2018-01-09 kl. 08-13

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.

TIPS 4:

Du kan kompilera med aliasen **g++11** eller **w++11**.

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 minuterna på tentan. Då ägnar vi all tid åt att rätta uppgifter för att alla skall hinna få svar innan ni går hem.

Betygsgränser:	**Tid**	**TekFak**
1 poäng	13:00	Betyg 3
3 poäng	12:00	Betyg 4
3 poäng	10:30	Betyg 5
>=4 poäng	12:00	Betyg 5

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

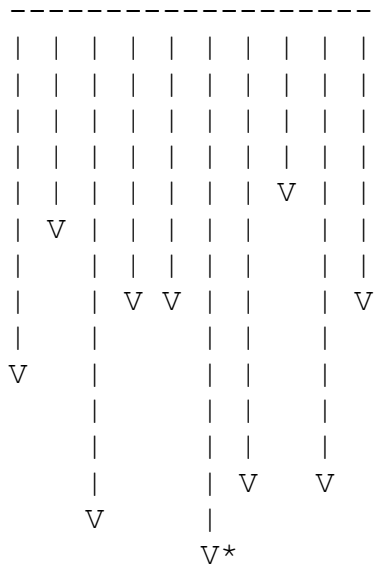
Lycka till med tenterandet och hoppas att alla får G på minst en uppgift idag.
M.v.h.
/Torbjörn (examinator)

2018-01-09 kl. 08-13

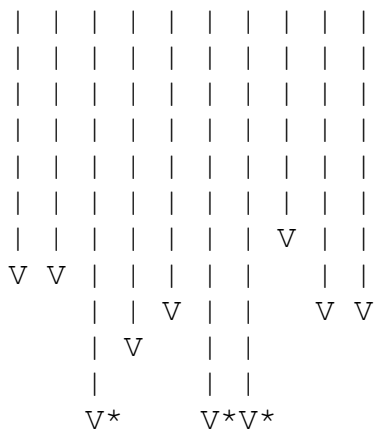
Uppgift 1 - Istappar [1p]

Istappar är vackra men kan vara farliga. Ju längre de blir desto tyngre blir de också. Plötsligt släpper de och faller mot marken. Då får man se upp! Skriv ett program som ritar ut istappar och markerar vilken istapp som troligtvis släpper först (den längsta) med en asterisk. Istapparnas skall vara mellan 5 och 15 tecken långa. Denna längd skall slumpas fram. Se körexemplet på hur istapparna skall se ut. Antalet tappar skall vara 10, men det skall inte vara svårt att ändra koden så att de är fler (eller färre). Om det finns flera istappar som är lika långa så skall programmet markera alla dessa.

Körexempel 1:



Körexempel 2:



TIPS 1: Det finns en given funktion i filen die_roll.cpp som visar hur man kan slumpa tal.
TIPS 2: Använd ett fält.

2018-01-09 kl. 08-13

Uppgift 2 - Mecka bil [1p]

En verkstad behöver en del av sitt datorsystem implementerat. Följande delar saknas:

- En datatyp Car_Type som innehåller märke och modell (en sträng), år (ett heltal), miltal (ett flyttal) och motor (en Engine_Type).
- En datatyp Engine_Type i sig innehåller en typ (en sträng t.ex "V8" eller "Rak 6") och en volym (ett heltal t.ex. 1200).

Utöver dessa två datatyper behövs rutiner för att hantera dem.

- Du skall ha ett underprogram *get* för vardera datatyp. Rutinerna läser in till **en** variabel av respektive typ. Get för Car_Type skall avnända sig av get Engine_Type
- Du skall ha ett underprogram *put* för vardera datatyp. Rutinerna skriver ut **en** variabel av respektive typ. Put för Car_Type skall använda sig av put för Engine_Type
- Du skall även ha ett underprogram *replace* som tar en parameter av typen Car_Type, en parameter av typen Engine_Type och stoppar in denna nya motor i bilen. Den gamla motorn skall returneras från funktionen.
- Tänk till om hur parametrarna skall vara i dina underprogram. D.v.s. det är viktigt med const och referens där det är lämpligt.

Det finns ett givet huvudprogram, carmeck.cpp, som visar hur typerna och underprogrammen är tänkta att användas. Du skall lägga till dina typer och underprogram i filen. I övrigt får du **inte** ändra på detta huvudprogram.

Körexempel:

```
Välkommen till CarMeck-2000!
Mata in en bil:
  Modell: Volvo V70
  År: 2013
  Miltal: 7200.3
  Motor: 1300 cc Rak 6

Tack! Du matade in följande bil:
Volvo V70 (2013), har gått 7200.3 mil, Motor: Rak 6, volym 1300

Mata in en extra motor: 2400 cc V8
Tack! Du matade in följande motor:
V8, volym 2400

Vi kommer nu ersätta motorn på din Volvo V70. Din bil är nu:
Volvo V70 (2013), har gått 7200.3 mil, Motor: V8, volym 2400

Vroom!
```

2018-01-09 kl. 08-13

Uppgift 3 - Kodord [2p]

Följande är ett vanligt sätta att koda ett hemligt meddelande som man vill sända mellan två parter utan risk för att det skall upptäckas och avkodas av fienden. Man bestämmer sig på förhand för ett litterärt verk, något som båda parter kan införskaffa utan att det påkallar någon uppmärksamhet. Denna bok kommer användas som en referens. Själva meddelandet är sedan bara en lång sekvens av tal-tripler, där första talet är sidnummer, andra talet är vilket ord på sidan och tredje talet är bokstaven inom det ordet. Vi tänker oss i denna uppgift ett liknande upplägg, fast vi skippar sidnumret och jobbar således bara med talpar.

Skriv ett program som avkodar meddelandet i MESSAGE.TXT, en fil som bara består av talpar, t.ex:

```
3 2 5 1 10 3 11 0 14 1
```

Om man vill ha ett mellanslag i meddelandet så är andra siffran i talparet 0. Första siffran ignoreras då.

Detta skulle alltså betyda "Tredje ordet, bokstav två. Femte ordet, bokstav ett. Tionde ordet, bokstav tre". Referenstexten ligger i REFERENCE.TXT och är bara en lång text där varje ord separeras med vita tecken (blanksteg, enter m.m.). T.ex.

```
My dear Philip,
In earlier years I would have rejoiced
over your arrival, but alas, now I do not.
```

Vid avkodning av meddelande ovan, skulle detta alltså bli "hej b".

TIPS: Du kan anta att ord-talen (det första talet i varje talpar) bildar en strikt stigande sekvens. D.v.s. man behöver aldrig leta "bakåt" i referenstexten.

TIPS2: Det finns ett längre meddelande att testa med i MESSAGE2.TXT, meddelandet hör ihop med referensen i REFERENCE2.TXT.

2018-01-09 kl. 08-13

Uppgift 4 - Veckopeng [2p]

Lars har kommit på ett smart sätt att ge sina barn veckopeng. Barnen får själv ange hur mycket de "tar ut", och det dröjer då porportionellt lång tid innan de får ta ut pengar igen. För att lösa detta använder han ett kösystem. Han ställer upp en kalender som har en ruta per dag och ger barnen var sin markör (med barnets initial på). Det får bara plats en markör i varje ruta i kalendern, d.v.s. det är begränsat till maximalt ett uttag per dag. Barnen drar lott om vem som får göra ett uttag på dag 1, och sedan dag 2 o.s.v.

Vi antar att Lars har barnen Ada, Berta och Caesar. Då ser kalendern från början ut så här:

```
ABC
```

På dag 1 gör Ada ett uttag på 10 kronor. Hennes nästa uttag blir då om 10 dagar (på dag 11):

```
-BC-------A
```

På dag 2 gör Beda ett uttag på 5 kr. Hennes nästa uttag blir då om 5 dagar (på dag 7):

```
--C---B---A
```

På dag 3 gör Caesar ett uttag på 8 kr. Hans nästa uttag skulle ha blivit på dag 11 men den platsen är redan tagen av Ada, så han hamnar på dag 12:

```
------B---AC
```

Därefter sker inga uttag förränn dag 7, då Beda gör ett nytt uttag. Om Beda tar ut 1, 2 eller 3 kr så kommer hon få ta ut ytterligare en gång innan Ada!

Skriv ett program som låter användaren mata in barnen (deras initialer) och sedan en godtyckligt lång rad med heltal. Det är maximalt 10 barn. Varje heltal som matas in är uttaget för nästa barn i turordningen. Programmet skall sedan mata ut kalenderns strängrepresentation (som exemplen ovan).

Körexempel 1:

```
Mata in barn: XY
Mata in uttag: 2 4

--X--Y
```

Körexempel 2:

```
Mata in barn: ABUM
Mata in tal: 9 6 10 5 5 4 1

----------A-UBM
```