TDP007 - Tenta

2023-08-15

Regler

- All kod efterrättas på denna tenta
- Vid toalettbesök ska vakt informeras.
- All form av kontakt mellan studenter under tentamens gång är strängt förbjuden.
- Böcker och anteckningssidor kan komma att granskas av assistent, vakt eller examinator under tentamens gång.
- Frågor om specifika uppgifter eller om tentamen i stort ska ställas via tentasystemets kommunikationsklient.
- Systemfrågor kan ställas till jourhavande genom att räcka upp handen
- Endast uppgifter inskickade före tentamenstidens slut rättas.

Hjälpmedel	En Rubybok (exempelvis the pickaxe) Ett A4-ark med egna anteckningar Tillgång till webresurser: ruby-docs, rubular och tidig version av kursboken

Information

Betygssättning vid tentamen

Tentamen består av ett antal uppgifter på varierande nivå. Uppgifter som uppfyller specifikationen samt följer god sed och konventioner krävs för maxpoäng på en uppgift. Avvikelser från ovanstående ger avdrag.

Tentan består av uppgifter (några indelade i deluppgifter) som totalt kan ge 50 poäng.

Ruby-docs

På tentan har du tillgång till referenssidorna på `https://ruby-doc.org/` (både core och std-lib finns tillgänglig) via webbläsaren.

Rubular

På tentan har du tillgång till sidan `https://rubular.com/`.

Givna filer

Eventuella givna filer finns länkade från kurssidan.

Avslutning

När du skickat in alla uppgifter och känner dig färdig kan du stänga tetaklienten och logga ut.

Uppgift 1 - Teori (7p)

Vissa uppgifter nedan kräver i princip att man löst vissa av de praktiska uppgifterna på tentan.

1a - Parser (2p)

I uppgift 5 har du skapat en parser som kan göra enkel aritmetik och klarar av att spara data i variabler. I aritmetiken saknas beräkningar med exponenter, beskriv hur du lägger till exponenter (upphöjt till) i parsern och hur du styr att associativiteten för dessa korrekt.

1b - DSL vs GPL (2p)

På vilka sätt skiljer sig ett domänspecifikt språk (DSL) från ett vanligt programmeringsspråk (eng. general purpose language, GPL).

1c - Lexer (1p)

Vad är en lexer?

1d - Parser (2p)

Vad är en Parser?

Uppgift 2 - Regexp (10p)

Foobar Inc. har en webbserver som är mycket viktig för företaget. På webbservern finns en loggfil som lagrar information om alla förfrågningar som har kommit in, alltså varje gång någon hämtar en fil. Raderna i loggfilen ser ut så här (detta är alltså en rad):

```
67.33.163.249 - - [16/Feb/2012:15:39:39 +0100] "GET /hdr/department/docum
ent.shtml HTTP/1.0" 200 11683 "-" "Mozilla/5.0 (compatible; Yahoo! Slurp;
http://help.yahoo.com/help/us/ysearch/slurp)"
```

Först kommer IP-adressen för den som skickade förfrågan. Därefter kommer två fält som inte används och som markeras med streck. Sedan visas tid och datum inom hakparenteser. Därefter kommer den egentliga förfrågan. Om det är en normal begäran om att bara ladda ner en fil börjar strängen med GET, följt av katalog och filnamn och sist versionen på protokollet som användes (vilket oftast är HTTP/1.0). Efter denna sträng följer två tal. Det första är statuskoden, som är 200 om allt var okej. Det andra är storleken på filen. Sist på raden är två strängar som anger varifrån användaren kom samt vilken webbläsare som används.

I filen `access_log` finns ett utsnitt av loggfilen. Din uppgift är att skriva ett program som läser in, analyserar loggfilen och presenterar vissa slutsatser. Foobar Inc. är just nu intresserade av att veta hur mycket av vissa typer av filer som användarna laddar ner. De funderar på att lägga dessa filer på en särskild server, om det visar sig att det blir mycket trafik. Just idag är de intresserade av PDF-filer och JPG-filer. Hur många sådana laddades ner under de fem minuter som loggfilen speglar och vad var den totala storleken? Exempel på hur det kan se ut:

```
read_log("access_log")
57 PDF-files (12985775 bytes)
88 JPG-files (4327304 bytes)
```

Tips: `IO.readlines(filename)` öppnar filen på sökvägen `filename` och returnerar en array med alla rader i filen. Sedan kan man söka efter sitt mönster i varje rad.
Tips: Tänk på att ditt regepx inte behöver vara överdrivet generellt, fokusera på att lösa det givna problemet.

Uppgift 3 - XML (11p)

The Recursive Shakespeare Company är en amatörteaterförening som sätter upp klassiska pjäser. Wilhelm Skakspjut är regissör och han har hittat en webbsida med manuskript till pjäser i XML-format. Nu behöver han ett program som kan hjälpa honom att räkna ut lite statistik om den pjäs han har valt, nämligen Hamlet. I den bifogade filen hamlet.xml finns hela manuset. Strukturen framgår av den DTD som finns i filen. Det finns också en förkortad version i example.xml som man kan använda för testning. (Det kan ta rätt lång tid att bearbeta hela pjäsen.)

Regissören vill ha en lista som talar om hur många repliker det är i varje scen (alltså antalet LINE). Det gör att han lättare kan planera repetitionerna. Denna funktion skulle t.ex. kunna bete sig så här:

```
>> scene_length("hamlet.xml")
ACT I/SCENE I (189)
ACT I/SCENE II (274)
ACT I/SCENE III (140)
ACT I/SCENE IV (101)
ACT I/SCENE V (209)
ACT II/SCENE I (131)
...
```

Exakt hur utskriften ser ut är inte intressant, så länge information om akt, scen och antal repliker finns med.

Uppgift 4 - XML (11p)

Regissören från uppgift 3 vill ha en lista över långa monologer. En monolog är ett stycke i pjäsen där en enda skådespelare har en massa repliker ensam, utan att någon annan avbryter. Det kan lätt bli ganska tråkigt om pjäsen har för långa monologer, för då tappar man dynamik. Därför behöver skådespelarna repetera dessa monologer extra mycket. William vill ha en lista över alla monologer med minst tjugo repliker i följd (alltså minst 20 st LINE i samma SPEECH). Han vill ha en funktion som gör ungefär så här:

```
>> find_monologues("hamlet.xml")
ACT I/SCENE I HORATIO (29)
ACT I/SCENE I HORATIO (27)
ACT I/SCENE II KING CLAUDIUS (39)
ACT I/SCENE II KING CLAUDIUS (31)
ACT I/SCENE II HAMLET (31)
ACT I/SCENE III LAERTES (35)
ACT I/SCENE III LORD POLONIUS (27)
ACT I/SCENE III LORD POLONIUS (21)
ACT I/SCENE IV HAMLET (26)
...
```

Exakt hur utskriften ser ut är inte intressant, så länge information om akt, scen, skådespelare och monologens totala antal repliker finns med. Det är för övrigt i akt 3, scen 1 som Hamlets berömda monolog Ätt vara eller inte varafinns.

Uppgift 5 - Parser (11p)

Varparser är början på ett enkelt programspråk som kan hantera variabler och enkel aritmetik. I den givna filen `rdparse.rb` finns Diceroller språket som vi arbetat med i kursen. Ditt jobb är att modifiera det språket så att det passar specifikationen för detta nya språk.

I `Varparser` så kan användaren antingen definera en variabel eller slå upp värdet på en redan definerad variabel. Nedan följer ett exempel på inmatning och resultatet av inmatningen:

```
a: 2
b: 3
c: a * (b + 4)
c
>> 14
```

Första raden skapar och sätter variabeln a till talet 2.
Andra raden skapar variabeln b och sätter den till 3.
Tredje raden skapar variabeln c och sätter den till `a*(b+4)`.
Fjärde raden slår upp värdet som sparats i variabeln c.
Femte raden är bara utskriften av värdet som lagrats i c.

För full poäng ska addition, subtraktion, division och multiplikation fungera med korrek associativitet och prioritet. Det ska även gå att styra prioritet med parenteser. Du kan anta att en variabel i detta språk är godtyckliga sekvenser av bokstäver. Om en variabel tilldelas ett värde så följs variabelnamnet av ett kolon. Följs inte variabelnament av ett kolon så ska värdet som sparats i variablen slås upp. Det är odefinerat beteende om man slår upp värdet på en variabel som inte tidigare tilldelats ett värde.