Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

TDT4102 Prosedyre- og objektorientert programmering Vår 2023

Øving 5

Frist: 2023-02-17

Mål for denne øvingen:

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

Generelle krav:

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

Anbefalt lesestoff:

- Kapittel 9

Bakgrunn for oppgavene

I denne øvingen skal vi lære å definere egne typer. Vi skal bruke enum-typer og klasser til å definere en kortstokk. Disse typene skal vi 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 vær *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 bør være på formen:

`MyType`

i motsetning til variabelnavn, som vanligvis er på formen

`myVariable` eller `my_variable`.

I øvingene brukes det en annen programmeringsstil for navngivning av get- og set-funksjoner enn læreboka. I øvingene bruker vi `Type getMember()` der boka bruker `Type member()`, og `void setMember()` der boka bruker `void member()`. Du kan selv bestemme hvilken programmeringsstil du vil benytte. Fordelen med varianten som blir brukt i øvingsopplegget er at man slipper å finne på to navn til hver variabel i en klasse som trenger get- og set-funksjoner. Det er også denne varianten som er mest utbredt.

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 (kildefil). 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%)

Nyttig å vite: Scoped enum

Enumerasjon, eller enum, er en enkel, brukerdefinert type der hvert element (enumerator) blir representert med et unikt tall. I denne oppgaven skal vi bruke *scoped enums*. Det er praktisk da vi slipper å være forsiktig med navngiving, i tillegg til at det gir typesikkerhet.

For å opprette en scoped enum er syntaksen:

```
enum class Name { enumerator0 = x, enumerator1, enumerator2 = y, /*osv.*/};
```

Man kan velge å ikke bruke "= x" delen av koden, da vil enumerator0 automatisk få tall-verdien 0. Om man tar i bruk "= x" vil enumerator1 få verdien x+1. Man kan også hoppe i verdier som vist med enumerator2.

Eks.: Slik oppretter vi en scoped enum med månedene hvor januar har verdien 1, februar vedien 2, osv.

```
enum class Month {january = 1, february, march, april, may, june, /*osv.*/};
```

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 `::`. Eks. `Month::may`. `::` kalles *scope resolution operator* og gjør at vi får tilgang til elementene som ligger i scopet.

Dersom man skal deklarere en variabel av type `Month` fra tallverdien kan man gjøre det på følgende måte:

```
Month birthday = static_cast<Month>(4);
```

Dersom man vil gå motsatt vei, gjør man det på følgende måte:

```
int birthayInt = static_cast<int>(Month::april);
```

Siden enumeratorer representeres med unike tallverdier vil Month::july $\neq$ Day::sunday, selv om de begge to representerer den syvende måneden og den syvende dagen, det gjør jobben som programmerer mye enklere. Se §9.5 i læreboken for mer informasjon om enums.

Merk: For at det skal være mulig å inkludere enum-typer i flere kildefiler, 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`.

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`. Denne enum-typen skal også ligge i `Card.h`.

Tips: Det kan lønne seg å gi enumerasjonene tallverdier som stemmer overens med verdien de representerer. Se §9.5 i læreboken for hvordan dette kan gjøres.

c) Definer funksjonen `suitToString()`.

Funksjonen skal ta inn en `Suit` og returnere en `string` som representerer `Suit`-en som tekst. F.eks: `Suit::spades` skal bli strengen "Spades".

Tips: Her kan det være nyttig å bruke et `map`. Se Infobanken i VS-Code for informasjon om hvordan dette kan brukes.

d) Definer funksjonen `rankToString()`.

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

e) Teori.

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

f) Test det du har gjort hittil

En måte du kan teste det du har gjort til nå er å opprette et par `Suit`- og `Rank`-variabler i `main()`. Og så kalle funksjonene med disse som parameter. F.eks.

```
int main(){
    Rank r = Rank::king;
    Suit s = Suit::hearts;
    string rank = rankToString(r);
    string suit = suitToString(s);
    cout << "Rank: " << rank << " Suit: " << suit << '\n';
}
```

2 Kortklasse (25%)

I denne deloppgaven skal du lage en klasse `Card` med grunnleggende funksjoner.

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

Nyttig å vite: Klasse-syntaks

I headerfilen (.h/.hpp) deklarerer man klassen.

```
class Person {
private:
    int age; // Medlemsvariabel
    string name; // Medlemsvariabel
public:
    Person(int a, string n);// Konstruktør
    void setAge(int a);// Medlemsfunksjon
};
```

I implementasjonsfilen (.cpp) definerer man konstruktører og medlemsfunksjoner.

```
Person::Person(int a, string n): age{a}, name{n} // Initialiseringsliste
{}

void Person::setAge(int a) {
    age = a;
}
```

Merk at vi bruker `::` til å indikerer at funksjonene vi definerer er en del av klassen `Person`

Eksempel på kall i `main`

```
int main(){
    Person p{20, "Bob"}; // Konstruerer ett objekt av klassen Person
                         // (også kalt en instans av klassen Person)
    p.setAge(21); // Kaller medlemsfunksjonen setAge() på instansen
}
```

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.

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

Denne konstruktøren skal ta inn variablene `suit` og `rank` av typene `Suit` og `Rank`.

Konstruktøren skal initialisere medlemsvariablene `s` og `r` for objektet. Det kan gjøres med en *initialiseringsliste* (*eng: member initializer list*), se læreboken §9.4.4.

c) Definer medlemsfunksjonen `getSuit()`.

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

d) Definer medlemsfunksjonen `getRank()`.

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

e) Definer medlemsfunksjonen `toString()`.

Funksjonen skal returnere en representasjon av kortet i form av en `string`-variabel. For eksempel «Ace of Spades» (engelsk) eller «spar ess» (norsk). Funksjonen skal være `public` og ikke ta inn noe.

f) Test klassen din

Nå er et godt tidspunkt å teste om klassen fungerer sånn den skal. Det kan du gjøre ved å opprette et par `Card`-objekter, og kalle de `public` medlemsfunksjonene på objektene. F.eks:

```
int main(){
    Card c{Suit::spades, Rank::ace};
    cout << c.toString() << '\n';
    return 0;
}
```

3 Kortstokklasse (25%)

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 den `private` medlemsvariabelen `cards`, en `vector` som kan holde `Card`-objekter. Dette er beholderen som skal holde alle kortene i kortstokken.

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». En konstruktør uten parametere kalles en default-konstruktør.

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 `string`-representasjonen du har definert tidligere til å skrive ut hvert kort.

e) 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()`?

Det er ikke så nøye hvilken metode du bruker for å stokke kortstokken, men resultatet bør være (pseudo-)tilfeldig og kortene bør være godt blandet.

f) Definer medlemsfunksjonen `drawCard()`.

Denne funksjonen skal trekke det «øverste» kortet i kortstokken. Funksjonen skal returnere det siste elementet fra `cards`-vektoren, og i tillegg fjerne dette elementet fra vektoren.

4 Blackjack (40%)

Til nå har du fått beskrevet hvordan funksjoner, klasser og programmer skal designes. I denne oppgaven skal du få prøve deg litt mer på egenhånd. Formålet er at du skal få trening i å definere egne funksjoner og klasser. Alternativt kan oppgaven løses ved hjelp av deloppgavene på neste side i oppgaveteksten. Du velger selv om du vil løse oppgaven på egenhånd eller ved hjelp av deloppgavene.

Selv om du ikke må fullføre hele spillet for å få godkjent hele øvingen anbefales du å forsøke. Det kan være litt vanskelig, 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 vil derimot 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.

a) Lag klassen `Blackjack`.

Klassen skal inneholde følgende `private` medlemsvariabler:

- `deck`, en variabel av typen `CardDeck`.
- `playerHand` og `dealerHand`, to variabler av typen `Card-vector` som kan holde `Card`-objekter.
- `playerHandSum` og `dealerHandSum`, to variabler av typen `int` som holder orden på de totale verdiene til spillerens og dealerens kort.

b) Lag hjelpefunksjonen `isAce()`.

Denne skal ta inn 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 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 11 som verdi for ess.

d) Lag medlemsfunksjonen `getHandScore()`.

Denne skal ta inn en vektor med kort og returnere poengsummen disse gir, som et heltall. Verdien til et ess er enten 1 eller 11, alt ettersom hva som gir best poengsum. For eksempel skal et ess og en konge gi 21 poeng, mens to ess skal gi 12 poeng. Du skal bruke `getCardValue()` og `isAce()` til å implementere denne funksjonen.

e) 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`.

f) Lag medlemsfunksjonene `drawPlayerCard()` og `drawDealerCard()`.

Disse funksjonen skal ikke ta inn eller returnere noe, men skal trekke et kort og legge til i henholdsvis spillerens og dealerens hånd, samt oppdatere `playerHandSum` og `dealerHandSum`.

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

g) Lag medlemsfunksjonen `playGame()`.

Denne funksjonen skal la brukeren spille Blackjack mot en dealer. Funksjonen skal overordnet gjøre som følger:

- Så lenge poengsummen er mindre enn 21 skal spilleren få valget om å trekke et nytt kort.
- Deretter skal dealeren trekke kort til poengsummen er minst 17. Husk at dealeren skal spilles automatisk.
- Til slutt skal vinneren bestemmes. 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!

Om du ønsker å kjøre spillet eller teste underveis kan dette gjøres på følgende måte:

```
int main(){
    Blackjack b;
    b.playGame();
    return 0;
}
```