NTNU
Institutt for datateknikk
og informasjonsvitenskap

Kontinuasjonseksamen i

TDT4102 - Prosedyre- og objektorientert programmering

Fredag 17 august 2012, 09:00

Kontaktperson under eksamen: Trond Aalberg (97631088)

Eksamensoppgaven er utarbeidet av Trond Aalberg

Språkform: Bokmål

Tillatte hjelpmidler (hjelpemiddelkode C):
Walter Savitch, Absolute C++ eller Lyle Loudon, C++ Pocket Reference.
Bestemt, enkel kalkulator tillatt.

Sensurfrist: Fredag 7 september.

Generell introduksjon

Les gjennom oppgavetekstene nøye. Noen av oppgavene har lengre tekst, men dette er for å gi kontekst, introduksjon og eksempler til oppgaven.

Når det står "implementer" eller "lag" skal du skrive en implementasjon. Hvis det står "vis" eller "forklar" står du fritt i hvordan du svarer, men bruk enkle kodelinjer og/eller korte forklaringer og vær kort og presis. Dersom du mener at opplysninger mangler i en oppgaveformulering, gjør kort rede for de antagelser og forutsetninger som du finner nødvendig.

Hver enkelt oppgave er ikke ment å være mer omfattende enn det som er beskrevet. Noen oppgaver fokuserer bare på enkeltfunksjoner og da er det utelukkende denne funksjonen som er tema. Andre oppgaver er "oppskriftsbasert" og vi spør etter forskjellige deler av en klasse, samarbeidende klasser eller et program. Du kan velge selv om du vil løse dette trinnvis ved å ta del for del, eller om du vil lage en samlet implementasjon. Sørg for at det går tydelig frem hvilke spørsmål du har svart på hvor i koden din.

Det er ikke viktig å huske helt korrekt syntaks for biblioteksfunksjoner. Oppgaven krever ikke kjennskap til andre klasser og funksjoner enn de du har blitt kjent med i øvingsopplegget. All kode skal være i C++.

Oppgavene teller med den andelen som er angitt i prosent. Den prosentvise uttellingen for hver oppgave kan/vil likevel bli justert ved sensur basert på hvordan oppgavene har fungert. De enkelte deloppgaver kan også bli tillagt forskjellig vekt.

NB! I denne denne versjonen av eksamensoppgaven har jeg rettet opp noen mindre feil/uklarheter som var i den opprinnelige oppgaven. Alle korreksjoner ble annonsert under eksamen.

Oppgave 1: Oppvarming (20%)

a) Implementer en funksjon **`double mean(int arr[], int size)`** som regner ut det aritmetiske gjennomsnittet av verdiene i en tabellen **`arr`**. Det aritmetiske gjennomsnittet finner du ved å dele summen av alle verdiene med antallet verdier. Merk at resultatet skal bli et flyttall selv om inputverdiene til funksjonen er heltall. Parameteren **`size`** er størrelsen på tabellen (antallet verdier i tabellen).

Eksempel: For verdiene {2,2,2,2,4,4,4,4} blir det aritmetiske gjennomsnittet 24 / 8 = 3.0 For verdiene {2, 3, 4, 5} blir det aritmetiske gjennomsnittet 14 / 4 = 3.5

b) I statistikk er median den verdien som ligger i midten av en sortert serie med verdier. Hvis antallet verdier er et oddetall, er medianen den midterste verdien. Hvis mengden av verdier er et partall er medianen gjennomsnittet av de to midterste verdiene. Implementer en funksjon **`double median(int arr[], int size)`** som finner medianen for verdiene i den sorterte tabellen **`arr`** (du trenger med andre ord ikke bekymre deg om sorteringen).

Eksempel: For verdiene {1, 1, 3, 4, 8} blir medianen 3.0 siden dette er tallet som ligger i midten. For tallene {1, 1, 3, 4, 5, 8} blir medianen 3.5 siden dette er gjennomsnittet av de to verdiene som ligger i midten (3 og 4).

c) Implementer en funksjon **`mode(int arr[], int size)`** som finner typetallet for verdiene i den sorterte tabellen arr. Typetallet er det tallet som forekommer oftest i en mengde av tall og for en tallrekke kan det være flere typetall. Du må selv bestemme hva som er en egnet returtype i denne deloppgaven (eller annen teknikk for å få svaret returnert).

Eksempel: For verdiene {1, 1, 2, 2, 3, 3, 3, 4, 5, 6} er typetallet 3 siden det er denne verdien som forekommer flest ganger. For verdiene {1, 1, 2, 2, 3, 3, 3, 4, 4, 4} er typetallene 3 og 4 siden begge disse verdiene forekommer like mange ganger.

Tips: Her er det selvsagt fritt frem å bruke klasser og funksjoner i C++ biblioteket.

Oppgave 2: Morro med objekter og pekere (30%)

Klassen **Person** under skal brukes for å lage et "nettverk av venner". Hver person kan ha en eller flere andre personer som venn (implementert ved at hvert Person-objekt har et set av pekere til andre Personer). Når Person **a** legger til Person **b** som venn ved hjelp av medlemsfunksjonen **connect**, er det meningen at også **b** sin venneliste oppdateres automatisk med en peker til **a**. Koden i **main** er et eksempel på bruken av funksjonen og skal ikke endres.

a) Hva er galt med implementasjonen av **connect**-funksjonen? Hvordan kan du implementere **connect** slik at den fungerer etter intensjonen?

```
class Person{
  private:
    string name;
    set<Person *> friends;
    vector<string *> msgs;
  public:
    Person(const string &name);
    void post(string msg);
    void receive(string *);
    void connect(Person *p){
      friends.insert(p);
      p->connect(this);
    };
};
```

```
int main(){
  Person *a = new person("Thea");
  Person *b = new Person("Tor");
  Person *c = new Person("Per");
  a->connect(b);
  a->connect(c);
  b->connect(c);
  a->post("er på konten");
  b->post("jeg også");
  c->post("ikke jeg");
}
//Nå skal alle være venner med alle
//og alle mottar alle meldinger som
//blir sendt
```

b) Implementer funksjonen **post** som brukes til å lage og poste meldinger til nettverket av venner og funksjonen **receive** som brukes for å motta meldinger fra venner. Kort beskrevet skal post-implementasjonen instansiere en dynamisk allokert string med innholdet fra msg-parameteren og iterere over alle vennene og gjøre funksjonskallet receive på disse. Meldinger som en person har skrevet selv eller mottatt fra andre lagres i variabelen **msgs**.

c) Hva skjer med dette nettverket av pekere hvis du sletter en person? Forklar eller vis hvordan du vil implementere en destruktør for **Person**-klassen.

d) Implementer funksjonalitet i Person-klassen slik at det ikke er mulig å ha flere Person-instanser med samme navn. Hvis du prøver å lage et nytt objekt mens det finnes et annet objekt som har samme personnavn, skal det kastes et unntak av typen **NameInUse**.
Hvis du har behov for å finne en verdi i en container kan du bruke find i <algorithm>som litt forenklet ser slik ut: **iterator find(iterator first, iterator last, T value)** *Den returnerer en iterator til verdien som er funnet, eller last hvis den ikke finner verdien. Det finnes også en egen find-funksjon for set:* **iterator set::find(T value)**. *Denne returnerer samme iterator som set::end() hvis verdien ikke finnes.*

Oppgave 3: Et litt større prosjekt (50%)

I denne oppgaven skal du implementere noen klasser og funksjoner som er inspirert av spillet Scrabble /Wordfeud. Dette er et spill hvor spillerne etter tur lager ord med bokstavbrikker på et brett med 15x15 felter. Detaljene og reglene du trenger å kjenne til blir beskrevet etter hvert i deloppgavene, og vi har utelatt det som ikke er relevant for oppgavene du skal løse. Du finner også en beskrivelse av spillet i vedlegget til oppgaven.

a) Implementer en klasse kalt **`Tile`** for bokstavbrikkene. En brikke har en bokstavverdi og en heltallsverdi som begge skal være innkapslet. Bokstav og tallverdi skal settes med konstruktøren og skal kunne leses, men ikke endres etter instansiering. Bruk initialiseringsliste for å sette verdiene. Eksempel på bruk av konstruktøren finner du i koden under.

b) Overlagre operatorene **`+`** og **`+=`** for **`Tile`**-klassen slik at det blir enkelt å summere tallverdiene av en samling brikker. Eksempel på bruk av begge operatorene er vist under.

```
vector<Tile> tiles;
tiles.push_back(Tile('K', 3));
tiles.push_back(Tile('O', 3));
tiles.push_back(Tile('N', 1));
tiles.push_back(Tile('T', 1));
```

```
int sum1 = 0, sum2 = 0;
for (int i = 0; i < tiles.size(); i++){
   sum1 += tiles[i];
}
for (int i = 0; i < tiles.size(); i++){
   sum2 = sum2 + tiles[i];
}
```

Brettet som brukes i Scrabble og Wordfeud har 15x15 felt. Noen av feltene er bonusfelt av typen DL, TL, DW og TW og gir ekstra poeng. For å representere brettet og implementere deler av spillereglene skal vi lage en klasse kalt **`Board`**.

c) Vis eller forklar medlemsvariabler og konstruktører du trenger for klassen **`Board`**. Hvert felt skal kunne adresseres med x,y koordinater, det skal være mulig å "legge" en brikke på et felt, brettet må vite om et felt er et bonusfelt eller ikke, du må kunne sjekke om et felt er tomt eller om det allerede ligger en brikke der og eventuelt hvilken brikke som ligger på feltet. TIPS: Her bør du lage en egendefinert datatype for felt som f.eks. har en enum-variabel for felttypen og en peker til en brikke.

d) Implementer følgende medlemsfunksjoner i Board-klassen:

```
bool placeTile(Tile *t, int x, int y);
bool play();
```

*Spilleren som har tur legger én og én brikke ved hjelp av **`placeTile`**. Spilleren kan legge på et hvilket som helst felt hvor det ikke ligger noen brikke fra før. Når spilleren er ferdig med å legge brikker gir han beskjed til brettet med **`play()`**. Denne funksjonen tester om brikkene er lagt riktig ved hjelp av medlemsfunksjonen **`bool isValidPlay()`** - som du skal implementere i deloppgaven under. Hvis brikkene er lagt riktig returnerer play() **`true`**. Hvis ikke skal alle brikker som ble lagt i løpet av spilleren sin tur fjernes igjen fra brettet og play() skal returnere **`false`**. TIPS: Her kan det være greit at klassen "husker" hvilke felt en spiller legger brikker på, slik at det er enkelt å fjerne brikkene igjen fra brettet.*

e) Implementer funksjonen **`bool Board::isValidPlay()`** som ble brukt i oppgaven over. Den skal sjekke at spillets regler for plassering av brikker er fulgt. Funksjonen returnerer **`true`** hvis brikkene er lagt på lovlige plasser, og **`false`** hvis ikke. Du kan basere deg på skissen til løsning som er beskrevet på neste side. NB! Her er det viktig at du dekomponerer løsningen i mindre funksjoner og viser hvordan disse kombineres til en helhetlig løsning.

Testingen på om brikkene er lagt riktig gjøres etter at en spiller har lagt ferdig sine brikker på brettet. Hvis du har fulgt tipset i oppgave d) har du også en variabel som holder styr på hvilke felt en spiller har lagt på. Da kan du sjekke med brettet at det ligger en brikke i midten og at alle brikker ligger inntil hverandre, og du kan bruke variabelen for "brikker som er lagt av spilleren" til å sjekke at alle brikker er på linje enten vertikalt eller horisontalt.

NB! Helt riktig logikk for å sjekke er litt underordnet, det viktigste er at du viser bruk av flere funksjoner som hver gjør sin del av jobben. Logikken som er beskrevet over er en litt forenklet løsning.

f) Ord som legges på brettet skal være vanlige, kjente ord. For å kunne sjekke om et ord er "lovlig", kan vi bruke ei ordbok. Implementer klassen **`Dictionary`** til dette formålet med medlemsfunksjonen **`bool isValid(const string &word)`**. Konstuktøren skal ta inn filnavnet og lese fra fila inn i en egnet datatype. I ordbok-fila er det brukt linjeskift for å skille mellom ordene.

g) Når en spiller har lagt sine brikker må vi finne ut om ordene som er laget er lovlige. Lag en medlemsfunksjon som finner alle ordene som en bruker har laget og sjekker om de er lovlige.

h) Og hvis du har tid igjen: implementer en funksjon for å regne ut poengsummen en spiller skal ha etter en runde. Poengene beregnes først for hvert ord og deretter summeres disse. Bonusfelt som brukeren legger på skal medregnes. DL gir dobbelt bokstavpoeng og TL gir trippelt bokstavpoeng. DW gir dobbel ordsum og TW gir trippelt ordsum. Bokstavbonus regnes ut først og deretter ordbonusene. Hvis en bokstav brukes i to ord skal den telles med i begge inklusive eventuell bokstavbonus.

Appendiks 1

Regler for Scrabble
(fritt etter de offisielle Scrabble-reglene og med forbehold om at det kan være feil)

Scrabble og Wordfeud spilles på et brett med 15x15 felt. En illustrasjon av det klassiske Scrabble-brettet finner du på siste side. Noen felt er såkalte bonusfelt av typen DL (double letter), TL (triple letter), DW (double word) og TW (triple word). Disse feltene gir ekstra poeng. Midten er markert med en stjerne siden det første ordet som legges må dekke midten. Utover dette har ikke stjerna noen spesiell funksjonalitet. Wordfeud er en variant av Scrabble tilpasset smartphones og pads. I Wordfeud kan du også velge et random-brett hvor bonusfeltene er tilfeldig plassert.

I Scrabble brukes 100 bokstavbrikker og de forskjellige bokstavene har forskjellig verdi. Eksempelvis har bokstavene A, N, R, S, T verdien 1 siden de er lette å bruke til å lage ord, mens D og G har verdien 2, K har verdien 5 osv. Distribusjon av bokstavbrikker og verdien til bokstavene kan varierer fra språk til språk. Wordfeud har sitt eget oppsett som avviker litt fra det klassiske spillet. Det finnes også to blanke brikker som kan brukes som hvilken som helst bokstav, men blanke brikker har ingen verdi (eller verdien 0).

Den første spilleren legger to eller flere av sine brikker slik at de former et ord enten bortover eller nedover. Diagonale ord er ikke tillatt og første ord må dekke midterste felt (det som er merket med stjerne). Alle brikkene som den første spilleren legger må selvsagt ligge inntil hverandre.

Alle ord som legges skal være gyldige ord. Når man spiller brettspillet må man avtale på forhånd hva som er lovlige ord og man kan f.eks. bruke en ordbok for å avgjøre hvis det er tvil. Spiller du Wordfeud på en smartphone eller pad vil programmet automatisk sjekke om ordene som legges er lovlige ved at det gjøres et oppslag i ei ordliste.

Neste spiller legger en eller flere bokstaver slik at de former et eller flere nye ord eller endrer på ord som allerede er lagt. Alle bokstaver som legges må plasseres på samme linje enten bortover eller nedover og alle brikker må ligge inntil en annen brikke. Spilleren kan legge til bokstaver før og etter brikker som allerede ligger på brettet, eller legge brikker parallelt med ord som allerede ligger på brettet. Slik kan det lages ett eller flere nye ord både horisontalt og vertikalt i samme runde. Se illustrasjon på slutten for eksempel på hvordan spillet kan utvikle seg. Den første spilleren legger ordet “SPANSK” slik at det dekker midtfeltet. Neste spiller legger bokstavene P, E og T slik at det former ordet “PENT” sammen med bokstaven forrige spiller la. Neste spiller legger bokstaven S og former ordene “SE” og “SA”. Siste spiller legger bokstavene I, E og T og får ordene “SEI”, “ISET” og “TE”.

Spillet har også regler om å melde pass, hvordan brikker skal trekkes, hvordan spillet skal avsluttes og mye mer, men de hopper vi over siden det ikke er relevant for oppgaven.

Poengene en spiller får etter sin tur er summen av bokstavene i alle ordene som er laget eller endret plus eventuelle ekstrapoeng hvis en eller flere brikker legges på et bonusfelt. DL gir 2x bokstavpoeng, TL gir 3x bokstavpoeng, DW gir 2x ordpoeng og TW gir 3x ordpoeng. Bokstavbonus regnes ut først. Bokstaver og bonusfelt som inngår i flere ord som en bruker lager, telles for hvert ord. Etter at det er lagt en brikke på et bonusfelt teller det ikke for andre spillere.

Spør faglærer som går eksamensrunden hvis du trenger en bedre forklaring på spillet. Oppgaven er ikke en øvelse i å forstå reglene og du skal få den hjelpen du trenger.

TW			DL				TW				DL			TW
	DW				TL				TL				DW	
		DW				DL		DL				DW		
DL			DW				DL				DW			
				DW						DW				
	TL				TL				TL				TL	
		DL				DL		DL				DL		
TW			DL				☆				DL			TW
		DL				DL		DL				DL		
	TL				TL				TL				TL	
				DW						DW				
DL			DW				DL				DW			
		DW				DL		DL				DW		
	DW												DW	
TW							TW							TW

1)

S	P	A	N	S	K

2)

			P		
			E		
S	P	A	N	S	K
			T		

3)

			P		
		S	E		
S	P	A	N	S	K
			T		

4)

			P		
		S	E	I	
S	P	A	N	S	K
			T	E	
				T	