2016-03-30.kl.14-19

Tentaupplägg

TIPS 1:

Läs igenom ALLA uppgifterna. Välj den du känner är lättast först. Det kan gärna ta 10-20 minuter. Försök skriva saker som kan vara problem i uppgifterna. Är det något du absolut kommer att fastna på så kanske det är fel uppgift att ge sig på. Tiden du lägger på att noga läsa uppgifterna tjänar du in på att välja rätt uppgift.

TIPS 2:

Kolla ibland till kommunikationsfönstret. Det kan ha kommit information till alla utan att ni skickat in en fråga. Kanske gäller det dig också (d.v.s. den uppgift du jobbar med).

TIPS 3:

Om ni har problem med kompilator, Emacs eller annat som INTE har med uppgifterna att göra, räck upp handen så kommer en assistent. Detsamma gäller om hur man kopierar givna filer
" cp given_files/* . " eller liknande.

Frågor om själva uppgifterna tar vi i första hand via tentasystemet.

I körexemplen har vi markerat det som användaren matar in på tangentbordet med ***fet och kursiverad*** stil. Tänk på att körexemplen bara är *ett* exempel på när programmet körs. Testa ditt program noga och tänka över hur programmet skall fungera vid andra indata.

Vi hinner normalt sett inte svara på frågor de sista 10 utminutera på tentan. Då ägnar vi all tid åt att rätta uppgifter för att alla skall hinna få svar innan ni går hem.

Betygsgränser:	**Tid**	**DI/EL**	**SVP**
1 poäng	19:00	Betyg 3	Betyg G
3 poäng	18:00	Betyg 4	
3 poäng	16:45		Betyg VG
3 poäng	16:30	Betyg 5	
4-6 poäng	18:00	Betyg 5	
4-6 poäng	18:15		Betyg VG

Bonustid från laborationerna tillkommer till dessa tidsgränser. Tiden för betyg 3/G överstiger dock aldrig 5 timmar.

OBS: Delpoäng delas inte ut på uppgifterna. För att få poäng på en uppgift måste man alltså lösa uppgiften helt (och enligt specifikation).

Lycka till med tenterandet och hoppas att alla får G på minst en uppgift idag.
M.v.h.
/Torbjörn (examinator)

2016-03-30.kl.14-19

Uppgift 1 - Skriva många variabler [1p]

Ibland när man programmerar behöver man en hel drös variabler! Antag att man vill lagra 100 heltal, då kan man behöva många! Eller, nej, tänker du kanske - vi kan ju alltid använda ett fält. I den här uppgiften skall du inte använda fält (eller vector, eller något annat sådant).

Skriv ett program som låter användaren mata in hur många variabler som den vill ha. Användaren skall sedan ange datatypen för dessa (t.ex. int, char, bool, string) och sedan ett prefix för variablernas namn (ett ord som endast innehåller bokstäver). Användaren skall också ange hur dessa skall initieras (t.ex. med {}, eller = true, etc.). Programmet skall sedan generera ett C++-program med dessa variabler! Programmet skall skrivas ut på filen variables.cpp. Det genererade programmet skall ha korrekt formatering (rätt indentering och klamrar på "rätt" ställe).

Körexempel 1:

```
Mata in antal variabler: 5
Mata in datatyp: int
Mata in namnprefix: age
Mata in initiering: {}

Klart! Ditt program med 5 ints finns nu på filen variables.cpp!
```

Filen variables.cpp skall efter körningen ovan se ut på följande sätt:

```
int main()
{
  int age0 {};
  int age1 {};
  int age2 {};
  int age3 {};
  int age4 {};

  return 0;
}
```

Körexempel 2:

```
Mata in antal variabler: 2
Mata in datatyp: string
Mata in namnprefix: happyword
Mata in initiering: = "happy"
```

```
int main()
{
  string happyword0 = ”happy”;
  string happyword1 = ”happy”;

  return 0;
}
```

```
Klart! Ditt program med 2 strings finns nu på filen variables.cpp!
```

Uppgift 2 - Läsa många variabler [2p]

Läs uppgiftstexten i uppgift 1. En fil "VARIABLES.CPP" med samma format som beskrivs i uppgift 1 finns i mappen given_files. I denna uppgift skall du skapa programmet *reconstruct* som går baklänges och utifrån filens innehåll få det som användaren en gång matade in. Filnamnet på den fil som skall läsas igenom anges på kommandoraden. Du kan anta att filen alltid existerar.

OBS: Ditt program behöver bara klara det format som anges i uppgift 1, inte allmän c++-kod.

Körexempel 1 ('$' markerar kommandoprompten):

```
$ reconstruct VARIBLES.CPP
Filen hade 10 variabler av typen char.
Variablerna hade namn som började på "letter".
Variablerna initierades med {'?'}.
```

Körexempel 2 ('$' markerar kommandoprompten):

```
$ reconstruct
FEL! Starta programmet med 1 argument.
```

Uppgift 3 - Sortera ord [2p]

Skriv ett program som låter användaren mata in ord tills man avslutar med ctrl-d. Orden skall sedan komma ut (ett per rad) sorterat i bokstavsordning*. Du skall använda dig av *vector* i detta program. Du skall ha en funktion *insert_sorted* som tar en vektor och en sträng som parametrar och som sätter in strängen "på rätt plats" i vektorn.

KRAV: Du får inte använda någon av standardbibliotekets inbyggda algoritmer för att lösa själva sorteringen (t.ex. funktionen *sort*).

Körexempel:

```
Mata in så många ord du vill: Hejsan svejsan idag är jag på glatt
humör hur glad är du?
[Ctrl-d]

Hejsan
du?
glad
glatt
humör
hur
idag
jag
på
svejsan
är
är
```

TIPS: Man kan jämföra strängar med "<".

---

* Observera att denna form av "bokstavsordning" innebär att versaler (stora bokstäver) alltid kommer före gemener (små bokstäver). Alltså kommer t.ex. ordet "Gurka" före ordet "alabaster".

2016-03-30.kl.14-19

Uppgift 4 - Tentasystemet [1p]

För att få miljön som du nu skriver tenta i så måste systemet konfigureras i förväg. När man konfigurerar en datortenta så finns det ett begrepp som kallas *kopplade tentor*. T.ex. är denna tenta egentligen *tre* tentor, en för TDIU08, en för TDIU01 och en för 726G78. Det enda som skiljer dessa tre är egentligen kurskoden, i övrigt är ju allting annat exakt lika.

Ett annat exempel på kopplade tentor är när det är så många tentander att lokalerna inte klarar av alla på samma gång. Om det t.ex. är 200 studenter som skall tentera så måste man bryta upp tentan över två tillfällen, t.ex. ett på förmiddagen (8-13) och ett på eftermiddagen (14-19). Återigen kan vi se dessa tentor som kopplade, eftersom det egentligen är samma tenta. Nu är detta fenomen inte precis som det första exemplet, eftersom det i detta fall kan vara (troligtvis) olika uppgifter på tentorna. För båda exemplen gäller dock att en tentand inte får gå på två tentor som är kopplade. För att kunna skilja dessa två fall åt måste vi precisera oss något. Vi kallar därför det första fallet ovan för *simultana kopplade tentor*. Medan i det andra fallet säger vi bara att de är *kopplade*.

Det finns andra situationer då kopplade tentor kan dyka upp i detta tentasystem, men det behöver vi inte bekymra oss om nu. Din uppgift är att skriva ett program som läser in två kopplade tentor (kurskod, start- och sluttid) och avgör ifall dessa två är simultana. Två tentor är simultant kopplade om de överlappar på något sätt i tiden. Du skall i din lösning representera en tenta med en *struct.*

Körexempel 1:

```
Mata in kopplad tenta 1
  Mata in kurskod: TDIU08
  Mata in starttid: 08:00
  Mata in sluttid: 13:00

Mata in kopplad tenta 2
  Mata in kurskod: 726G78
  Mata in starttid: 08:00
  Mata in sluttid: 13:00

Tentorna för TDIU08 och 726G78 är
simultant kopplade.
```

Körexempel 2:

```
Mata in kopplad tenta 1
  Mata in kurskod: TDDD11
  Mata in starttid: 08:00
  Mata in sluttid: 13:00

Mata in kopplad tenta 2
  Mata in kurskod: TDDD11
  Mata in starttid: 14:00
  Mata in sluttid: 19:00

Tentorna för TDDD11 och TDDD11 är
kopplade.
```

Körexempel 3:

```
Mata in kopplad tenta 1
  Mata in kurskod: TDIU08
  Mata in starttid: 08:00
  Mata in sluttid: 13:00

Mata in kopplad tenta 2
  Mata in kurskod: ABCD01
  Mata in starttid: 08:00
  Mata in sluttid: 15:00

Tentorna för TDIU08 och ABCD01 är
simultant kopplade.
```

Körexempel 4:

```
Mata in kopplad tenta 1
  Mata in kurskod: ROLI64
  Mata in starttid: 08:00
  Mata in sluttid: 13:00

Mata in kopplad tenta 2
  Mata in kurskod: TRIS74
  Mata in starttid: 10:00
  Mata in sluttid: 17:00

Tentorna för ROLI64 och TRIS74 är
simultant kopplade.
```

TIPS: Gör en funktion *before* som tar två tider som parametrar (t.ex. fyra heltal eller två strängar) och returnerar sant om den första tiden är före den andra tiden och falskt annars.