2017-04-19.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**
1 poäng	19:00	Betyg 3
3 poäng	18:00	Betyg 4
3 poäng	16:30	Betyg 5
4-6 poäng	18:00	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)

Uppgift 1 - O va' bra! [1p]

Skriv ett program som läser in två ord (orden separeras med ett godtyckligt antal vita tecken, d.v.s. mellanslag, enter, tab) och som sedan skriver ut orden med några extra 'o'. För båda orden så skall dessa extra o:n sättas in efter sista o:et i ordet. Antalet nya o:n skall vara lika många som antalet bokstäver i ordet, borträknat antal ursprungliga o:n. Om vi t.ex. tar ordet "kontroll" så skall 6 extra o:n sättas in efter det andra o:et (eftersom kontroll innehåller 8 bokstäver, varav två av dessa är o:n). Resultatet av detta blir alltså "kontroooooooll" Om ett ord inte innehåller ett o så skall inga extra o:n sättas in.

Vi utgår ifrån att man matar in orden med små bokstäver.

Körexempel 1:

```
Mata in två ord: bosse boxare
Dina ord var booooosse booooooxare.
```

Körexempel 2:

```
Mata in två ord: till    godo
Dina ord var till godooo.
```

Körexempel 3:

```
Mata in två ord: lo
       ok
Dina ord var loo ook.
```

Körexempel 4:

```
Mata in två ord: nononononono          yuhuu
Dina ord var nononononoooooo yuhuu.
```

KRAV: Tänk på att göra programmet generellt. Se upp för kopiering av kod...

2017-04-19.kl.14-19

Uppgift 2 - Dödens bro [1/2p]

Den som vill korsa dödens bro kommer att få tre frågor ställda till sig av broväktaren. Endast om alla tre frågor svaras korrekt får man korsa över till andra sidan. I annat fall kastas man ned i avgrunden av evig vånda.

Skriv ett program som ställer tre frågor till den som vill korsa dödens bro. Frågorna ligger på filen QUESTIONS.TXT. Man skall alltid få tre unika frågor. D.v.s. samma fråga skall inte dyka upp mer än en gång under samma körning. Du får lagra alla frågor i en vector om du vill.

Körexempel 1:

```
Stop! Who would cross the Bridge of Death must answer me these questions
three, ere the other side they see.
1. What... is your name?
2. What... is your quest?
3. What... is your favorite color?
```

Körexempel 2:

```
Stop! Who would cross the Bridge of Death must answer me these questions
three, ere the other side they see.
1. What... is your quest?
2. What... is the captial of Assyria?
3. What... is your favorite color?
```

Körexempel 3:

```
Stop! Who would cross the Bridge of Death must answer me these questions
three, ere the other side they see.
1. What... is your quest?
2. What... is your name?
3. What... is the air-speed velocity of an unladen swallow?
```

TIPS: Eftersom det är slump inblandat så är det inte säkert att resultatet blir exakt som i körexemplen ovan, och det bör bli olika från körning till körning.

KRAV: Tänk på att lösa problemet generellt. Det skall inte vara svårt att utöka programmet så att det blir fyra frågor.

OBS: Som uppgiften är beskriven ovan är den värd 2 poäng. Du får göra följande förenkling, i sådant fall blir uppgiften värd 1 poäng. Ta bort frågorna om "name" och "quest" från QUESTIONS.TXT och låt dessa istället alltid komma ut som fråga 1 respektive 2. Du behöver sedan alltså bara slumpa fråga nummer 3 från filens kvarvarande frågor. Du behöver då inte ta hänsyn till att det skall vara lätt att utöka till fyra frågor.

Uppgift 3 - Heltal eller flyttal? [2p]

På filen test_num_type.cpp finns ett givet program som skall testa datatypen num_type. Du skall i denna uppgift lägga till kod i den filen. Du skall skapa själva datatypen, samt tre funktioner som hanterar den.

Datatypen num_type skall användas för att kunna representera ett heltal *eller* ett flyttal. Här vill man på något sätt få det bästa av två världar. Heltal är praktiska för att de är exakta, flyttal är kanske inte alltid det men erbjuder istället möjligheten att inte behöva representera hela tal. Din num_type behöver därför innehålla följande tre delar.

- Ett heltal
- Ett flyttal
- Något som talar om huruvida det är heltalet eller flyttalet som används för tillfället.

Du skall också göra funktionerna *get*, *put* och *add.*

Funktionen get skall ta en parameter av typen num_type och från tangentbordet läsa in data till parametern. Om användaren matar in ett tal utan decimaler (eller om decimaldelen är 0) så skall det sparas i num_type:ens heltalsdel. Om användaren matar in ett tal med decimaler (där decimalerna är skiljt från 0) så skall talet sparas i num_type:ens flyttalsdel.

Funktionen put skall ta en parameter av typen num_type och skriva ut talet på skärmen. Här gäller det att tänka till så att man skriver ut rätt del av num_type:en.

Funktionen add skall ta två parametrar av typen num_type och **returnera** resultatet som en ny num_type. Endast i det fall då två heltal adderas med varandra är resultatet fortfarande ett heltal. I alla andra fall blir resultatet ett flyttal.

Körexempel 1:

```
Mata in ett tal: 1
Mata in ett till tal: 3.4
Summan av dessa tal är flyttalet 4.4.
```

Körexempel 2:

```
Mata in ett tal: 7
Mata in ett till tal: 3
Summan av dessa tal är heltalet 10.
```

Körexempel 3:

```
Mata in ett tal: 1.0
Mata in ett till tal: 2.0
Summan av dessa tal är heltalet 3.
```

KRAV: Du får inte ändra i huvudprogrammet i test_num_type.
TIPS: Funktionerna ceil och floor i biblioteket cmath kan vara bra.

Uppgift 4 - Lyckad kompilering? [2p]

Skriv ett program som kollar huruvida ett annat program har gått att kompilera. Denna kontroll skall göras genom att leta efter filer (i den mapp man står) med specifika filändelser. Användaren matar in källkodsfilens namn (utan filändelse), sedan skall programmet mata ut alla tillhörande filer som hittats i mappen. För att veta vilka filändelser som är intressanta så skall ditt program använda filen ENDINGS.TXT, som t.ex. kan se ut på följande vis:

```
.out
.cpp
.h
.cc
.o
```

Den allra första filändelsen i filen motsvarar den exekverbara filen, d.v.s. om den finns så har programmet gått att kompilera (vi tänker oss här att man döper den körbara filen till *programmets_namn*.out för just c++). Om inga filer hittas alls så skall programmet meddela det istället.

Körexempel 1:

```
Mata in filens namn: emmas_program
Kompilering lyckades!
Funna filer:
emmas_program.out
emmas_program.o
emmas_program.cpp
```

Körexempel 2:

```
Mata in filens namn: magnus_kod
Kompilering har inte lyckats!
Funna filer:
magnus_kod.cpp
```

Körexempel 3:

```
Mata in filens namn: viktors_hax
Inga tillhörande filer funna!
```

KRAV: Filändelserna måste läsas ifrån ENDINGS.TXT. Inga av filändelserna får hårdkodas in i programmet. (Filen ser annorlunda ut för andra språk och kompilatorer).