NTNU
Institutt for datateknikk
og informasjonsvitenskap

Eksamensoppgave i

TDT4102 - Prosedyre- og objektorientert programmering

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.

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;
cout << d << endl;
```

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;
cout << *b << endl;
cout << *(++c) << endl;
```

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.

```
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){
      newNode->next = current;
      head = newNode;
    }else if (current->next == NULL){
      current->next = newNode;
    }else{
      prev->next = newNode;
      newNode->next = current;
    }
  }
}
```

Eksempel som illustrerer feil i insert-funksjonen:

Følgende sekvense av kall:
insert(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!

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.

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.

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

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*.

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()`**.

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.

```
class Time{

private:
   int year, month, day;
   int hour, minute, second;
   ......
public:
   Time();
   ....
};
```

```
class LocalTime : public Time{

private:
   ......
public:
   LocalTime(int timezone);
   void setTimezone(int timezone);
   int getTimezone();
   ......
};
```

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.*

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`**?

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).

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.

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.

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.

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.

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.

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).

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

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
```

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.