TDP007 - Dugga 2

2024-02-27

Regler

- All kod efterrättas på denna dugga
- Inget sent insläpp tillåts under duggan
- Inga elektroniska hjälpmedel får medtas. Mobiltelefon ska vara avstängd och ligga i jacka eller väska.
- Inga ytterkläder eller väskor vid skrivplatsen.
- Student får lämna salen tidigast en timme efter start.
- 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 assistent i sal 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.

Duggan består av uppgifter (några indelade i deluppgifter) som totalt kan ge 25 poäng. Dessa poäng summeras ihop med poängen från den andra duggan.

Inloggning

Logga in på datorn i labbsalen med ditt liu-id och lösenord. Detta är samma inloggningsuppgifter som du använder i Lisam.

Skrivbordsmiljön

När du kommit in i tentasystemet har du en normal skrivbordsmiljö (Mate-session i Ubuntu). Efter en stund kommer även din kommunikationsklient att dyka upp automatiskt. Startmenyn är nedskalad till enbart det som examinator bedömt relevant. Andra program kan fortfarande finnas tillgängliga genom att starta dem från en terminal. Observera att en del program som använder nätverkstjänster inte fungerar normalt, eftersom nätverket inte är åtkomligt.

När du är inloggad är det viktigt att du har tentaklienten igång hela tiden. Om den inte dykt upp fem minuter efter inloggning och inte heller när du startar den manuellt från menyn (fisken) tar du kontakt med assistent eller vakt i sal.

Terminalkommandon

`ruby` används för att köra en ruby-fil.
`irb` används för att starta ruby i interaktivt läge.
`ri` används för att komma åt den inbyggda dokumentationen.

Ruby-docs

På tentan har du tillgång till referenssidorna på `https://ruby-doc.org/` (både core och stdlib finns tillgänglig) via webbläsaren Chrome. Starta `chromium-browser` i terminalen eller välj lämpligt alternativ från startmenyn. Observera att allt utom referenssidorna är avstängt. Om du inte kan komma åt en sida du tycker hör till referenssidorna (som kanske blockerats av misstag) kan du skicka ett meddelande via tentaklienten. Tag hjälp av assistent i sal om det inte fungerar.

Rubular

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

Givna filer

Eventuella givna filer finns i katalogen `given_files`. Denna underkatalog är skrivskyddad, så det är ingen risk du råkar ändra på dessa filer. Skrivskyddet gör dock att du måste kopiera in de givna filer du vill använda till tentakontots hemkatalog. Hur du listar och kopierar filer ska

du kunna. Hemkatalogen står du i från början, och du kommer alltid tillbaka till den genom att bara exekvera kommandot `cd` i terminalen.

Avslutning

När du skickat in alla uppgifter och känner dig färdig kan du logga ut och lämna salen. Antalet poäng meddelas i efterhand via webreg.

Avsluta alla öppna program och logga ut ur datorn. Lämna inte datorn innan du ser att du är utloggad.

Uppgift 1 - Teori (6p)

1. (2p) (Det är rimligt att besvara denna fråga efter du löst Uppgift 2) I uppgift 2 har du modifierat en parser som hanterar en begränsad och udda aritmetik. Förklara vilka tillägg du behöver göra för att körningen av parsern ska generera ett abstrakt syntaxträd (AST) istället för att bara räkna fram resultatet.

2. (2p) (Det är rimligt att besvara denna fråga efter du löst Uppgift 4) I uppgift 4 har du modifierat en parser som hanterar aritmetik i form av en postfix-räknare/stackmaskin. Ge ett förslag på hur denna parser kan byggas ut för att också hantera selektion (if-satser). Du behöver inte skriva en faktisk lösning på problemet, men du behöver förklara vilka delar som skall läggas till och hur prioriteten ska se ut.

3. (2p) I den givna filen `amb_test.rb` finns problemlösaren vi arbetat med i föreläsningsserien. Hur använder denna problemlösare sig av backtracking för att lösa problem.

Operator	Prioritet	Beskrivning
mod	1	Modulo
add	1	Addition
exp	2	Potens (upphöjt till)
dev	3	Division

Uppgift 2 - Parser (6p)

I denna uppgift ska du använda parsern i den givna filen rdparse.rb. Dicerollerspråket är lite modifierat för att inte ta inmatning via kommandoraden (för att göra din testning snabbare). Det enda som är ändrat är medlemsfunktionen `roll` som nu tar emot en sträng istället för att att låta användaren göra inmatning. Du måste modifiera det existerande diceroller-språket eller skapa ett nytt. Du ska skapa ett verktyg för att utvärdera aritmetiska uttryck. Dessa aritmetiska uttryck skall utvärderas på ett sätt som är oväntat för en ovetande slutanvändare. I språket skall operatorerna i tabellen finnas och fungera för godtyckliga sammansatta uttryck, modulo och addition har alltså delad högst prioritet i detta språk och division har lägst prioritet.

Det finns inte heller några parenteser i detta språk för att göra saker extra förvirrande. Alla uttryck i detta språk är högerassociativa, exempelvis kommer uttrycket **2 dev 8 dev 4** kommer utvärderas i ordningen **2/(8/4)**.

Körexempel:

```
> ap.parse "4 mod 3"
[aritparser] => 1

> ap.parse "4 mod 1 mod 3"
[aritparser] => 0

> ap.parse "2 dev 8 dev 4"
[aritparser] => 1
```

Tips: Körexemplet är uppenbart inte heltäckande, resonera dig fram till rätt funktionalitet eller testa relevanta kantfall.

Uppgift 3 - DSL (7p)

I filen `uppgift3.rb` finns den givna koden för en dsl-läsare som du känner igen från föreläsningsserien samt koden för constraint-nätverk som hanterar logiska grindar från föreläsningsserien. I filen `dsl.rb` finns ett domänspecifikt språk som bygger constraint nätverk. Ditt jobb är att modifiera den givna dsl-läsaren så att den hanterar det domänspecifika språket, det domänspecifika språket har följande konstruktioner:

- **create NAME** skapar en `Wire` med namnet NAME och sparar denna i en lämplig behållare. En ny Wire skapas alltid med default värdet (`false`).
- **agate WIRE_A WIRE_B WIRE_C** skapar en `AndGate` med anslutningarna (a, b och c) som skickas med i anropet
- **ogate WIRE_A WIRE_B WIRE_C** skapar en `OrGate` med anslutningarna (a, b och c) som skickas med i anropet
- **ngate WIRE_A WIRE_B** skapar en `NotGate` med anslutningarna (a och b) som skickas med i anropet
- **set WIRE BOOLEAN** sätter värdet på en wire till det angivna sanningsvärdet

Körexempel:

```
D, [2024-02-26T12:35:21.463560 699834] DEBUG -- : c = false
D, [2024-02-26T12:35:21.663872 699834] DEBUG -- : e = false
D, [2024-02-26T12:35:21.864211 699834] DEBUG -- : f = true
D, [2024-02-26T12:35:21.864288 699834] DEBUG -- : a = true
D, [2024-02-26T12:35:22.064561 699834] DEBUG -- : c = false
D, [2024-02-26T12:35:22.264870 699834] DEBUG -- : e = false
D, [2024-02-26T12:35:22.465174 699834] DEBUG -- : f = true
D, [2024-02-26T12:35:22.465257 699834] DEBUG -- : b = true
D, [2024-02-26T12:35:22.665512 699834] DEBUG -- : c = true
D, [2024-02-26T12:35:22.865840 699834] DEBUG -- : e = true
D, [2024-02-26T12:35:23.066237 699834] DEBUG -- : f = false
```

Krav: Godtyckliga namn på Wires skall tillåtas genom användning av `method_missing`, resterande funktioner som krävs (create, agate, ogate, ngate och set) kan läggas till som vanliga medlemsfunktioner.

Tips: Börja med att lösa läsningen av en del av programmet i `dsl.rb` till en början.

Uppgift 4 - Variabler (6p)

I filen `stackparse.rb` finns en stackmaskin som liknar den vi stött på i föreläsningsserien. Denna fungerar i sin helhet som den ser ut, men vi ska bygga ut den för hantering variabler på ett särskilt sätt. Stackparsern ska förutom nuvarande funktionalitet kunna spara undan tal i variabler och slå upp variabler. Variabler kan skapas på 2 olika sätt:

- **NAME=INTEGER** Exempelvis kan man skriva `a=4` för att tilldela variabeln a värdet 4. Detta uttryck interagerar inte med värdena som finns lagrade i stacken
- **NAME=_** Exempelvis kan man skriva `b=_`, värdet som ligger högst upp på stacken tas då bort och sparas i variabeln b.

För att göra beräkningar med variabler så skriver man helt enkelt namnet. När man tolkar en token tar man värdet som är sparat i variabeln och lägger till det på stacken. Nedan följer ett steg för steg exempel för vad som bör hända när uttrycket `a=4 3 a + b=_ b PRINT` tolkas:

1. Variabeln a sätts till 4. Stacken är tom.
2. Värdet 3 läggs till på stacken. Stacken: 3
3. Värdet som lagras i a läggs till på stacken. Stacken: 4, 3
4. Två värden poppas från stacken och summan läggs till på stacken. Stacken: 7
5. Ett värde poppas från stacken och tilldelas till variabeln b. Stacken är tom
6. Värdet som lagras i b läggs till på stacken. Stacken: 7
7. Ett värde poppas från stacken och skrivs ut. Stacken är tom

Körexempel: Resultatet av alla 3 körexempel som finns i den givna filen ska vara 27.