NTNU
Institutt for datateknikk
og informasjonsvitenskap

Kontinuasjonseksamen i

TDT4102 - Prosedyre- og objektorientert programmering

Torsdag 12. august 2010, 09:00 - 13:00

Kontaktperson under eksamen: Trond Aalberg (97631088)

Eksamensoppgaven er utarbeidet av Trond Aalberg
og kvalitetssikret av Svein Erik Bratsberg

Språkform: Bokmål

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

Sensurfrist: Fredag 3 september.

Generell introduksjon

Les gjennom oppgavetekstene nøye 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.

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.

Oppgavesettet er arbeidskrevende og det er ikke foreventet at alle skal klare alle oppgaver innen tidsfristen. Disponer tiden fornuftig!

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: Grunnleggende programmering (30%)

Se nederst på siden for nyttige tips til noen av deloppgavene.

a) Hva skrives ut av følgende kode:

```
int a = 4;
int b = 5;
int c = (a++) + (--b);
cout << c << endl;

double d = 2;
int e = 4;
double f = d / e;
cout << f << endl;

int g = 2002;
int h = 10;
int i = g % h;
cout << i << endl;
```

b) Implementer en funksjon **`int countchar(char str[], char c)`** som kan brukes for å finne ut hvor mange ganger et spesifikt tegn er brukt i en tekst-streng (C-streng).

c) Implementer en funksjon **`string trim(string s)`** som fjerner blanke tegn i starten og slutten av en tekststreng. Hvis du kaller funksjonen med strengen *" Dette er en test "* som argument skal funksjonen returnere strengen *"Dette er en test"*.

d) Implementer en funksjon **`vector<string> split(string s, char c)`** som deler opp tekststrengen **`s`** i flere delstrenger med tegnet **`c`** som skilletegn. Hvis du gjør kallet **`split(`***"en;to;tre"*`, ';')`** skal du få returnert en vector som inneholder strengene "*en*", "*to*", "*tre*".

e) Implementer en rekursiv funksjon **`string int2string(int i)`** som gjør om et heltall av datatypen int til en strengrepresentasjon av tallet. Kalles funksjonen med heltallet *124* som argument skal den returnere strengen *"124"*. NB! oppgaven skal løses med bruk av rekursjon og du kan IKKE benytte deg av tilsvarende funksjoner som allerede finnes fra før i C/C++ biblioteket.

f) Implementer en funksjon **`int string2int(string s)`** som gjør om en strengrepresentasjon av et tall til en int-verdi. Hvis tekststrengen inneholder tegn som ikke er tall skal den kaste et unntak. Du bestemmer selv hva slags unnntakstype som skal brukes. Kalles funksjonen med strengen *"124"* skal den returnere int-verdien 124. Kalles funksjon med strengen *"12K3F"* skal den kaste et unntak.

Lengden til en string-variabel (antallet tegn i strengen) kan du finne med medlemsfunksjonen **`size()`**.
Enkelttegn i en string-variabel kan leses med medlemsfunksjonen **`at(int)`** eller ved å bruke indeksoperatoren **`[]`**.
Tegnene '0' - '9' finner du på plassene 48 - 57 i ascii-tabellen og for å konvertere fra int-verdiene 0 - 9 til tilsvarende chartegn kan du bruke **`x + 48`**.
Operatorene **`+`** og **`+=`** kan brukes for å legge et tegn til en string-verdi (konkatenere).

Oppgave 2: Klasser og lenkede lister (40%)

I denne oppgaven skal du implementere klasser for å organisere en liste med musikkfiler (tracks). Disse klassene skal for eksempel kunne brukes i et mediaavspillingsprogram, men i denne oppgaven skal du kun ta for deg funksjonalitet som håndterer innlesing fra ei fil, oppretting av en lenket liste hvor hver node inneholder informasjon om de enkelte tracks, og en funksjon for å bla igjennom lista.

Klassen **`Playlist`** representerer en lenket liste med informasjon om musikkfiler (tracks) hvor nodene er instanser av klassen **`Track`**. Innholdet i lista skal leses fra en tekstfil i konstruktøren til **`Playlist`**. I fila er informasjon om hvert track er lagret som en separat linje på formen "*Artist; Tracktitle; Rating; Trackfilename*". Eksempel på innholdet i en slik fil kan være:

Bob Dylan; Spirit on the water; 4; C:\MyMusic\spiritonthewater.mp3
Coldplay; Vival La Vida; 4; C:\MyMusic\vivalavida.mp3
The The; This Is The Day; 6; C:\MyMusic\thisistheday.mp3
No Doubt; Don't Speak; 5; C:\MyMusic\dontspeak.mp3
Annie Lennox; I Saved The World Today; 4; C:\MyMusic\isavedtheworldtoday.mp3

Oppgaven kan besvares del for del, eller du kan velge å implementere klassene samlet. Les igjennom alle deloppgaver slik at du får en oversikt over hva som skal gjøres og sørg for at det går tydelig frem hvor du svarer på hvilke deloppgaver.

a) Klassen **`Track`** skal brukes for å lagre informasjon om de enkelte musikk-filer og instansene skal være noder i en lenket liste. Hvilke medlemsvariabler vil du definere for klassen og hvilke datatyper velger du? Det skal være mulig å lese de enkelte informasjonselementene Artist, Tracktitle, Rating og Trackfilename med get-funksjoner. Merk at Trackfilename bare er et filnavn og at vi ikke skal gjøre noe som helst med selve innholdet i de enkelte musikkfiler.

b) Forklar hva som er formålet med en klasses konstruktør og implementer en egnet konstruktør for **`Track`**-klassen.

c) Vi ønsker at Track-objekter skal være uforanderlige (immutable) etter at de er instansiert. Dvs. at det kun er klassens egne medlemsfunksjoner som skal kunne endre på verdiene. Hvordan oppnår vi dette?

d) For Playlist-klassen skal du implementere konstruktøren **`Playlist::Playlist(string playlistfilename)`** og **`Playlist::add(string trackline)`** samt de medlemsvariablene du har bruk for i denne klassen. Konstruktøren skal lese fila som parameteren navngir linje for linje. Innholdet i fila er som beskrevet i den innledende teksten til denne oppgaven. Hver linje i fila er informasjon om enkelttrack og medlemsfunksjonen add skal brukes for å legge ett og ett track til fortløpende slik at de lagres i samme rekkefølge som i playlistfila. I denne oppgaven kan du godt benytte funksjonene som er beskrevet i oppgave 1 for å splitte og trimme teksten. Du kan med fordel benytte deg av biblioteksfunksjonen **`istream& getline (istream& is, string& str)`** for å lese playlist-fila.

e) Hvordan kan du deklarere **`Playlist:add()`** slik at det bare er konstruktøren (og andre medlemsfunksjoner i Playlist) som kan benytte denne funksjonen?

f) I oppgave 2c) har vi bestemt at Track-objekter skal være uforanderlige (immutable). Dette skaper et problem når du skal legge til noder i lista. Hva kan du gjøre for at Playlist-klassen (og bare denne) skal få mulighet til å endre på Track-objekter?

g) Implementer operatoren **<<** slik at det er mulig å få skrevet ut track-informasjon i samme format som i inputfila. Hvis du har en instans **`Track t`** skal det være mulig å bare skrive **`cout << *t;`** Hvorfor kan vi ikke implementere denne operatoren som medlem av klassen?

h) Implementer en medlemsfunksjon **`Track* Playlist::next()`** . Denne funksjonen skal kunne brukes til å bla i listen. Første gang funksjonen kalles skal den returnere første node i lista, andre gang den kalles skal andre node returneres osv. Hvis lista er tom eller du er kommet til slutten av lista skal funksjonen returnere NULL.

i) Forklar hvorfor det er mulig å bruke løkka under for å bla seg igjennom alle noder i en liste ved hjelp av next-funksjonen. Her er vi ute etter en forklaring på hvordan betingelses-delen av while-løkka fungerer.

```
Playlist favourites("mylist.txt");
Track *t;
while(t = favourites.next()){
  cout << *t;
}
```

Oppgave 3: Arv (30%)

I denne oppgaven skal du jobbe videre med klassene du laget i oppgave 2. Mens implementasjonen i oppgave 2 var en lenket liste hvor track-objektene var organisert i samme rekkefølge som de ble lest fra fil, skal vi nå bruke arv og virtuelle funksjoner og lage subklasser av `Playlist` med endret oppførsel.

Merk at den eneste klassen du skal lage subtyper av er `Playlist`. Det er i denne oppgaven greit å modifisere din implementasjon av `Playlist` og `Track` hvis du mener det er nødvendig for å kunne løse oppgaven. Husk å beskrive og vise hvilke endringer du evt. gjør.

I evalueringen legger vi vekt på at du viser kunnskap om og fornuftig bruk av arv samt at du velger hensiktmessige løsninger (ikke unødvendig komplekse løsninger, minst mulig kode, færrest mulig endringer i Playlist og Track-klassene etc.).

a) Lag en subklasse av `Playlist` kalt `RandomPlaylist` hvor `next()` returnerer Track-objektene i tilfeldig rekkefølge. Du skal i denne oppgaven ikke endre på den faktiske rekkefølgen av den lenkede listen, men kun legge til eller endre på funksjonalitet som har med rekkefølgen `next()` returnerer objektene i. Forklar løsningen din og dine valg.

b) Lag en subklasse av `Playlist` kalt `RatedPlaylist` hvor nodene i lista er organisert slik at tracks med høyest rating kommer først. Tracks som har lik rating skal sorteres alfabetisk på artist og tittel. Forklar løsningen din og dine valg.