Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

TDT4102 Prosedyre- og objektorientert programmering Vår 2021

Øving 5

Frist: 2021-02-19

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. Til det brukes enum-typer 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 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. 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

Enumerasjoner er enkle, brukerdefinerte typer med et sett enumeratorer (verdier). Enumeratorene er symbolske konstanter. I denne oppgaven skal vi bruke *scoped enums*. Det er nyttig, da vi slipper å være veldig forsiktig med navngivning og 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`. `::` kalles *scope resolution operator* siden den brukes for å velge hvilket skop navn skal finnes i. 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. Se også §9.5 i læreboken.

For å opprette en scoped enum er syntaksen:

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

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.

Nyttig å vite: map

`map` er en assosiativ beholder. Hvert element i et `map` er et par, som består av en nøkkel og en verdi. Nøklene må være er unike, men verdiene kan være like. I tillegg må det være mulig å ordne(sortere) nøklene, siden elementene i et `map` blir ordnet etter nøklene. Et eksempel på et `map` er:

```
map<string, int> myMap; // her er nøkkelen en string og verdien en int
```

Vi kan legge til elementer i `map`-et med `[]`-operatoren.

```
myMap["five"] = 5;
```

Vi kan aksessere elementer med `[]`-operatoren og `at()`.

```
int i1 = myMap["five"];
int i2 = myMap.at("five");
```

Merk: Hvis vi har et konstant `map` kan vi kun bruke `at()` for å aksessere elementer. Vi kan også legge til elementer når vi deklarerer et `map`.

```
map<string,int> example_map{{"two", 2}, {"five", 5}};
```

Vi kan iterere gjennom et `map` med en «for each»-løkke.

```
for(const auto& m:myMap){
    cout << "key: " << m.first << " value: " << m.second << ", ";
}// vil gi utskriften: key: five value: 5, key: two value: 2,
```

Du kan lese om `map` i forelesningsnotater og §21.6 og B.4 i læreboken.

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". Denne oppgaven kan løses ved bruk av et `map`, `map<Suit, string>`.

d) Definer funksjonen `rankToString()`.

Gjør tilsvarende som i *c)*, for `Rank`, `map<Rank,string>`. 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) Definer medlemsfunksjonen `toStringShort()`.

Denne funksjonen skal returnere en *kort og kompakt* representasjon av kortet i form av en `string`-variabel, 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. Funksjonen skal være `public` og ikke ta inn noe.

Nyttig å vite: Konvertering fra tall til tekststreng

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.*

g) 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{Rank::ace, Suit::spade};
    cout << c.toString() << '\n';
    cout << c.toStringShort() << '\n';
}
```

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-vector`en 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()`?

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.

g) 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.

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 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.