NTNU
Institutt for datateknikk
og informasjonsvitenskap

Løsningsforslag til konteringseksamen i

TDT4102 - Prosedyre- og objektorientert programmering

Lørdag 8. august 2009

NB! Dette er en veiledende løsningsoppgave. Mange av oppgavene kan løses på forskjellige måter.

Eksamensoppgaven er utarbeidet av Trond Aalberg

Språkform: Bokmål

Tillatte hjelpmidler: Walter Savitch, Absolute C++ eller Lyle Loudon, C++ Pocket Reference

Sensurfrist: Mandag 29 august.

Generell intro

Les gjennom oppgaveteksten nøye og finn ut hva det spørres om. Noen av oppgavene har forklarende tekst som gir informasjon og kontekst for det du skal gjøre.

All kode skal være C++. Implementasjonene skal gi et mest mulig fullstendig svar på oppgaven, men du kan utelate kode som du har med i en tidligere oppgave eller som du mener ikke er relevant i forhold til det som er deloppgavens tema.

Dersom du mener at opplysninger mangler i en oppgaveformulering, gjør kort rede for de antagelser og forutsetninger som du finner det nødvendig å gjøre. Hver enkelt oppgave er ikke ment å være mer krevende enn det som er beskrevet.

Selv om vi i enkelte oppgaver ber om en funksjon, kan du lage hjelpefunksjoner der du finner at dette er formålstjenelig (f.eks. fordi det gjør det enklere å programmere eller gjør koden mer lesbar).

Enkelte oppgaver er formulert som spørsmål ("hva", "hvordan" etc). Her er vi ute etter korte svar og kode som viser hvordan du ville valgt å løse dette.

Oppgavene teller med den andelen som er angitt i prosent. Den prosentvise uttellingen for hver oppgave kan likevel bli justert ved sensur. De enkelte deloppgaver kan også bli tillagt forskjellig vekt. De enkle oppgavene vil telle noe mindre enn de mer vanskelige.

Oppgave 1: Litt av hvert - grunnleggende programmering (25%)

a) Hva er blir skrevet ut av koden under?

```
int x = 10;
double y = 4;
int z = x/y;
cout << "1: " << z << endl;
cout << "2: " << x/y << endl;
cout << "3: " << 5 % 3 << endl;
```

```
1: 2
2: 2.5
3: 2
```

b) Funsjonen under er en implementasjon av Euclids algoritme som kan benyttes for å finne største felles divisor for to tall - det vil si det største tallet som begge inputverdier kan deles med. Vi har lagt til en linje som skriver ut hvilke argumenter funksjonen ble kalt med og du skal vise hva følgende program vil skrive ut:

```
int gcd(int a, int b){
  cout << "gcd(" << a << ", " << b << ")" << endl;
  if ( b == 0 ){
    return a;
  }
  return gcd(b,a%b);
}

int main(){
  int a = gcd(32, 264);
  cout << a << endl;
}
```

```
gcd(32, 264)
gcd(264, 32)
gcd(32, 8)
gcd(8, 0)
```

Viktig å vise forståelse for rekursjon her. Helt riktig utregning er mindre viktig så lenge du viser funksjonskallene og klarer å følge rekursjonen.

c) Implementer en funksjon **`void printFraction(int numerator, int denominator)`** som skriver ut (til cout) resultatet av en divisjon som heltallet og den restrerende brøken. Utskriften skal være litt forskjellig avhengig av hvilke verdier som brukes (se eksemplene under). Du trenger ikke spesialhåndtere 0 eller negative verdier, men kan anta at funksjonen bare skal brukes på positive tall. Her kan det være nyttig å bruke funksjonen **`gcd()`** fra oppgaven over. Numerator er engelsk for teller og denominator engelsk for nevner.

```
printFraction(5, 6);
// Dette skal gi utskriften: "5/6"
printFraction(3, 3);
// Dette skal gi utskriften: "1"
printFraction(6, 4);
// Dette skal gi utskriften: "1 + 1/2"
```

Her er vi interessert i håndtering av forskjellige tilfeller ved hjelp av if-else samt bruk av heltallsdivisjon for å finne heltallet og modulus-operatoren for å finne rest-brøken. Brøken kan forenkles ved hjelp av gcd-funksjonen som finner det største tallet både teller og nevner kan deles med.

```
void printFraction(int nominator, int denominator){
  // Skriver ut heltallsdelen først (hvis det er en heltallsdel)
  if (nominator >= denominator) {
    cout << nominator/denominator;
    if ((nominator % denominator) != 0){
      // " + " skal bare skrives ut hvis det er en heltallsdel,
      // derfor tester vi og skriver ut " + " her og ikke i if-delen under
      cout << " + ";
    }
  }
  if ((nominator % denominator) != 0){
    //finner minste fellesnevner
    int d = gcd(nominator, denominator);
    cout << (nominator%denominator)/d << "/" << denominator/d;
  }
  cout << endl;
}
```

d) Implementer en funksjon **`bool isAnagram(char *, char *)`** som sjekker om to strenger er anagrammer av hverandre. Anagram (av gresk: ana og graphein = omskrive) er et ord, navn eller et fast uttrykk som er blitt satt sammen ved å stokke rundt på bokstavene i et annet ord eller uttrykk. Funksjonen skal ikke ta hensyn til skilletegn (mellomrom, komma, punktum etc.) og skal håndtere store og små bokstaver som like tegn. Eksempler:

"*I am Lord Voldemort*" er et anagram av "*Tom Marvolo Riddle*"
"*Balkongdjevelen Kim*" er et anagram av "*Kjell Magne Bondevik*"

Her er det mange mulige løsninger og vi er ute etter din evne til å tenke løsning, bruke funksjoner etc.

Forslaget under baserer seg på:

-kopiering av strengene til lower-case temp-strenger hvor vi bare beholder bokstavene
-sammenligne lengden på temp-strengene (må være like)
-sortere temp-strengene og teste om de er like bokstav for bokstav (etter sorteringen)

```
void sort(char str[], int size){
  for(int x=0; x < size; x++){
    for(int y=0; y < size - 1; y++){
      if(str[y] > str[y+1]){
        swap(str[y+1], str[y]);
      }
    }
  }
}

bool isAnagram(char* str1, char* str2){
  //kopierer strengene siden vi må manipulere de før vi kan teste
  char t1[256]; //eller vi kan rett og slett bare bruke vector<char>
  char t2[256];

  //copying only alph char. and converting to lowercase
  int s1 = 0;
  for (int i = 0; i < strlen(str1); i++){
    if (isalpha(str1[i])){
      t1[s1++] = tolower(str1[i]);
    }
  }

  int s2 = 0;
  for (int i = 0; i < strlen(str2); i++){
    if (isalpha(str2[i])){
      t2[s2++] = tolower(str2[i]);
    }
  }

  if (s1 != s2){ //tester på antall bokstaver
    return false;
  }

  //sorting
  sort(t1, s1); //bruker egendefinert alg. her, men kan også bruke
  sort(t2, s2); //sort-funksjoner i STL <algorithm>

  //comparing
  for (int i = 0; i < s1 && i < s2; i++){
    if (t1[i] != t2[i]){
      return false;
    }
  }
  return true;
}
```

Oppgave 2: Yatzee (25%)

Yatzee er et spill hvor du kaster 5 terninger og prøver å oppnå forskjellige kombinasjoner av tall for eksempel to like (et par), tre like, fire like, hus (to like + tre like) etc . For hver tur kan du kaste terningene tre ganger men du bestemmer selv hvilke terninger som du vil kaste om igjen. I denne oppgaven skal du lage variabler og funkjsoner for en klasse kalt **`Yatzee`** som stort sett skal representerer de 5 terningene som brukes i Yatzee og noe funksjoner relatert til spillet.

a) Hvilken medlemsvariabel vil du benytte for å representere de 5 terninger i klassen **`Yatzee`**?

Vis deklarasjonen av medlemsvariabelen og forklar kort ditt valg. Hver enkelt terning kan ha tallverdien 1-6, det skal være mulig å angi hvilke terninger som skal kastes/ikke-kastes og det skal være mulig å sortere terningene i stigende rekkefølge.

Her er det naturlig å bruke int terninger[5]. Her forventer vi bare deklarasjon av medlemsvariabelen, men for enkelthets skyld viser vi også klassedeklarasjonen her:

```
class Yatzee{
private:
   int values[5];
public:
   Yatzee();
   void roll();
   void roll(vector<int> pos);
   int getMultiples(int howmany, int excludevalue = 0);
   int getPair(int excludevalue = 0);
   int getThreeOfAKind(int excludevalue = 0);
   int getHouse();
};
```

b) Implementer en medlemsfunksjon **`void Yatzee::roll(...)`** som simulerer et terningkast (gir terningene tilfeldige verdier). Vi har bruk for en funksjon som kan kaste alle terninger eller bare utvalgte terninger. Du må selv bestemme evt. parameter til funksjonen.

Kan løses på flere måter. Hvis vi skal kaste bare utvalgte terninger kan vi ha en vector<int> som forteller hvilke terninger som skal kastes. Hvis alle terninger skal kastes kan vi ha en overlagret funkjson som ikke tar noe parameter.

```
void Yatzee::roll(){
//kaster alle terninger
for (int i = 0; i < 5; i++){
   values[i] = (rand() % 6) + 1;
}
}

void Yatzee::roll(vector<int> pos){
//kaster bare utvalgte terninger angitt med 1-5
for (int i = 0; i < pos.size(); i++){
   values[pos[i] - 1] = (rand() % 6) + 1;
}
```

```
}
```

c) Hvordan vil du sørge for at en instans av Yatzee-klassen bestandig har meningsfylte verdier for terningene (f.eks. selv før første bruker har utført det første terningkastet)? Med meningsfylte verdier mener vi at alle terningene alltid skal ha en verdier mellom 1-6. Vis og forklar kort hva du må implementere for å gjøre dette.

Her er det bruk av konstruktør vi er ute etter og en fornuftig løsning er f.eks. å kalle roll() i konstruktøren. Generelt er det mer fornuftig å starte med et sett tilfeldige verdier enn at alle terningene initielt har samme verdi (men så lenge man har en definert starttilstand er det OK).

```
Yatzee::Yatzee(){
  roll(); //roll gir tilfeldige tall 1-6
}
```

d) Etter at en bruker har kastet terningene tre ganger er det behov for en funksjon som kan sjekke resultatet og returnere en poengsum. Implementer funksjonene under samt eventuelle hjelpefunksjoner som kan være nyttige:

- **`int Yatzee::getPair()`**
  som finner det paret (terninger med to like verdier) blant terningene som har høyest verdi og returnerer summen av dette terningparet. Hvis det ikke finnes noe par blant terningene skal funksjonen returnere 0.

- **`int Yatzee::getThreeOfAKind()`**
  som finner tre like blant terningene og returnerer summen. Hvis det ikke finnes tre like blant terningene skal funksjonen returnere 0.

  Her ser vi etter kode som virker rimelig fornuftig: kan løses på mange måter.

  PS! I den opprinnelige eksamensoppgaven var klassenavnet feil i denne oppgaven. I evalueringen er det tatt hensyn til forvirringer og feil forårsaket av dette.

  I praksis kan disse funksjonene være ganske enkle. Løsningen under er basert på en generell parameterisert funskjon som kan finne n antall like verdier. Her har vi også tatt høyde for neste oppgave og lagt inn støtte for at vi skal kunne eksludere en enkelt verdi f.eks. for å kunne finne to like - men ikke to seksere.

```
int Yatzee::getMultiples(int howmany, int excludevalue){
  int sum = 0;
  for (int value = 1; value <=6; value++){
    int count = 0;
    for (int pos = 0; pos < 5; pos++){
      if ((value != excludevalue) && (values[pos] == value)){
        count += 1;
      }
    }
    if (count >= howmany){
      sum = value * howmany;
    }
  }
  return sum;
}

int Yatzee::getPair(int excludevalue){
  return getMultiples(2, excludevalue);
}

int Yatzee::getThreeOfAKind(int excludevalue){
  return getMultiples(3, excludevalue);
}
```

e) Lag en tilsvarende funksjon som i oppgaven over, men for å sjekke om en bruker har fått "fullt hus". Dette består i at terningene skal inneholde to like av ett tall + tre like av et annet tall, f.eks. to femmere og tre seksere.

- **`int Yatzee::getHouse()`**
  Finner to like + tre like verdier blant terningene og returnerer summen av disse. Hvis terningene ikke utgjør et hus skal funksjonen returnere 0.

  Her ser vi etter kode som virker rimelig fornuftig: kan løses på mange måter. Viktig her er å huske at par og tre like ikke skal være av samme terning-verdi. Det er også viktig at vi ikke returnerer par-verdien for det som er egentlig er tre like.

```
int Yatzee::getHouse(){
  int threeofakind = getThreeOfAKind();
  int pair = getPair(threeofakind / 3);
  //finner par som ikke har samme terningverdi som tre like
  if (threeofakind > 0 && pair > 0){
    return threeofakind + pair;
  }else{
    return 0;
  }
}
```

Oppgave 3: Generalisering (20%)

a) Terninger med 6 sider og tallverdiene 1-6 brukes i mange forskjellige spill.

- Lag en klasse kalt **`DiceGame`** som skal kunne brukes for forskjellige terningspill hvor antallet terninger kan variere fra spill til spill. Denne supertypen skal inneholde en medlemsvariabel for et variabelt antall terninger, en funksjon **`roll()`** for å kaste en eller flere av terningene (gi terningene en tilfeldig verdi - tilsvarende den du laget for klassen Yatzee i oppgave 2 b) samt andre funksjoner og variabler som er nødvendig hvis antallet terninger skal bestemmes dynamisk (ved instansiering).
- Vis hvordan du kan deklarere en subtype av denne som har et spesifikt antall terninger. Hvis for eksempel klassen Yatzee hadde vært en subtype, skulle denne hatt 5 terninger.

NB! Oppgave er uavhengig av det du evt. har implementert i oppgave 2, men du kan selvsagt referere til eller basere deg på kode du allerede har laget.

I Denne oppgaven må du parameterisere tabell-størrelsen og dermed er du nødt til å bruke en peker type int *dices, angi størrelse med int size og bruke konstruktøren for å sette størrelse, og allokere tabell-variabel i konstruktøren med dices = new int[size].

Et annet alternativ som er enklere er å bruke en vector<int>, men vi må likevel ha en variabel som sier hvor mange terninger vi skal ha.

```
class DiceGame{
private:
   int size;
   int *values;
   void sort();
public:
   DiceGame(int size);
   ~DiceGame();
   void roll();
   void roll(vector<int> pos);
   int getMultiples(int howmany, int excludevalue = 0);
};

DiceGame::DiceGame(int size){
   this->size = size;
   values = new int[size]; //oppretter en tabell av størrelse size
   roll();
   sort();
}

DiceGame::~DiceGame(){
   delete[] values; //sletter tabellen
}
```

b) Det finnes mange forskjellige slags terninger. I noen spill brukes terninger med bokstaver eller andre tegn, og noen terningtyper har også et annet antall sider.

- Hvordan vil du lage en generisk klasse som støtter spill hvor terningene har andre typer verdier? For eksempel kan ett spill ha terninger med verdier av typen char, mens et annet spill kan ha terninger med string som verdier. Forklar med hjelp av kort tekst og kode.
- Vis og forklar hvordan du med denne løsningen kan/må sette spesifisere hvilke verdier som er lovlige. For eksempel hvis datatypen til terningene er char så kan det være kun bokstavene 'A', 'B', 'C', 'D', 'E', 'F' som er gyldige verdier.

Template-klasse er stikkordet her, og gyldige verdier kan f.eks. oppgies vha. en vector<T> som parameter til konstruktøren.

```
template<class T>
class DiceGame{
private:
   int numberofdice;  //antallet terninger for spillet
   int numberofvalues; //antallet mulige verdier en terning kan ha
   T *valueset; //de lovlige verdiene for en terning
   T *dice;  //terningene
public:
   DiceGame(set<T> valueset, int numberofdice);
   ~DiceGame();
   void roll();
};
```

Oppgave 4: Static og unntakshåndtering (15%)

a) Implementer en klasse kalt OneAndOnly som har den egenskapen at du kun kan finnes en enkelt instans av denne klassen i programmet ditt. Dette kan løses ved å bruke en static medlemsvariabel kombinert med unntaksmekanismen m.m. Hvis du allerede finnes en instans av klassen i programmet ditt skal det kastes et unntak hvis du prøver å instansiere enda et objekt.

Her må vi bruke en static medlemsvariabel av typen int eller bool. Konstruktøren tester på om denne medlemsvariabelen. Hvis den er false (eller 0) kan vi instansiere et objekt og sette verdien til true (eller 1). Hvis verdien er true, har vi allerede en instans og da kaster vi et unntak som avbryter instansieringen. Hvis du i tillegg har med en destruktor slik at du i tillegg kan resette medlemsvariabelen til false ved sletting av objekter, har du en mekaniske som kun lar deg ha en instans aktiv, men det er lov å oprette og slette en instans flere ganger.

```
class OneAndOnly{
private:
  static bool instance;
public:
  OneAndOnly();
  ~OneAndOnly();
};

bool OneAndOnly::instance = false;

OneAndOnly::OneAndOnly(){
  if (instance){
    throw 1;
  }else{
    instance = true;
    cout << "instans opprettet" << endl;
  }
}

OneAndOnly::~OneAndOnly(){
  cout << "instans slettet fra minne" << endl;
  instance = false;
}
```

Oppgave 5: Old MacDonald had a farm (15%)

Klassen **MacDonald** som er deklarert og implementert under kan brukes for å skrive ut et av versene i sangen "Old MacDonald had a farm". Denne sangen består av et uendelig antall vers hvor det eneste som er forskjellig fra vers til vers er dyreart og lyden som dyret laget. I verset som er gjengitt er dyrearten "cows" og dyrelyden "moo".

```
class MacDonald{
public:
  string lines();
};

string MacDonald::lines(){
  stringstream text;
  text << "Old MacDonald had a farm," << endl << "Ee i ee i oh!" << endl;
  text << "And on that farm he had some cows," << endl;
  text << "Ee i ee i oh!" << endl;
  text << "With a moo-moo here," << endl;
  text << "And a moo-moo there" << endl;
  text << "Here a moo, there a moo," << endl;
  text << "Everywhere a moo" << endl;
  text << "Old MacDonald had a farm," << "Ee i ee i oh!" << endl;
  return text.str();
}

ostream& operator<<(ostream& o, MacDonald* m){
  o << m->lines();
  return o;
}
```

a) Lag en variant av klassen MacDonald hvor du implementerer virtuelle funksjoner for dyreart og dyrelyd. Vis hvordan du kan bruke disse funksjonene for å lage et generelt vers som har forskjellige dyr som subtyper. Lag minst to eksempler på dyreklasser f.eks. "cows" som sier "moo" og "pigs" som sier "oink".

```
class Verse{
public:
  string lines();
  virtual string name() = 0;
  virtual string sound() = 0;
};
string Verse::lines(){
  stringstream text;
  text << "Old MacDonald had a farm," << endl;
  text << "Ee i ee i oh!" << endl;
  text << "And on that farm he had some " << name() << "," << endl;
  text << "Ee i ee i oh!" << endl;
  text << "With a " << sound() << "-" << sound() << " here," << endl;
  text << "And a " << sound() << "-" << sound() << " there" << endl;
  text << "Here a " << sound() << ", there a " << sound() << "," << endl;
  text << "Everywhere a " << sound() << endl;
  return text.str();
}
```

```
ostream& operator<<(ostream& o, Verse* v){
  cout << v->lines();
  return cout;
}

class Cows: public Verse{
public:
  string name(){return string("Cows");};
  string sound(){return string("moo");};
};

class Pigs: public Verse{
public:
  string name(){return string("Pigs");};
  string sound(){return string("oink");};
};

class Chickens: public Verse{
public:
  string name(){return string("Chickens");};
  string sound(){return string("cluck");};
};
```

b) Lag en main-funksjon hvor du viser bruk av en vector som inneholder forskjellige "vers" av sangen (instanser av Cows, Pigs, Chicken etc) og hvor du kan iterere over innholdet for å få skrevet ut sangen.

```
int main(){
  vector<Verse*> verses;
  verses.push_back(new Cows());
  verses.push_back(new Pigs());
  verses.push_back(new Chickens());

  for (int i = 0; i < verses.size(); i ++){
    cout << verses[i] << endl;
  }
}
```

c) Hvorfor er ikke operatoren << implementert som medlem av klassen over? Har det i dette tilfellet noen hensikt å deklarere denne operatoren som friend av klassen MacDonald?

Den kan ikke være medlem fordi medlemstypen da må være venstre operand. Det har heller ikke noen hensikt å deklarere denne som friend siden den ikke trenger tilgang til medlemsvariabler (bruker bare en funksjon).