Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

TDT4102 Prosedyre- og objektorientert programmering Vår 2017

Øving 5

Frist: 2017-02-17

Mål for denne øvingen:

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

Generelle krav:

- Bruk de eksakte navn og spesifikasjoner som er gitt i oppgaven.
- Det er valgfritt om du vil bruke en IDE (Visual Studio, Xcode), men koden må være enkel å lese, kompilere og kjøre.
- Lag et bibliotek for hver klasse. Et biblotek består av en header-fil og en implementasjons-fil.

Anbefalt lesestoff:

- Kapittel 2.2, 6, 7, 9.3 & 11.1 Absolute C++ (Walter Savitch)

Bakgrunn for oppgavene

I denne øvingen skal vi lære om enum-typer og strukturer ved å skrive kode for håndtering av en kortstokk, og bruke denne koden til å implementere 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 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 ikke distinksjonen mellom svarte og røde kort.

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.

Typenavnene dine skal derfor være på formen:

```
MyType
```

i motsetning til variabelnavn, som vanligvis er på formen

```
myVariable
```

og konstantar, som vanligvis har formen

```
MY_CONSTANT
```

En annen konvensjon du skal følge er at du skal lage ei headerfil (`.h`) og ei implementasjonsfil (`.cpp`) for hver klasse. Hvis klassen heter `Car`, skal du altså lage filene `Car.cpp` og `Car.h`, og disse skal inneholde all kode for klassen `Car`.

1 Enum-typer (10%)

I denne oppgaven skal du lage og bruke enum-typer (også kjent som enumerasjoner eller *enumerations* på engelsk). En variabel av en enum-type er kjent som en enum-variabel. Merk at verdiene i en enum-type er konstante og skal derfor skrives med store bokstaver.

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

Du kan lese mer om enum-typer på side 67-68 i læreboka (6th ed.). Der står det også noe om *strongly typed enums* (også kalt *enum classes*), som er nytt i C++11. Det er ikke påkrevd å bruke dette ettersom boka er svært mangelfull i beskrivelsen av denne typen, men er valgfritt for de som ønsker å skrive mer moderne C++.

a) Lag enum-typen `Suit`. Definisjonen skal ligge i headerfila `card.h`.

`Suit` skal representere *fargen* til et kort, og kan ha verdiene CLUBS, DIAMONDS, HEARTS og SPADES.

b) Lag enum-typen `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) Skriv funksjonen `suitToString()`.

`suitToString` skal ta en `Suit` som argument og returnere en `string` som representerer verdien. Bruk en `switch` til å velge riktig tekst for riktig `Suit`. Husk å plassere prototypen i headerfila.

Hint: En variabel som er av en enum-type er fremdeles en variabel, og skal derfor begynne med liten bokstav.

d) Skriv funksjonen `rankToString()`.

`rankToString` skal ta inn en `Rank` som argument og returnere en `string` som representerer `Rank`-en som tekst. Bruk en `switch`-struktur til å velge riktig tekst for riktig `Rank`.

2 Struktur (10%)

I denne delen skal du lage og bruke strukturer.

a) Lag strukturen (`struct`) `CardStruct` i `card.h`. Strukturen skal ha følgende variabler:

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

b) Skriv funksjonen `toString()`.

`toString` skal ta et kort (en `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) Skriv funksjonen `toStringShort()`.

`toStringShort` skal ta et kort (en `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. *Hint: Du kan hente ut en del av en tekststreng med `.substr()`.*

Nyttig å vite: Konvertering fra tall til tekstreng

Dersom du ønsker å konvertere en `int` til en `string` var ikke dette så lett før C++11. Da måtte man ofte lage en egen funksjon for å få til dette:

```
#include <sstream>
std::string intToString(int number){
    std::stringstream ss;
    ss << number;
    return ss.str();
}

// kan nå konvertere til std::string
std::string s = intToString(10); // s inneholder nå "10"
```

I C++11 har vi fått funksjonen `std::to_string` (i `string`–biblioteket)som tar inn et tall, og returnerer en `std::string`. Dermed trenger vi ikke gjøre noe mer en dette:

```
std::string s = std::to_string(10);
```

Merk: *`std::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: http://en.cppreference.com/w/cpp/string/basic_string/to_string.*

d) Test begge strukturene og funksjonene dine fra `main`-funksjonen.

3 Kortklasse (20%)

I denne deloppgaven skal du lage en klasse `Card` med grunnleggende funksjoner. Denne klassen kommer til å bli brukt videre i øvingen.

a) Lag 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 invalid`, en boolsk variabel som skal indikere hvorvidt `s` og `r` har fått tilegnet verdier eller ikke, dvs. hvorvidt kortet er *(u)gyldig*.

b) Skriv medlemsfunksjonen `initialize(Suit s, Rank r)`.

Denne funksjonen skal sette `Suit s` og `Rank r`. I tillegg skal `invalid` settes til `false`. Funksjonen skal være `public`, ta inn en `Suit` og en `Rank` og ikke returnere noe.

Hint: Ettersom parametrene til `initialize` har samme navn som medlemsvariablene i `Card`, kan det være lurt å bruke `this`-pekeren for å få tak i medlemsvariablene.

c) Skriv medlemsfunksjonen `getSuit()`.

Denne funksjonen skal ikke ta inn noe, og skal returnere kortets `Suit`. Funksjonen skal være `public`.

d) Skriv medlemsfunksjonen `getRank()`.

Denne funksjonen skal ikke ta inn noe, og skal returnere kortets `Rank`. Funksjonen skal være `public`.

e) Skriv medlemsfunksjonen `toString()`.

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

Hint: Her skal du gjenbruke kode du har skrevet i Oppgave 1.

f) Skriv medlemsfunksjonen `toStringShort()`.

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

Hint: Her skal du gjenbruke kode du har skrevet i Oppgave 1.

g) Skriv konstruktøren `Card()`.

Denne konstruktøren skal ikke ta inn noen verdier. Dette gjør at vi ikke kan sette opp kortets `Suit` og `Rank` riktig, noe som skal indikeres ved at `invalid` skal settes til `true`.

h) Skriv konstruktøren `Card(Suit s, Rank r)`.

Denne konstruktøren skal ta inn `Suit` og `Rank`, og `invalid` skal settes til `false`. Bruk gjerne funksjoner du har laget tidligere for å forenkle koden for denne konstruktøren.

4 Kortstokklasse (20%)

I denne deloppgaven skal du implementere klassen `CardDeck`, som bruker klassen `Card` for å representere en kortstokk. Du skal deretter implementere enkel funksjonalitet for `CardDeck`.

a) Lag klassen `CardDeck`.

Klassen skal inneholde følgende `private` medlemsvariabler:

- `cards`, en `std::vector` bestående av 52 `Card`-objekter, som representerer en kortstokk.
- `currentCardIndex`, et heltall som brukes til å holde styr på hvor mange kort som er blitt delt ut.

Hint: I kap. 7.3 i boka står det du trenger å vite om `std::vector`.

b) Lag 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) Lag medlemsfunksjonen `swap()`.

Denne funksjonen skal ta inn to indekser (heltall) til `cards`-tabellen og bytte om på kortene som finnes ved disse to posisjonene. Funksjonen skal være `private`.

d) Lag 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) Lag 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) Lag medlemsfunksjonen `shuffle()`.

Denne funksjonen skal stokke kortstokken, dvs. plassere kortene i tilfeldig rekkefølge i `cards`-tabellen. Bruk gjerne `swap()` når du implementerer denne.

g) Lag 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.

Hint: Bruk currentCardIndex.

5 Blackjack (40%)

I denne oppgaven skal vi implementere en klasse `Blackjack`, som skal brukes til å spille det populære kortspillet Blackjack. Reglene til Blackjack vil bli forklart, men du kan også lese om dem blant annet på Wikipedia.

I Blackjack spiller en *dealer* mot en spiller. Hvert spill begynner med at både spilleren og dealeren får utdelt to kort. Kortene som tilhører spilleren og dealeren kalles en «hånd». Det første kortet dealeren får utdelt er synlig både for spilleren og dealeren, mens det andre kortet er kun synlig for dealeren.

Hvert kort har en viss *verdi*. I Blackjack har kortene 2-10 samme verdi som tallet på kortet, mens bildekortene (knekt, dame og konge) har verdi 10. Ess kan ha både 1 og 11 som verdi, og verdien bestemmes etter hva personen med esset i hånden ønsker.

Dersom spilleren ikke er fornøyd med kortene sine, kan han trekke et kort fra toppen av kortstokken, med det forbehold om at han umiddelbart taper dersom den totale kortverdien overstiger 21. Når spilleren har trukket og ikke har tapt, eller sagt pass (dvs. latt være å trekke), skal dealeren trekke dersom den totale kortverdien hans er under 17. Som for spilleren har dealeren umiddelbart tapt dersom den totale kortverdien hans overstiger 21. Deretter begynner en ny runde, der spilleren igjen har mulighet til å trekke og der dealeren skal trekke dersom han fortsatt har en total kortverdi under 17.

Når dealeren har en total kortverdi på minst 17 og spilleren ikke ønsker å trekke lenger, og verken spilleren eller dealeren har tapt fordi deres totale kortverdi har oversteget 21, avgjøres spillet avhengig av spillerens og dealerens totale kortverdier. Spilleren vinner dersom han har «blackjack» uten at dealeren har det, eller dersom den totale verdien til kortene hans overstiger den totale verdien til dealerens kort. «Blackjack» betyr at de to første kortene til spilleren/dealeren er et ess og enten 10 eller et bildekort (med andre ord total kortverdi på 21 fra de to første kortene).

Eksempler:

- Dersom spillet slutter med at spilleren har en total kortverdi på 20 og dealeren har en total kortverdi på 19, har spilleren vunnet.
- Dersom spillet slutter med at spilleren har et ess og en konge (totalverdi 21), og dealeren har åtte, syv og seks (totalverdi 21), har spilleren vunnet fordi han har «blackjack».
- Dersom spillet slutter med at spilleren har syv og dame, og dealeren har syv og konge, taper spilleren fordi dealeren har like høy totalverdi.

Dersom du føler at noe er uklart, ta en antakelse og diskuter med studentassistenten din.

Ettersom det finnes ulike former for Blackjack er vi ikke strenge på nøyaktig hvilken variant som velges. Du står også fritt til å avvike fra oppgaven så lenge det endelige resultatet virker omtrent likt (eller bedre).

a) Lag klassen `Blackjack`.

Klassen skal inneholde følgende `private` medlemsvariabler:

- `deck`, en variabel av typen `CardDeck`.
- `playerHand` og `dealerHand`, to variabler av typen `int` som holder orden på de totale verdiene til spillerens og dealerens kort.
- `playerCardsDrawn` og `dealerCardsDrawn`, to variabler av typen `int` som holder orden på hvor mange kort spilleren og dealeren har trukket.

b) Lag medlemsfunksjonen `isAce()`.

Denne skal ta inn en peker til et kort og returnere en `bool`-verdi som indikerer om kortet er et ess (`true`) eller ikke (`false`).

c) Lag medlemsfunksjonen `getCardValue()`.

Denne skal ta inn en peker til et kort og returnere verdien kortet har i Blackjack som et heltall. Vi håndterer foreløpig det at ess kan ha to ulike verdier ved å returnere -1 som verdi for ess.

d) Lag medlemsfunksjonen `getPlayerCardValue()`.

Denne skal ta inn en peker til et kort og returnere verdien kortet har for spilleren i Blackjack. Dersom kortet er et ess skal funksjonen spørre spilleren om kortet skal ha verdien 1 eller 11. Du skal bruke `getCardValue` og `isAce` til å implementere denne funksjonen.

I denne funksjonen må du passe på å validere brukerens respons, slik at han ikke kan velge at et ess kan ha en annen verdi enn 1 og 11.

e) Lag medlemsfunksjonen `getDealerCardValue()`.

Denne funksjonen skal ta inn en peker til et kort i dealerens hånd og et heltall som representerer verdien til dealerens hånd *utenom* det aktuelle kortet. Funksjonen returnerer verdien til kortet gitt følgende regler:

- Hvis kortet er et ess skal verdien til kortet regnes som 11, med mindre dette gjør at verdien til dealerens hånd *inkludert* kortet går over 21. I så fall skal verdien til kortet regnes som 1.
- For alle andre kort enn ess skal funksjonen returnere kortets verdi.

f) Lag medlemsfunksjonen `askPlayerDrawCard()`.

Denne funksjonen skal ikke ta inn noe, men skal spørre om hvorvidt spilleren ønsker et nytt kort. Dersom spilleren vil ha et nytt kort, skal funksjonen returnere `true`, og ellers skal den returnere `false`.

g) Lag medlemsfunksjonen `drawInitialCards()`.

Denne funksjonen skal ikke ta inn eller returnere noe, men skal trekke de to første kortene til spilleren og dealeren og oppdatere `playerHand`, `dealerHand`, `playerCardsDrawn` og `dealerCardsDrawn`. Den skal også skrive ut til skjerm det første kortet som blir trukket til dealeren. Pass på å håndtere ess riktig når kortet trekkes.

Vi skal nå sette sammen alle funksjonene vi har laget med mål om å kunne spille Blackjack etter reglene beskrevet tidligere.

h) Lag medlemsfunksjonen `playGame()`.

Denne funksjonen skal la brukeren spille Blackjack mot en dealer. Noen tips:

- Som i forrige deloppgave er det viktig at du håndterer ess riktig.
- Ikke glem at dealeren skal spilles automatisk – spilleren skal kun bestemme hva han selv skal gjøre.
- Når du sjekker de ulike mulighetene for seier og tap etter at spillet er ferdig, er det viktig at du gjør dette i riktig rekkefølge (tenk `if`/`else if`).

Husk å stokke kortstokken før spillet begynner!