NTNU
Institutt for datateknikk
og informasjonsvitenskap

Eksamensoppgave i

TDT4102 - Prosedyre- og objektorientert programmering

TENTATIVT LØSNINGSFORSLAG

HUSK AT DET KAN VÆRE MANGE LØSNINGER PÅ EN OPPGAVE

Onsdag 1. juni 2011, 09:00

Kontaktperson under eksamen: Trond Aalberg (97631088)

Eksamensoppgaven er utarbeidet av Trond Aalberg
og kvalitetssikret av Hallvard Trætteberg

Språkform: Bokmål

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

Sensurfrist: Fredag 24 juni.

Generell introduksjon

Les gjennom oppgavetekstene og finn ut hva det spørres om. Noen av oppgavene har lengre forklarende tekst, men dette er for å gi mest mulig presis beskrivelse av hva du skal gjøre. Fokuser på det som er det sentrale spørsmålet i hver deloppgave. Alle oppgaver kan løses med et relativt lite antall kodelinjer.

All kode skal være C++.

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.

Noen av oppgavene er "oppskriftsbasert" og vi spør etter forskjellige deler av et litt større helhetlig 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.

Hele oppgavesettet er arbeidskrevende og det er ikke foreventet at alle skal klare alle oppgaver innen tidsfristen. Tenk strategisk i forhold til ditt nivå og dine ambisjoner. !

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.

Kommentarer fra rettingen:

Mange varianter av løsninger som ga lik uttelling. LF viser en A oppgave (som ville fått 100%).

Det er mulig å få en C selv om du bare har besvart 1-3 og har trekk i enkelte av deloppgavene.

Sideantallet i besvarelsene er vesentlig mindre enn ved tidligere eksamensoppgaver og det setter sensor pris på. Garantert ingen som har kommet dårligere ut ved ved at oppgavene la opp til kortere svar (og at jeg har oppfordret til å skrive mindre).

Oppgaver som utpreget seg med å være vanskelige: 1d, 2e, 3a, 3e, 4a, 4b, 4d. Det er likevel mange nok som har fått til disse på en utmerket måte og det er ofte disse som skiller.

Oppgave 1: Kodeforståelse og feilfinning (20%)

a) Hva skrives ut av følgende kode?

```
int a = 10;
int b = 20;
int c = 30;
int d = 40;
a += ++b;
d = (d % c) + (d / c);
cout << a << endl; //31
cout << d << endl; //11
```

b) Hva skrives ut av følgende kode?

```
int a[] = {1, 2, 3, 4, 5};
int *b = a;
int *c = &a[2];
a[0] = 0;
cout << a[0] << endl;     //0
cout << *b << endl;       //0
cout << *(++c) << endl;   //4
```

c) Funksjonen **`insert`** i følgende kode er ment å skulle sette inn noder i en lenket liste i sortert rekkefølge, men den inneholder feil som gjør at enkelte noder blir plassert feil. Identifiser og forklar kort hva som er feil, og vis hvilke(n) endring(er) som kan gjøres for at dette skal fungere korrekt.

Eksempel som illustrerer feil i insert-funksjonen:

Følgende sekvense av kall:
in->itemsert(head, "epler");
insert(head, "bananer");
insert(head, "appelsiner");
insert(head, "øl");
insert(head, "gulost");

Gir følgende feilsortert liste:
appelsiner, bananer, epler, øl, gulost

I en riktig implementasjon skulle gulost ha kommer før øl!

```
struct ListNode{
   string item;
   ListNode *next;
};
typedef ListNode* ListNodePtr;

void insert(ListNodePtr &head, string item){
   ListNodePtr newNode = new ListNode;
   newNode->item = item;
   newNode->next = NULL;
   if (head == NULL){
      head = newNode;
   }else{
      ListNodePtr current = head;
      ListNodePtr prev = NULL;
      while((current->next != NULL) && (item > current->item)){
         prev = current;
         current = current->next;
      }
      if (prev == NULL && item =< current->item){
         newNode->next = current;
         head = newNode;
      }else if (current->next == NULL && item > current->item){
         current->next = newNode;
      }else{
         prev->next = newNode;
         newNode->next = current;
      }
   }
}
```

I praksis er det en mangelfull håndtering av innsetting på start og slutt. Selv om prev == NULL vet vi ikke om ny node skal inn før eller etter current og må utvide testen. Selv om current->next == NULL vet vi ikke om ny node

skal inn før eller etter siste node. I alle andre tilfeller vet vi at ny node skal inn mellom prev og current.
PS! Koden som ble brukt i forelesingen tester på en litt annen måte som unngår hele problemet.

Kommentar fra rettingen: For å få fullt score må du påpeke feil både for innsetting i starten og slutten, samt ha med forslag til løsning som viser at det er sammenligningen med item som mangler.

Noen mente det var bruk av norske bokstaver som feilet, men ææå vil bestandig ha en verdi som er større en z og ÆØÅ vil bestandig ha større verdi en Z. Norske tegn vil også bestandig ha en verdi som er i stigende rekkefølge for ææå.Tegnsett kan være problematisk ved lesing, skriving, samt konvertering fra små til store bokstaver etc., men her er det ikke noe som indikerer at dette er problemet. Vi har heller ikke fokusert på slike problemer og lærdommen er at dere ikke trenger å bekymre dere om "sære" problemer som vi ikke har lagt vekt på i forelesingen eller øvingen

Oppgave 2: Funksjoner (25%)

Deloppgavene henger sammen og det er meningen at en funksjon i en oppgave skal kunne benyttes i en annen oppgave. Merk at det likevel IKKE er nødvendig å ha løst en oppgave for å kunne bruke funksjonen denne beskriver i en annen oppgave - så lenge du skjønner hva funksjonen gjør.

I evalueringen av deloppgavene ser vi etter riktig logikk, at koden er lettlest og enklest mulig, at du bruker en hensiktsmessig kontrollstruktur m.m.

a) Lag en funksjon **`bool isLeapYear(int year)`** som returnerer true hvis **`year`** er et skuddår og false hvis ikke. Funksjonen skal implementere reglene for skuddår.
Fritt etter Wikipedia: "Et skuddår er et år som har en dag mer enn et normalt år. Jorden bruker ca. 365,2422 dager på en runde rundt solen. For at dette skal passe inn med vår kalender, har vi skuddår. Et skuddår er normalt hvert fjerde år – i alle årstall som er delelige med 4. Unntaket er hundreårene (1700, 1800, 1900 etc.) som ikke er skuddår med mindre de er delelige med 400 (1200, 1600, 2000, 2400 etc.). Det ble derfor skuddår i 2000, men skuddåret faller bort i 2100." Et skuddår har 366 dager mens et vanlig år har 365.

```
//Her er det en grei løsning å bruke if og else if kombinert
//med % i selve testen
//Merk bruken av return - da slipper du å tenke på temp variabler
//Starter med den mest spesifikke betingelsen (year % 400) og
//da kan du i neste ledd teste på % 100
bool isLeapYear(int year){
  if ((year % 400) == 0){
    return true;
  }else if((year % 100) == 0){
    return false;
  }else if((year % 4) == 0){
    return true;
  }else{
    return false;
  }
}

//Her er en enda mer kompakt løsning som bruker % kombinert med boolske
//operatorer. Den er litt mer krevende å forstå/lese logikken i (og lage),
//men viser hvor kompakt og enkelt det kan gjøres hvis man har full
//kontroll på boolske operatorer.
bool isLeapYear(int year){
  return ( (year%4 == 0) && (!(year%100 == 0)) || (year%400==0));
}
```

Ryddige forslag til logikk som viser at du tenker riktig fikk bra score her. Mulige feil i koden din som det er naturlig at du først oppdager ved kompilering/uttesting, gir bare mindre trekk.

b) Lag en funksjon **`int daysOfMonth(int month, int year)`** som returnerer antallet dager det er i en måned i et gitt år. For februar skal funksjonen returnere riktig tall basert på om det er skuddår eller ikke. Hvis month er en ugyldig verdi (dvs. verdi som ikke representerer en måned) skal funksjonen returnere 0.
I parameteren for måned brukes månedsnummer hvor januar = 1, februar = 2, mars = 3 osv. Hvis det er skuddår er det 29 dager i februar, hvis ikke er det 28 dager. Måneder med 31 dager er januar, mars, mai, juli, august, oktober, desember. Alle andre måneder har 30 dager.

```
int daysInMonth(int month, int year){
  //test på om månedsnummer er riktig
  //bruker en separat if-else til dette
  //siden vi skal teste på < og >
  if(month < 1 || month > 12){
    return 0;
  }
  //her oppnår du den mest ryddige løsningen ved å bruke switch
  //kombinert med fall-through. Vi trenger ikke break siden vi avslutter
  //funksjonen med return. Oppgaven kan også løses med if-else, men det
  //blir litt mer uryddig siden du må teste mot de spesifikke verdiene
  //Merk at det ikke er mulig å bruke % siden måneder med 31 dager er
  //både partall og oddetall og det samme gjelder for måneder med 30 dager
  switch (month) {
    case 2: //februar
      return (isLeapYear(year) == true ? 29 : 28);
      //if-else operatoren kommer til sin rett her
    case 1: //måneder med 31 dager, merk at vi bruker fall-trough
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
      return 31;
    default: //alle andre måneder
      return 30;
  }
};
```

```
NB! Det er også mulig å ha alle andre måneder som case og heller la default
håndtere alle andre tilfeller (de som erugyldige måneder).

Fra rettingen:
også mange gode og ryddige løsninger med bruk av if. Mange glemte å sjekke
for ugyldige verdier og fikk trekk for dette.
```

c) Lag en funksjon **`int secondsInMonth(int month, int year)`** som returnerer antallet sekunder det er i en bestemt måned i et år.

```
//bruk konstant for antall sekunder i en dag - dette er ikke et krav,
//men gjør koden litt ryddigere siden det er et tall som skal brukes
//også i andre oppgaver
```

```
const int DAYSECONDS = 24 * 60 * 60;

//enkel utregning - husk å bruke daysInMonth
int secsInMonth(int month, int year){
  return daysInMonth(month, year) * DAYSECONDS;
};
//Her var det viktig å bruke funksjonen fra c). Duplisering av kode er en
//uting - og hva er vitsen med å lage en funksjon som gir deg antall dager
//i en gitt måned, hvis du ikke bruker den etterpå?
```

d) Lag en funksjon **`int secondsInYear(int year)`** som regner ut og returnerer antallet sekunder det er i **`year`**. Eksempel vil **`secondsInYear(`***2000***`)`** returnerer *31622400*, mens **`secondsInYear(`***1999***`)`** vil returnere *31536000*.

```
//En enkel å grei funksjon;-)
//Plasseringen av denne som oppgave 3d) var en test på
//om dere klarte å se at denne ikke er avhengig av spesifikk utregning
//for hver måned. Det er bare en dag forskjell mellom skuddår eller ikke
//og da er det mer effektivt å hardkode sekundutregningen.
int secsInYear(int year){
  //hjelpefunksjon for å finne antall sekunder i et angitt år
  if (isLeapYear(year)) {
    return 366 * DAYSECONDS;
  }else {
    return 365 * DAYSECONDS;
  }
};
```

e) Lag en funksjon **`void printCurrentDate()`** som bruker funksjonen **`time()`** i **`<ctime>`** biblioteket, regner ut gjeldene år, måned og dag og skriver ut resultatet til skjerm. Utskriften skal inneholde år, måned og dag, men du bruker bare tall i utskriften og formatet velger du selv. Eksempel på grei utskrift er: "*2011 1/6*".
I C++ biblioteket ***`<ctime>`*** *finnes det en funksjon* ***`time()`*** *som vil gi deg antallet sekunder som er gått siden klokken 00:00, 1 januar, 1970. Denne informasjon henter funksjonen fra klokka i datamaskinen og verdien som returneres kan omregnes til gjeldende dato og tidspunkt. For enkelthets skyld kan du anta at funksjoner er deklarert som* ***`int time()`***.

```
//En litt mer utfordrende funksjon.
//Her er vi interessert i hvilken logikk du velger for å beregne.
//Løkke som trekker fra sekunder år for år til vi ikke kan trekke fra
//flere år, deretter måned for måned og dag for dag, er en ryddig og enkel
//løsning.
void printDate(){
  int secs = time(NULL); //bruker <ctime> funksjonen
  //vi regner ut året først, trekker fra sekundene for hvert år
  //mens vi inkrementerer år
  int year = 1970;
  while (secs >= secsInYear(year)) {
    secs -= secsInYear(year);
    year++;
  }
```

```
//deretter regner vi ut måned basert på rest av sekundene
  int month = 1;
  while (secs >= secsInMonth(month, year)) {
     secs -= secsInMonth(month, year);
     month++;
  }
  //til slutt regner vi ut dagen med heltallsdivisjon (resttallet
  //kan vi ignorere)
  int day = (secs / DAYSECONDS) + 1 ;

  cout << day << "/" << month << " " << year <<  endl;
};
```

Oppgave 3: Klasser (25%)

I denne oppgaven skal du jobbe med en klasse **`Time`** som representerer en dato og et tidspunkt (dvs. har variabler for år, måned, dag, time, minutt, sekund). **`Time`**-klassen har en konstruktør som benytter seg av funksjonen **`time()`** i **`<ctime>`** slik at objektene blir instansiert med gjeldende verdier for det øyeblikket de blir opprettet (se oppgave 2 e). I siste deloppgaven skal du også lage subtypen **`LocalTime`** hvor dato/tid er justert med tidssone.
NB! Merk at i praksis er det fullt mulig å løse alle deler i oppgave 3 uten at du har besvart oppgave 2 - så lenge du har lest og forstått funksjonene hva funksjonene i oppgave 2 gjør.

a) **Time** sin konstruktør skal beregne riktig år, måned, dag, time, minutt og sekund for objektene som klassen instansierer. Da er det greit å ha funksjonene som er beskrevet i 2a- 2d som medlemsfunksjoner. Hvordan bør parameterlista til disse være når de er medlemsfunksjoner? Beskriv kort og/eller gi et eksempel. *NB! Logikken for konstruktøren var tema i oppgave 2e og det er kun svar på spørsmålet over vi ser etter i denne deloppgaven.*

Beregningsfunksjonene brukes i utregningen av hvilket år, måned, dag det er og argumentene er ikke det samme som medlemsvariablene. Dermed blir argumentlista akkurat de samme.
I praksis er det også mulig å se for seg at disse gjøres om til medlemsfunksjoner slik at f.eks. isLeapYear() gir deg true hvis året som objektet representerer er et skuddår. Dette vurderes som en dårlig løsing. Selv om det likevel vil være mulig å bruke funksjonene i konstruktøren ved at medlemsvariabelene inkrementeres under setting av riktig dato/tid, endrer du funksjonene fra å være reine beregningsfunkjsoner til å være get-funksjoner.

b) Hva betyr det at en medlemsfunksjon er **`static`** og er det noen av funksjonene fra 2a - 2d som kan deklareres som **`static`** hvis de gjøres om til medlemmer av **`Time`**?

static medlemsfunksjoner for en klasse A er er funksjoner som kan kalles uten at du har et objekt av klasse A. Dvs. de har klassen som kontekst og ikke et spesifikt objekt. Bruker ikke medlemsvariabler. Alle beregningsfunksjonene kan være static medlemsfunksjoner. Hvis du tolker printCurrentDate som en funksjon som skal skrive ut "tida akkurat nå" uavhengig av når objektet ble instansiert (dvs gjør et kall med time og regner ut tia på nytt), så kan også denne være static.

Kommentar etter retting: Alle som hadde svar som var riktig i forhold til a) fikk lik uttelling. Her var det også mange gode forklaringer på hvorfor de skulle være private. Spesielt gode var de som påpekte at funksjonene ville fungere like godt om de ble brukt private variabler.

c) For **`Time`** har vi ikke noen get-funksjoner og objektene som instansieres blir såkalte uforanderlige/immutable objekter. Overlagre insertion-operatoren (**`<<`**) slik at du for objektet **`Time t`** kan skrive **`cout << t << endl;`** Vis hvilke andre endringer som må gjøres i klassedeklarasjonen for at en slik overlagring skal være mulig (men du får ikke legge til get-funksjoner siden objektene fortsatt skal være uforanderlige).

Vi må legge til denne operatoren som friend av klassen så den kan lese variablene (siden vi ikke vil ha public get-funksjoner). Merk at denne ikke kan implementeres som medles siden ostream er venstre operand.

```
ostream& operator<<(ostream &o, Time &t){
  //selve formatteringen og hvilke av medlemmene du skriver ut er
  //ikke viktig, men du må bruke ostream variabelen og ikke cout inne i
  //implementasjonen
  o << t.year << "/" << t.month << "/"
  << t.day << " " << t.hour << ":"
  << t.minute << ":" << t.second;
  return o;
}
```

Kommentar fra rettingen: Her så vi etter at du forstått at den måtte være friend, at du faktisk brukte ostream argumentet til utskrift og at du returnerte ostream argumentet.

d) Gi et eksempel på en annen operator som generelt kan være nyttig å overlagre for klassen **`Time`** og vis implementasjonen av denne som medlem av klassen.

```
//her kan du f.eks. implementere ==, < eller >
//de boolske operatorene er gode eksempler, aritmetiske er dårlige
//eksempler siden det er vanskelig å si hva som skal gjøres ved +,-,* etc.
//Merk at det selvsagt kan være litt foruft i å implementere f.eks. -
//siden den kan returnere antallet sekunder i differanse, men da bør
//du ha forklart godt hva du mener med operatoren
bool Time::operator==(const Time &t) const{
  return year == t.year && month == t.month && day == t.day
       && minute == t.minute && second == t.second;
}
```

Kommentarer etter å ha rettet:
Alle som foreslo en av sammenligningsoperatorene OG hadde bare en parameter fikk full score av meg. Selve implementasjonen er irrelevant så lenge du viser et eller annet som returnerer en bool - selvsagt trekk hvis du demonstrerer at du overhodet ikke kan å programmere ;-)
Skrekkeksempler: + og % er rett og slett elendige og HELT FEIL eksempler. Det aller verste eksempelet er %. Stort verre operator kan du ikke finne på å foreslå for en slik klasse (joda det var en del som brukte denne).

I **`Time`**-klassen brukes **`time()`** for å beregne dato og tid, men dette gir en verdi som er "universell tid UTC" - tid som ikke er justert for tidssone. For å få riktig tid, f.eks. for Norge, må vi justere for lokal tidssone (norsk tid er f.eks. 1 time før UTC) og vi lager subtypen **`LocalTime`** for dette. Vi antar at sone er heltall (og utelater alt som har med sommertid å gjøre).

e) Vis hvordan du vil implementere konstruktøren **`LocalTime::LocalTime(int timezone)`** og funksjonen **`LocalTime::setTimezone(int timezone)`** slik at disse funksjonene setter riktig verdi for de arvede variabler år, måned, dag, etc. (verdier basert på tidssone). Merk at funksjonen **`setTimezone`** brukes for å endre tidssone for et **`LocalTime`**-objekt. Her må du også vurdere om det er behov for andre endringer (inklusive endringer i **`Time`**) for å få til en hensiktsmessig oppgavefordeling og arv mellom supertype og subtype.
Du trenger ikke å lage fullstendige funksjonsimplementasjoner i denne oppgaven, men skal vise/forklarer det som er essensielt og f.eks. påpeker eventuelle problemer som dette innebærer. NB! Det er selvsagt viktig å demonstrere god innsikt i objektorientering.

Konstruktøroppgave var en litt åpen oppgave med noen utfordringer. Denne kan løses på flere måter, avhengig av hvordan du du vil ha samspillet og oppgavefordelingen mellom sub og supertypen.

NB! Merk at det spørres etter subtype og da er det arv som er tema her. Selvsagt like bra hvis man med gode argumenter og innsikt i objektorientering mener at dette bedre kan (eller bør) løses på andre måter, men her var det å vise impl. ved arv som var det egentlige spørsmålet.

Viktig å unngå at time() kalles flere ganger når et objekt instansieres, samt at dato/tid ikke beregnes unødvendig (flere ganger) eller at logikken for beregning er duplisert i flere funksjoner.
Siden setTimeZone skal endre tidssone må du kunne beregne riktig dato på nytt hver gang denne kalles. Det er også mulig å ha en løsning som er basert på at LocalTime gir deg manipulert tid i forhold til Time sine verdier. Da må du ha funksjonalitet som håndterer at tidssonen også kan påvirke år, måned og dag (og ikke bare time).

```
class Time {
private:
   int year, month, day, hour, minute, second;
   static bool isLeapYear(int year);
   static int daysInMonth(int month, bool leapyear);
   static int secsInYear(int year);
   static int secsInMonth(int month, int year);
protected:
   void setTime(int t);
   int rawtime; //Nå er rawtime lesbar/skrivbar for subtyper
public:
   Time();
   Time(int t);
   friend ostream& operator<<(ostream &o, Time &t);
};

class LocalTime : public Time {
private:
   int zone;
public:
   LocalTime(int z);
   int getTimeZone();
   void setTimeZone(int z);
};

LocalTime::LocalTime(int z): Time(time(NULL) + (z * 3600)){
   zone = z;
};

void LocalTime::setTimeZone(int z){
   rawtime = rawtime + ((z - zone) * 3600);
   setTime(rawtime);
   zone = z;
};

Time::Time(){
   rawtime = time(NULL);
   setTime(rawtime);
}
```

```
Time::Time(int t){
   rawtime = t,
   setTime(rawtime);
}

//Alle beregningsfunksjonene samt denne trenger du ikke ha med, men det
//er tatt med kun for de som i ettertid vil lage seg en fullstendig
//implementasjon
void Time::setTime(int secs){
   year = 1970;
   while (secs >= secsInYear(year)) {
      secs -= secsInYear(year);
      year++;
   }
   month = 1;
   while (secs >= secsInMonth(month, year)) {
      secs -= secsInMonth(month, year);
      month++;
   }
   //så regner vi ut dagen
   day = (secs / DAYSECONDS) + 1; //første dag er 1
   secs = secs % DAYSECONDS;

   //så timen
   hour = secs / (60 * 60);
   secs = secs % (60 * 60);

   //så minutt
   minute = secs / 60;
   secs = secs % 60;

   //så sekund
   second = secs;
}
```

Oppgave 4: Bruk av Standard Template Library - STL (30%)

I denne oppgaven skal du lage deler av et program som leser ei tekstfil, lager en indeks over ordene som finnes i fila, og til slutt skriver ut innholdet i indeksen.

Indeksen skal være organisert på bokstavene A-Å (store bokstaver i alfabetisk rekkefølge), under hver bokstav skal du finne alle ordene i fila som begynner på denne bokstaven (organisert i alfabetisk rekkefølge), og for hvert ord skal du lagre alle linjenumrene som inneholder dette ordet (organisert i stigende rekkefølge). Det skal være enkelt å slå opp i indeksen på bokstav og du skal ikke ha med bokstaver som det ikke finnes ord for i indeksen. Det skal ikke være duplikate ord i indeksen eller duplikate linjenummer for et ord.

I denne oppgaven skal du vise din kjennskap til STL og lage kode som i størst mulig grad benytter seg av STL (f.eks. er det en dårlig ide å lage funksjonalitet som allerede finnes i STL). I appendiks finner du en oversikt over relevante klasser og funksjoner som du kan bruke (men du kan også bruke andre biblioteks-funksjoner som du kjenner eller finner f.eks. i læreboka).

a) Hvilke(n) datatype(r)/datastruktur(er) vil du velge for selve indeksen? Denne skal være i logisk overenstemmelse med beskrivelsen over. Tips: det er mulig å lage seg en enkelt variabel for hele indeksen hvis du bruker STL-klassene litt kreativt.

```
//En typedef (alias navn for en type) gjør det litt lettere å skrive typen
//I seinere oppgaver skal du bla. bruke iterarorer og da er det greit
//med slike typedef's
typedef map<char, map<string, set <int> > > indextype;

//Bruker du en slik typedef kan du skrive:
indextype index;

//eller du kan bruke typen direkte uten typedef
map<char, map<string, set <int> > >  index;
```

Oppgaven kan også løses på andre måter, men da må du kode en MYE mer selv. Eksempelvis er det mulig å lage seg en klasse for CharEntry som inneholder en collectionvariabel for alle ordinnførslene. Ordinnførsel kan du igjen ha en egen type WordEntry for. Ulempen med en slik løsning er at du lager noe du like gjerne kunne løst med map, samt at du selv må kode unikhetsmekanismen og støtte sorteringen hvis du baserer deg på vector. Bruker du set vil du få problemer med at objektene som lagres ikke kan oppdateres (du må hente ut kopi, slette fra settet, endre og legge til på nytt).

b) Lag en funksjon **`void makeIndex(string filename, .....)`** som leser fra fila identifisert med **`filename`** og fyller indekser med innhold. Hva som kommer etter **`filename`** i parameterlista avhenger selvsagt av hva slags datastruktur(er)/datatype(r) du har valgt. NB! Du kan basere deg på en fiktiv funksjon **`vector<string> words(string)`** for å hente ut alle enkeltordene i en streng.

```
//vi antar at words returnerer ordene i små bokstaver
//det er positivt hvis du har merket at utskriften av ordene var
//i små bokstaver, men det står ingenting om det i oppgaven
//Spesifiseringe av A-Å i norsk og nynorsk utgaven var der fordi det er
//et realistisk eksempel (men det står ingenting i LF om at toupper er
//begrenset til ASCII (i praksis må du bruke en annen variant av touppper
//som støtter lokale tegnsett for å få det til å virke


void makeIndex(string filename, indextype &index){
  //åpning av fila og standar sjekk, kaster unntak ved feil (2c)
  //her kaster vi et unntak av typen string, men det beste er å lage
  //seg en egen klasse for en slik unntakstype (med en string variabel)
  ifstream inputfile;
  inputfile.open(filename.c_str());
  if (inputfile.fail()){
    //Dette er svar på oppgave 2c
    //I 2b holder det å f.eks. avslutte programmet eller
    //skrive ut feilmelding
    throw FileException(filename);
  }
  //koden for å opprette indeksen
  string s;
  int lnr = 0;
  while (getline(inputfile, s)){
    lnr++;
    vector<string> lwords = words(s);
    for (int i = 0; i < lwords.size(); i++){
      //i prinsippet: index['A']["arrows"].insert(22);
      index[touupper((lwords[i])[0], loc)][lwords[i]].insert(lnr);
    }
  }
  //Husk å stenge fila
  inputfile.close();
}

//Her brukes en variant av toupper som finnes i <locale>, men denne er
//kun brukt for å vise kode som kan håndtere norske tegn (men merk at du
//da må ha litt kontroll over tegnsettet som er brukt i fila og tegnsettet
//som kommandovinduet bruker for utskrift osv.
//Siden oppgaven handlet om STL er tegnsettproblemer noe dere glatt
//kan ignorere: toupper(char) fra <cctype> som er listet i appendix
//er grei selv om denne i realiteten bare håndterer ASCII
```

c) Funksjonen **`makeIndex`** tar inn et filnavn som det skal leses fra, men det er mange feil som kan oppstå ved filhåndtering. Funksjonen **`makeIndex`** skal kaste et unntak hvis funksjonen prøver å lese fra ei fil som ikke finnes og du skal vise hvordan du implementere dette samt hvordan du vil fange opp unntak som er kastet, på et fornufig sted i koden din. Når unntak fanges opp skal du skrive ut filnavnet som forårsaket unntaket.

```
//Se koden over for hvor du skal kaste unntaket
//Her bruker vi en egen unntakstype som arver fra std::runtime_error.
//Dette er den ideelle lønsingen, men siden det ikke står noe om
//unntakstypen er det greit om du har valgt en mer pragmatisk løsning og
//f.eks. kaster filname som unntak (mao en string)
//Det vi primært ser etter her er at du kaster på riktig sted (etter
// riktig test)og vet hvordan du skal fange opp unntak.

class FileException : public std::runtime_error {
public:
  FileException(const string &what) : std::runtime_error(what) { }
};

int main(){
  indextype index;
  string filename("test.txt");
  try{
    makeIndex(filename, index);
  }catch(FileException &exc){
    cout << exc.what() << endl;
    exit(1);
  }
  cout << index << endl;;
}
```

d) Lag en utskriftsfunksjon med fritt valgt navn (eller overlagre **<<**) og vis hvordan du kan skrive ut indeksen på en ryddig/pen måte (i overenstemmelse med eksempelet under).

```
//her er det riktig å bruke iteratorer for siden STL er hovedtema
//hvis du løste oppgaven uten STL vil vi evaluere den i forhold til
//datatypen(e) du brukte

ostream& operator<<(ostream &o, indextype &idx){
  indextype::iterator cit;
  for (cit = idx.begin(); cit != idx.end(); ++cit){
    o  << cit->first << endl; //skriver bokstaven
    map<string, set <int> >::iterator wit;
    for (wit = cit->second.begin(); wit != cit->second.end(); ++wit){
      o  << '\t' << wit->first << ": "; //skriver ut "ord: "
      print_lines(wit->second); //skriver ut linjenr med komma
      cout << endl; //linjeskift etter lista med linjenr.
    }
  }
  return o;
}
```

e) Merk bruken av komma mellom linjenumrene i utskriften under. Riktig håndtering av komma i utskriften teller som en egen deloppgave.

```
//Utfordringen er at vi må vite når vi skriver ut siste element
//siden det ikke skal være komma etter dette

//Her er en enkel løsning for problemet siden den
//forutsetter at et set av linjer aldri er tomt.
//Vi skriver ut første verdi før løkka og kan deretter skrive
//ut ", verdi" i hver iterasjon. Da slipper du å spesialhåndtere
//siste element og får en enkel og grei løsning.
void print_lines(set<int>& lnrs){
   set<int>::const_iterator i=lnrs.begin();
   cout << *i;
   i++;
   while ( i != lnrs.end() ){
      cout << ", " << *i;
      i++;
   }
}

//Hvis du vil skrive ut verdi og komma etter verdien kan du også bruke
//size() for å finne antall verdier i set<> og ha en teller for hvor mange
//du har skrevet ut. Da kan du teste og finne ut om du skriver ut siste
//verdi og droppe komma etter denne. Da blir det noe sånt som dette:
//for (int i = 0; wordmapit->second->size(); i++, lineit++){}

//En variant med en annen logikk for utskrift av komma kan baseres på å
//bruke iteratorer for å teste, men da må du teste mot rbegin() siden
//end() peker til etter siste verdi basert på at verdi skrives før komma,
//men en sjekk på om vi er kommet til siste element
//merk at iteratoren rbegin returnerer er en annen type enn lit
//og derfor må derefere og teste på verdiene - noe som er OK for set,
//ellers ikke. Her vist som del av operatorimlementasjonen
ostream& operator<<(ostream &o, indextype &idx){
   indextype::iterator cit;
   for (cit = idx.begin(); cit != idx.end(); ++cit){
      o  << cit->first << endl;
      map<string, set <int> >::iterator wit;
      for (wit = cit->second.begin(); wit != cit->second.end(); ++wit){
         o  << '\t' << wit->first << ": ";
         set <int>::iterator lit;
         for (lit = wit->second.begin(); lit != wit->second.end(); ++lit){
            o  << *lit;
            if (*lit != *wit->second.rbegin()){
               o  << ", ";
            }
         }
         cout << endl;
      }
   }
   return o;
}
```

Innhold i en eksempelfil:

To be, or not to be, that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them? To die, to sleep,
No more; and by a sleep to say we end

Eksempel på utskrift av indeksen

```
A
  a: 4, 6
  against: 4
  and: 3, 5, 6
  arms: 4
  arrows: 3
B
  be: 1
  by: 5, 6
D
  die: 5
E
  end: 5, 6
```

```
vector<string> words(string& line){
//Denne skal du IKKE implentere, men du kan bruke den
  vector<string> wordlist;
  string word = "";
  for (int i = 0; i < line.size(); i++){
    if (isalpha(line[i], loc)){ //hvis tegnet er en bokstav
      word += tolower(line[i], loc); //legger til enkeltbokstaver
    }else if (word != ""){ //hvis det ikke er bokstav og word ikke er tom
      wordlist.push_back(word); //lagrer ordet i vectoren
      word = ""; //setter word-variabelen til tom streng
    }
  }
  if (word != ""){ //spesialhåndtering for siste ord i linja
    wordlist.push_back(word); //storing last word
  }
  return wordlist;
}
```

Appendix 1: Classes and functions that may be of interest[a]

Functions common for all container classes	
begin	Return iterator to beginning (public member type)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
size	Return size (public member function)
empty	Test whether e.g. vector is empty (public member function)
vector	
operator[]	Access element (public member function)
at	Access element (public member function)
front	Access first element (public member function)
back	Access last element (public member function)
push_back	Add element at the end (public member function)
set	
insert	Insert element (public member function)
find	Get iterator to element (public member function)
list	
front	Access first element (public member function)
back	Access last element (public member function)
push_front	Insert element at beginning (public member function)
pop_front	Delete first element (public member function)
push_back	Add element at the end (public member function)
insert	Insert elements (public member function)
map	
operator[]	Access element (public member function) T& operator[] (const key_type& x); Creates a new key-value pair if key does not exist. Example: map<char,string> mymap; mymap['a']="an element"; Alternative access based on map iterator: (*it).first; (*it).second;
insert	Insert element (public member function) Example: mymap.insert (pair<char,int>('z',500));

Appendix 1: Classes and functions that may be of interest[a]

string	
operator[]	Get character in string (public member function)
at	Get character in string (public member function)
operator+=	Append to string (public member function)
append	Append to string (public member function)
c_str	Get C string equivalent (public member function)
size	Return length of string (public member function)
find	Find content in string (public member function)
rfind	Find last occurrence of content in string (public member function)
find_first_of	Find character in string (public member function)
find_last_of	Find character in string from the end (public member function)
substr	Generate substring (public member function)

Various functions
istream& getline (istream& is, string& str, char delim); Get line from stream (function)
istream& getline (istream& is, string& str); Get line from stream (function)
void sort (RandomAccessIterator first, RandomAccessIterator last); Sort elements in range
bool binary_search (ForwardIterator first, ForwardIterator last, const T& value);
int tolower (int c); Convert uppercase letter to lowercase
int toupper (int c); Convert lowercase letter to uppercase
int isalpha (int c); Check if character is alphabetic

time_t time (time_t * timer); Example:
time_t seconds;
seconds = time (NULL);
printf ("%ld hours since January 1, 1970", seconds/3600);
// Possible output: “266344 hours since January 1, 1970”
This is the actual time-function in <ctime>, but you can use the simplified version that is described under task 2 and 3 in the exam.

a. Note that we only have included the names and desciptions for most functions.