString`(). Funksjonen skal ta et kort (`CardStruct`) som argument. Funksjonen skal returnere en tekstlig representasjon av kortet i form av en `string`-variabel, for eksempel «Ace of Spades» (engelsk) eller «spar ess» (norsk).

c) Kompakt tekstrepresentasjon av kort.

Definer funksjonen `toStringShort`(). Funksjonen skal ta et kort (`CardStruct`) som argument. Funksjonen skal returnere en `string` som inneholder en kort og kompakt tekstlig representasjon av kortet, på formen `<farge (en bokstav)><verdi som tall>`. For eksempel skal spar ess representeres som `S14`, der «S» står for spar (spades) og 14 er verdien til ess.

Nyttig å vite: Konvertering fra tall til tekstreng

Standardbiblioteket har en funksjon, `to_string`, som konverterer fra flere typer til string. Enn så lenge er det nok å være klar over at den fungerer for heltall. Den ligner på:

```
string to_string(int number) {
    ostringstream os;
    os << number;
    return os.str();
}

// kan konvertere heltall til string
string s = to_string(10); // s inneholder "10"
```

Merk: *`to_string` kan også ta inn flyttall, men oppfører seg ikke alltid som forventet. Blant annet får du bare seks siffer etter komma, og litt annerledes oppførsel for store tall. Dersom du selv ønsker å bestemme presisjonen må du bruke den «gamle» metoden. For mer info, se eksempelet her: https://en.cppreference.com/w/cpp/string/basic_string/to_string.*

d) Test strukturen og funksjonene dine fra `main()`.

Det forventes at du alltid tester koden din, selv om det ikke er oppgitt i oppgaveteksten. Hvis du er usikker på hvordan du kan teste koden din kan du spørre en læringsassistent.

3 Kortklasse (20%)

I denne deloppgaven skal du lage en klasse `Card` med grunnleggende funksjoner. Fra nå av kan du se bort fra structen `CardStruct` som ble definert i oppgave 2, framover skal klassen `Card` benyttes.

Kortstokklassen modellerer en fransk kortstokk som består av 52 unike kort med fire forskjellige farger og 13 forskjellige verdier. Vår kortstokk inneholder ikke jokere.

a) Class.

Deklarer klassen `Card`. Denne klassen skal inneholde følgende `private` medlemsvariabler:

- `Suit s`, en variabel av enum-typen `Suit`, som definert i oppgave 1.
- `Rank r`, en variabel av enum-typen `Rank`, som definert i oppgave 1.
- `bool valid`, en logisk variabel som skal indikere om `suit` og `rank` har fått tilegnet verdier eller ikke, dvs. hvorvidt kortet er *gyldig*.

b) Definer defaultkonstruktøren `Card()`.

Konstruktøren skal ikke ta inn noe. Siden det ikke finnes et kort som ikke har farge og verdi, er kortet ugyldig.

c) Definer konstruktøren `Card(Suit suit, Rank rank)`.

Denne konstruktøren skal ta inn variablene `suit` og `rank` av typene `Suit` og `Rank`. Det tilfredsstiller invarianten til klassen og kortet er gyldig.

Konstruktøren skal initialisere medlemsvariablene `s` og `r` for objektet. Det bør gjøres med det som kalles *member initializer list*, det kan du lese om i læreboken nederst på side 314 og 315.

d) Definer medlemsfunksjonen `suit()`.

Funksjonen skal returnere kortets farge. Funksjonen skal være `public`.

e) Definer medlemsfunksjonen `rank()`.

Funksjonen skal returnere kortets verdi. Funksjonen skal være `public`.

f) Definer medlemsfunksjonen `isValid()`.

Denne funksjonen skal returnere en logisk verdi som gjenspeiler om kortet er gyldig eller ikke. Et kort er gyldig når det har farge og verdi.

g) Definer medlemsfunksjonen `toString()`.

Funksjonen skal returnere en representasjon av kortet i form av en `string`-variabel. Dersom kortet er ugyldig skal strengen «Ugyldig kort» returneres. Funksjonen skal være `public` og ikke ta inn noe.

h) Definer medlemsfunksjonen `toStringShort()`.

Denne funksjonen skal returnere en *kort og kompakt* representasjon av kortet i form av en `string`-variabel. Dersom kortet ikke er gyldig skal strengen «Ugyldig kort» returneres. Funksjonen skal være `public` og ikke ta inn noe.

i) Refleksjon og teori.

I denne deloppgaven har vi valgt å skrive en klasse for å representere et spillkort. I forrige deloppgave gjorde vi akkurat det samme med en struct.

Hvorfor ønsker vi heller å representere spillkortet som en klasse framfor en struct?

Hva er klassens invariant?

4 Kortstokklasse (20%)

I denne deloppgaven skal du implementere klassen `CardDeck`, som bruker en samling av objekter av klassen `Card` for å representere en kortstokk. Du skal deretter implementere enkel funksjonalitet for `CardDeck`. Klassen deklareres og defineres i egne filer, hhv. `CardDeck.h` og `CardDeck.cpp`.

a) Deklarer klassen `CardDeck`.

Klassen skal inneholde følgende `private` medlemsvariabler:

- `cards`, en `vector` som kan holde `Card`-objekter - dette er beholderen som skal holde alle kortene i kortstokken.
- `currentCardIndex`, et heltall som brukes til å holde styr på hvor mange kort som er blitt delt ut.

b) Definer konstruktøren `CardDeck()`.

Konstruktøren må sørge for at alle kortene i kortstokken blir satt opp riktig. Det vil si at hvert kort må settes opp med rett farge (`Suit`) og verdi (`Rank`), slik at kortstokken representerer en standard kortstokk som beskrevet i «bakgrunn for øvingen». I tillegg må `currentCardIndex` settes til 0, siden ingen kort har blitt delt ut ennå. Denne konstruktøren skal ikke ta inn noe.

c) Definer medlemsfunksjonen `swap()`.

Denne funksjonen skal ta inn to indekser (heltall) til `cards-vectoren` og bytte om på kortene som finnes ved disse to posisjonene. Bør funksjonen være `private` eller `public`? Hvorfor?

d) Definer medlemsfunksjonen `print()`.

Denne funksjonen skal skrive ut alle kortene i kortstokken til skjerm. Bruk den *lange* `string`-representasjonen til å skrive ut hvert kort.

e) Definer medlemsfunksjonen `printShort()`.

Denne funksjonen skal skrive ut alle kortene i kortstokken til skjerm. Bruk den *korte* `string`-representasjonen til å skrive ut hvert kort.

f) Definer medlemsfunksjonen `shuffle()`.

Denne funksjonen skal stokke kortstokken, dvs. plassere kortene i tilfeldig rekkefølge i `cards`-tabellen. Kan du bruke en annen medlemsfunksjon til å implementere deler av `shuffle()`?

g) Definer medlemsfunksjonen `drawCard()`.

Denne funksjonen skal trekke det «øverste» kortet i kortstokken. Når den kalles for første gang skal den returnere det første kortet i `cards`-tabellen, deretter det andre kortet, og så videre.

Siden kortet ikke skal kopieres, men trekkes, ut av kortstokken bør det returneres en referanse til kortet. Kortet skal heller ikke kunne endres, det strider mot fysikken til kortet, så det medlemsfunksjonen bør returnere er en referanse til et konstant objekt, altså return by const reference, `const Card& drawCard()`.

Hint: Bruk currentCardIndex.

5 Blackjack (40%)

Til nå har du fått beskrevet hvordan funksjoner, klasser og programmer skal designes. I denne oppgaven skal du få kjenne litt på hvor vanskelig selv tilsynelatende enkle regler er å implementere i kode. Formålet er at du skal få trening i å definere egne funksjoner og klasser.

Selv om du ikke må fullføre hele spillet for å få godkjent hele øvingen anbefales du å forsøke. Det er krevende, og nettopp derfor verdifull erfaring. Å lære seg programmering er ikke bare syntaks og språk, det er også en øvelse i hvordan du kan uttrykke dine ideer og din logikk med et programmeringsspråk.

I denne oppgaven skal du implementere klassen `Blackjack`. Klassen skal brukes til å spille det populære kortspillet Blackjack. Reglene til Blackjack vil bli forklart i tekst og din oppgave er å implementere logikken i spillet. Reglene er kopiert fra Wikipedia og lyder:

Både du og dealeren vil få to kort hver, spilleren får det første kortet, dealeren det andre, spilleren det tredje, osv. Spilleren vil kun se det ene kortet til dealeren, mens det andre kortet vil ligge vendt ned mot bordet. Heretter vil spilleren måtte velge om han skal ha flere kort, eller «stå» med kortene han allerede har. Dealeren må alltid stå på 17, noe som betyr at hvis han havner på 17 eller over vil han ikke kunne trekke flere kort.

- Et ess teller enten 1 eller 11
- Alle bildekort (J, Q, K) har verdi 10
- Du får alltid utdelt to kort til å begynne med
- Hvis den samlede verdien på kortene er over 21 er du ute
- Dealeren må stå på totalverdi 17 eller mer
- Et ess sammen med 10 eller et bildekort er en «ekte» blackjack
- Du vinner på én av tre måter:
  - Du får ekte blackjack uten at dealer gjør det samme,
  - Du oppnår en høyere hånd enn dealer uten å overstige 21, eller
  - Din hånd har verdi mindre enn 21, mens dealerens hånd overstiger 21

Vi forventer *ikke* at du støtter flere spillere, en spiller og en dealer er tilstrekkelig. Du velger selv hvor mange klasser, funksjoner eller andre typer du ønsker å definere utover klassen `Blackjack`. Det er også åpent for å legge til medlemsfunksjoner i klassene du allerede har definert så langt i øvingen.

Det finnes flere varianter av spillet og flere regler, hvis du ønsker det kan du utvide regelsettet og implementere bl.a. flere spillere, flere kortstokker og "doubling down".

Dersom noe er uklart, gjør en antakelse og diskuter med studentassistenten din.