Oppgaver

PROG1003 -
Objekt-orientert programmering

Forord

Dette kompendie/hefte innholder oppgaveteksten for ulike oppgaver i emnet ”PROG1003 – Objekt-orientert programmering” ved NTNU.

Opplysninger om emnet er å finne på: **http://folk.ntnu.no/frh/ooprog**

Under ”Opplegg” er det angitt hvilke oppgaver i dette heftet som til enhver tid er relevante som ukeoppgaver.

(Data)filer som oppgavetekstene henviser til og alle løsningsforslag er begge deler å finne på katalogen ”OPPGAVER”.

Evt. nye/andre oppgavetekster vil bli lagt til bakerst i dette heftet.

NTNU

Frode Haug

Oppgave 1

Ta utgangspunkt i løsningsforslaget for **oppgave** nr.17 (OPPG_17.c) fra i høst. Skriv om koden fra C til C++. Pass fortsatt på at poengsummen *alltid* skrives ut med *eksakt* tre desimaler.

Oppgave 2

Ta utgangspunkt i **eksempel** nr.17 (EKS_17.c , *ikke oppgave* nr.17 – som ovenfor) fra i høst. Skriv om koden fra C til C++, men *kutt ut/ta vekk all bruken av array*. Husk fortsatt å få utskrift av aktuelle tall høyrejustert. Utvid koden med at dagnummeret det falt mest nedbør og dette antall mm også skrives ut på skjermen (er flere størst, skrives bare den første ut).

Oppgave 3

Ta utgangspunkt i løsningsforslaget til oppgave nr.2 (rett ovenfor).
Innfør, implementer og bruk i størst mulig grad de to funksjonene:

`int lesTall(const char t[], const int min, const int max)`
C++ -versjon av `lesInt(…)` fra `LesData.h` (i høst/PROG1001).

`float gjennomsnitt(const int verdi1, const int verdi2)`
Beregner og returnerer `float`-gjennomsnittet av to `int`-verdier.

Legg merke til at lesingen av 'J'/'N' to steder, også kan erstattes av å lese '1'/ '0'.

Oppgave 4

Lag et program som inneholder en `struct Tid` med `int`-datamedlemmene `time`, `minutt` og `sekund`. Lag også funksjonen:

`struct Tid storst(const struct Tid tid1, const struct Tid tid2)`

som finner ut hvilken av de to parametrene som er størst/senest/høyest, og returnerer denne.

Lag (i `main`) også *fire* `Tid`-variable som initieres til noen passende/ønskede verdier. Programmet skal så finne ut (vha. `storst(…)`) hvilken av de fire structene som totalt er størst/senest/høyest, og skrive så til slutt ut (på skjermen) dennes datamedlemmer.

Oppgave 5

Lag et program inneholdende:

- ```
  const int STRLEN           = 40;  ///< Max.tekstlengde.
  const int MAXFORELESNINGER = 10;  ///< Max.antall forelesninger.
  ```
- ```
  struct Forelesning {
      char emne[STRLEN/2],
           foreleser[STRLEN],
           sted[STRLEN/2];
      int  timeStart, minuttStart,
           timeSlutt, minuttSlutt;
  };
  ```
- `main` med:
  - En array med `MAXFORELESNINGER Forelesning`-structer.
    NB: Arrayen inneholder struct-variable, og *ikke* pekere til structer.
    Men, de to funksjonene (nedenfor) *skal* ta imot og håndtere *peker til en struct*.
  - To `for`-løkker som sørger for at henholdsvis *alle* structene først får lest inn *alle* sine data, og deretter skrevet dette ut igjen på skjermen.
- Alt dette siste gjøres vha. funksjonene:
  - `void forelesningLesData(Forelesning* f)`
    Leser inn og fyller *alle* structens seks data med rimelige verdier.
    Bruk den nye `LesData2.h` til å lese tall (og tegn, men *ikke* tekster lengre).
  - `void forelesningSkrivData(const Forelesning* f)`
    Klarer du å få til at en-sifrede minutter skrives med en innledende '0' (null)?

Oppgave 6

Lag et program som inneholder fire string-variable. Eksperimenter litt med disse via mekanismene/funksjonaliteten introdusert og brukt/vist i EKS_02.CPP.
Dvs:

- definer string-variable
- initier en eller flere av dem
- les inn enkeltord og lengre stringer/tekster
- skriv ut stringene
- kopier stringer over til/i hverandre
- sammenlign stringer
- skjøt sammen og forleng stringer
- finn stringers lengde

Ekstra:

- Bruk/bli kjent med flere av de andre ti-talls funksjonene ifm. string-klassen

(**NB:** Løsningsforlag til denne oppgaven finnes ikke.)

Oppgave 7

Lag et program som inneholder:

- `struct Klokkeslett` med `int`'ene `time`, `minutt` og `sekund`
- de tre funksjonene:
  - `void les(string & s)`
    Leser inn og fyller `s` med en *hel* tekstlinje (ikke bare ett enkeltord).
    Det skal *ikke* godtas at brukeren *kun* skriver '\n' – dvs. en tom tekst.
  - `void les(Klokkeslett & klokkeslett)`
    Får fylt *alle* structens datamedlemmer med rimelige verdier.
    Bruk den nye `LesData2.h` til å lese alle tallene.
  - `void skriv(const Klokkeslett klokkeslett)`
    Skriver structens datamedlemmer på formen: TT:MM:SS
    gjerne med innledende '0' (null) foran alle en-sifrede tall.
- `main` med:
  - en array med tre stk. `string` (*ikke* pekere til slike)
  - en array med tre stk. `Klokkeslett` (*ikke* pekere til slike)
  - *to* `for`-løkker som leser inn og får fylt *alle* elementene i de to arrayene ovenfor ved å bruke aktuell overloaded `les`-funksjon
  - *to* `for`-løkker som skriver ut *alt* innholdet i de to arrayene.
    Ifm. klokkeslettene brukes `skriv`-funksjonen ovenfor.

Oppgave 8

Lag et program som inneholder:

- `struct Kamp` med `string`'ene `hjemmeNavn` og `borteNavn` og `int`'ene `hjemmeMaal` og `borteMaal`.
- den (initielt tomme) globale `vector`'en `gKampene`, med *pekere* til kamper
- `main` som:
  - lager nye kamper og legger disse inn i `gKampene`, så lenge brukeren ønsker.
    Hver kamp sine data leses inn vha. funksjonen:
    `void kampLesData(Kamp & kamp)`
  - skriver ut *alle* de lagrede kampene sine data, bl.a. vha. funksjonen:
    `void kampSkrivData(const Kamp* kamp)`
  - til slutt sletter *alle kampene tilpekt og pekerne lagret* i `gKampene`

Oppgave 9

Lag et program som inneholder klassen `Dato` med `int`-datamedlemmene `dag`, `maaned` og `aar` og med medlemsfunksjonene:

- `Dato()`
  Setter datamedlemmene til 1/1-2000 (1.januar 2000)
- `Dato(int d, int m, int a)`
  Setter datamedlemmene til parametrenes verdier
- `void lesData()`
  Leser datamedlemmenes verdier fra brukeren
  (vi lar det *kun* være tillatt med 30 dager i *alle* måneder)
- `void skrivData()`
  Skriver datoen på formen: DD/MM-AAAA (dag / måned - år)
- `bool sammeAar(const Dato dato)`
  Returnerer `true`/`false` til om eget `aar` er lik med `dato.aar`
- `bool sammeAarsdag(const Dato dato)`
  Returnerer `true`/`false` til om egen dag i året er lik med parameteren sin

Klarer du også å lage? :

- `bool tidligereEnn(const Dato dato)`
  Returnerer `true`/`false` til om egen totale dato (både dag, måned og år) kommer før enn parameteren sin. For å slippe å beregne alt eksakte omkring skuddår eller ei, at månedene har ulikt antall dager, samt at dette ikke blir for avansert, så lar vi det som nevnt bare *alltid være 30 dager i alle måneder*.

Lag en `main` som bruker *alle* de lagde medlemsfunksjonene ved å:

- Lage to dato-objekter, der *kun* det initieres til en eller annen selvvalgt dato
- Be den andre datoen om selv å lese inn sine data
- Skrive ut begge datoenes data
- Skrive ut om datoene:
  - er i det samme året eller ei
  - har den sammme årsdagen (dag og måned) – uansett år
  - faktisk er *helt* identiske, og om så *ikke* er tilfelle, skriver hvem av dem som kommer først

Oppgave 10

Skriv om EKS_06.CPP til å inneholde `class` i stedet for `struct`. Funksjonaliteten og dialogen med brukeren på skjermen (både innlesning og utskrift) skal fortsatt våre *eksakt* den samme. Den *eneste* forskjellen skal være at kommandoen '2' er kuttet ut,
og kommandoen '1' er erstattet med 'D' (for «Display»).

Oppgave 11

Lag et program med de to klassene:

`Posisjon` - som inneholder:

- datamedlemmene: `int`en `grad` (0-90 eller 0-180), `minutt` (0-59) og `sekund` (0-59), samt `char`en `retning` ('N'/'S' eller 'E'/'W')
- en constructor som initierer/nullstiller alle datamedlemmene
- `void lesData(int gr, char r1, char r2)` som sørger for at *alle* datamedlemmene blir lest inn fra brukeren, og sikrer at de *tilordnes korrekte* verdier (dvs: `grad` er 0 - `gr`, `minutt` og `sekund` er begge 0 - 59, `retning` er `r1` *eller* `r2`)
- `void skrivData()` som skriver ut datamedlemmene eksempelvis på formen: 76º23'12" N (dvs. `grad-minutt-sekund-retning`)

`Skip` - som inneholder:

- datamedlemmene: `int`en `nr` og `string navn`, samt `breddegrad` og `lengdegrad` (begge er av klassetypen `Posisjon`)
- en constructor som setter `nr` og `navn` lik de to parametrene som kommer inn
- `void lesData()` som leser inn skipets `breddegrad` (max. 90, 'N' og 'S') og `lengdegrad` (max. 180, 'E' og 'W') vha. `lesData(...)` i `Posisjon`
- `void skrivData()` som sørger for at alle skipets data skrives ut. Denne bruker bl.a. `skrivData()` i `Posisjon`

Lag til slutt et hovedprogram som oppretter tre ulike `Skip`-objekter, og som sørger for at alle deres data både blir lest inn og skrevet ut på skjermen.

Oppgave 12

Vi har følgende klasse og prototype på medlemsfunksjoner:

```
class Kunde {
   private:
      string navn;
      vector <float>* kontoer;     // Peker til kontoer og deres beløp.
   public:
      Kunde(const int n);
```

- Setter dynamisk `kontoer` til å *peke* på en `float-vector` som er `n` lang, og alle elementene initieres til `0.0F`

```
      ~Kunde();
```

- Sletter *alt* det som objektet har sagt `new` om mens det har levd

```
      void lesData();
```

- Spørr om kundens navn og beløpet som står inne på *hver* av de aktuelle kontoene

```
      void skrivData();
```

- Skriver navnet og innestående beløp på *alle* aktuelle kontoer

```
};
```

Lag videre et program (`main`) som innholder en `vector` med pekere til `Kunde`-objekter. For hver av dem, leses først antall kontoer kunden har. Dette antallet sendes med til constructoren i det objektet opprettes. Objektet leser deretter selv inn navnet og beløpet på hver av kontoene. Til slutt legges objektet inn i vectoren. Det opprettes nye slike objekter så lenge brukeren ønsker.
Når registreringen altså er ferdig, gås det gjennom *alle* kundene, og hver enkelts data skrives ut på skjermen. Helt til slutt slettes *alle* Kunde-objektene og deres allokerte data.

Oppgave 13

Lag klassen `Publikasjon` med datamedlemmene: `string navn;  int aar;`
Fra denne, lag de to avledede klassene:
`Bok` med datamedlemmene: `string forfatter, ISBN;  float pris;` og
`Blad` med datamedlemmene: `int nummer, ukeNr, aarsAbonnement;`
Alle datamedlemmer i *alle* klassene *skal* være `private`.
Inni hver av de tre klassene *skal* det lages funksjonene `lesData()` og `skrivData()` som hver for seg sørger for at *alle* klassens data blir lest inn /skrevet ut, også de arvede. Dette aller siste gjøres ved at funksjonene i de avledede klassene kaller de tilsvarende funksjonene i baseklassen.
Lag `main` som tester de lagde klassene og koden ved å lage ett `Bok`- og ett `Blad`-objekt. Deretter kalles begge objektenes `lesData()` og deretter deres `skrivData()`.

Oppgave 14

Lag et program (primært bestående av `main`), der det prøves/eksperimenteres/utforskes flere av funksjonene i `string`-klassen (jfr. EKS_15.CPP).
Ignorer (ikke bruk) de funksjonene som har med `iterator` å gjøre.

(**NB:** Løsningsforlag til denne oppgaven finnes ikke.)

Oppgave 15

Lag klassen `Kjoretoy` med datamedlemmet: `string registreringsNr`
Fra denne, lag de to avledede klassene `Bil` med datamedlemmet `int antPassasjerer` (= antall *nåværende* passasjerer i *tillegg* til føreren) og `Vogntog` med datamedlemmet `float tonnLast` (= antall *nåværende* tonn med last).
Alle datamedlemmer i *alle* klassene *skal* være `private`.
Inni hver av de tre klassene *skal* det lages de ***virtuelle*** funksjonene `lesData()` og `skrivData()` som hver for seg sørger for at klassens data blir lest inn/skrevet ut, også de arvede. Dette aller siste gjøres ved at funksjonene i de avledede klassene kaller de tilsvarende funksjonene i baseklassen.

Lag også (inni alle klassene) ***pure virtual*** funksjonen `bool tomt()` som returnerer `true`/`false` til om henholdsvis bilen er tom for passasjerer eller at vogntoget er tomt for last.
Lag `main` som:

- inneholder (globalt) en `vector` med `Kjoretoy`-pekere
- har en løkke som går så lenge brukeren ønsker, og som for hver runde oppretter en ny `Bil` *eller* et nytt `Vogntog` (ut fra det brukeren angir). Leser inn det aktuelle objektets data, før det til slutt i løkka legges inn i `vector`en.
- etter at løkka ovenfor er ferdig:
    - skrives antall objekter i `vector`en
    - skrives *alle* objektene i `vector`en ut med sine data
    - skriver ut indeksen i `vector`en for *alle* de kjøretøyene som er tomme
    - sletter *alle* objektene i `vector`en *og* pekerne til disse

Oppgave 16

Lag klassen `Publikasjon` med datamedlemmene: `string tittel; float pris;`
Lag de to avledede klassene: `Bok` med datamedlemmene: `string forfatter; int antallSider;` og `Magasin` med datamedlemmene: `int aar, volum;`
Alle datamedlemmer i *alle* klassene *skal* være `private`.

Inni *hver av de tre klassene skal* det lages:

- en constructor som leser inn objektets data fra fil (et filobjekt er parameter)
- funksjonen `virtual void skrivTilFil(ofstream & ut)` som hver for seg sørger for at *alle* klassens data blir skrevet ut på `ut`, også de arvede

Filen («OPPG_16.DTA») det leses fra, har følgende format for en *post*/ett objekt:

```
En Bok:        B  <Pris>  <Tittel>
               <Antall sider>  <Forfatter>
Et Magasin:    M  <Pris>  <Tittel>
               <År>  <Volum>
```

Lag `main` som:

- lager og åpner et filobjekt for innlesning og et for utskrift
- kommer med en melding om det ikke er mulig å finne innlesningsfilen
- leser starten på en og en post (dvs. 'B' eller 'M') til det er slutt på filen
- for hver post: oppretter aktuelt objekt (ut fra innlest bokstav), får objektets constructorer til å lese inn resten av dataene i posten, og til slutt legger objektet inn i en `vector` med `Publikasjon*`
- skriver ut *alle* objektene og *alle* deres data vha. `skrivTilFil(…)` -ene til filen «OPPG_16.DT2». Denne *skal ha eksakt samme format som filen det leses inn fra*.

Oppgave 17

Lag et program som leser fra filen «OPPG_17.DTA» og finner ut hvilken *desember*dato det har vært kaldest (jfr. TAN-kolonnen) og hva denne temperaturen er.
Er det flere datoer med denne minimumstemperaturen, skal den første angis.
Det er *ikke* tillatt å editere vekk tilsynelatende «ekstra» linjer i starten og slutten av filen.
(*Om* det skulle ha noen betydning: hver linje har en max. lengde på 160 tegn.)

Utvidelse/ekstra:

Brukeren angir:

1. hvilken måned det skal letes i (0 betyr *alle* måneder).
2. om det skal letes etter minimums- eller maksimumstemperaturen.

Oppgave 18

Studer/sett deg godt inn i de ulike aktuelle (`list`, `stack`, `queue` og `map`) containernes tilgjengelige funksjoner/operasjoner. Gjør dette ved å bruke/følge linkene under: «**String-klassen, Containers og Algorithms**» og «**Quick Reference Cards/Cheat Sheets**» på «**Ressurser/mer stoff**»-siden på emnets hjemmeside.

Lag ett eller flere program der du bruker flere/mange av funksjonene (jfr. EKS_21.CPP og EKS_22.CPP). Containerne må gjerne inneholde: `int`, `float`, `char` eller `string`.

(**NB:** Løsningsforlag til denne oppgaven finnes ikke.)

Oppgave 19

Gjør noe lignende til det i forrige oppgave (nr.18), bare at bruk/vektlegg «kun» `list`, `map` og iteratorer på disse to container-typene (jfr. EKS_23.CPP og EKS_24.CPP). Bruk/eksperimenter også med Algorithms (jfr. EKS_25.CPP) ifm. koden du lager/skriver.

(**NB:** Løsningsforlag til denne oppgaven finnes ikke.)

Oppgave 20

Endre/utvid EKS_27*.* med å legge til et nytt menyvalg (og funksjon/kode) som sletter den siste/bakerste personen

(**NB:** Løsningsforlag til denne oppgaven finnes ikke.)