Labb S2: Sköldpaddegrafik

Problem-ID på Kattis: kth.progp.s2

I denna labb ska du implementera en parser för ett enkelt programmeringsspråk för grafik, baserat på det klassiska programmeringsspråket Logo (som du inte behöver känna till sedan tidigare eller ens när du är klar med den här labben). Du får använda vilket programmeringsspråk du vill bland de som finns på Kattis - utom Javascript/Ecmascript/Nodejs. När du väljer språk, tänk på att uppgiften kan blir signifikant mycket svårare i ett ovanligt programmeringsspråk. Det gäller både att lösa uppgiften själv samt att kunna få hjälp från lärarlaget. Kolla statistik på Kattis för att se hur många som har löst labben i ditt språk. Ett stort undantag från föregående är dock språket Haskell. Haskell gör den här labben förvånansvärt enkel att implementera eftersom språket är som gjort för att konstruera lexers, parsers och kompilatorer. Många av de stora svårigheterna med att få labben rätt undviks med Haskell. Du får inte använda inbyggda bibliotek/verktyg vars syfte är att konstruera parsers (t.ex. DCG i Prolog) utan ska implementera denna "från scratch" med rekursiv medåkning. Däremot får du gärna använda inbyggda bibliotek/verktyg för reguljära uttryck för din lexikala analys, om du vill (även om språket som ska parsas är såpass enkelt att det inte finns något egentligt behov av det). Du får gärna utgå ifrån Java-kodexemplen för rekursiv medåkning från föreläsningarna om du vill. Denna labb testas på Kattis och Kattis kan i vissa lägen ge viss feedback genom en ledtråd om hur testfallet som stoppade din inlämning såg ut. Observera dock att denna feedback varken är eller ska vara en ersättning för egen testning utan det är mycket möjligt att ett testfall som skrivits i ett syfte hittar en helt annan typ av problem med din kod. Det är inte Kattis eller lärarlaget som i det läget ska hjälpa dig med felsökning utan var mycket noggrann med denna labb. Följ riktlinjerna i boken, dokumentera din kod väl och gör egen testning så kan du steg för steg ta dig närmare en lösning.

Observera att i denna labb är det du och en eventuell labbpartners uppgift att välja vilka datatyper som ska lagra datan, att namnge och utforma eventuella hjälpfunktioner, att komma på lösningsmetoder och till sist implementera den. Det är ett brott mot hederskodexen att be personer utanför labblaget med hjälp om detta, eller att använda artificiella intelligenser. Listan med förbjudna artificiella intelligenser inkluderar men är inte begränsad till ChatGPT och GitHub Copilot.

Sköldpaddegrafik

Till vår hjälp har vi en sköldpadda (låt oss kalla den Leona) med en penna. Vi kan instruera Leona att gå till olika platser och linjer ritas då längs vägen Leona går. Instruktionerna till Leona ges som "program" i Leona-språket. Språket har följande instruktionsuppsättning:

`FORW` d	Leona går framåt d punkter (för ett *positivt* heltal d).
`BACK` d	Leona går bakåt d punkter (för ett *positivt* heltal d).
`LEFT` θ	Leona svänger vänster θ grader (för ett *positivt* heltal θ), utan att flytta sig från sin nuvarande position.
`RIGHT` θ	Leona svänger höger θ grader (för ett *positivt* heltal θ), utan att flytta sig från sin nuvarande position.
`DOWN`	Leona sänker ned pennan så att den lämnar spår efter sig när Leona rör sig
`UP`	Leona höjer pennan så att inget spår lämnas när Leona rör sig
`COLOR` c	byter färg på pennan till färgen c. Färg specas på hex-format, t.ex. `#FFA500` för orange (om du är osäker på vad detta innebär så är din favoritsökmotor din vän, som vanligt).
`REP` r `<REPS>`	Leona upprepar `<REPS>` r gånger (för ett *positivt* heltal r). `<REPS>` är en sekvens av en eller flera instruktioner, omgivna av citationstecken (‘"’). Om sekvensen bara består av en enda instruktion är citationstecknen valfria.

Språket är case insensitive – i både kommando-namn och beskrivning av färger kan små och stora bokstäver blandas. Kommandon i språket avslutas med punkt (‘.’), med undantag för `REP`-kommandon,

efter dessa har man inga punkter (däremot ska varje kommando i `REP`-sekvensen avslutas med punkt). Kommentarer kan skrivas i språket med procenttecken ('%'), allt som står efter ett procenttecken på en rad anses vara en kommentar. All whitespace (mellanslag, tabbar och nyradstecken) är ekvivalent förutom i kommentarer (där nyrad betyder "slut på kommentar"). Det måste finnas whitespace mellan ett kommando och dess parameter (t.ex. mellan `RIGHT` och θ), samt mellan r-argumentet till `REP` och `<REPS>`-delen. I övrigt är all whitespace godtycklig.

Leona startar på positionen $(0, 0)$ och är vänd i riktning mot punkten $(1, 0)$. Pennan är initialt blå (`#0000FF`) och i upphöjt läge.

Notera att även om alla indataparametrar är heltal så kan Leona hamna på koordinater som inte är heltal. Om vi t.ex. från startläget utför `LEFT 30. FORW 2.` kommer Leona att befinna sig på positionen $(\sqrt{3}, 1) \approx (1.732, 1)$.

Uppgifter

Det övergripande målet med uppgiften är att skriva en *översättare* för Leona-språket, som översätter ett program på Leona-språket till en lista med linjesegment givna i kartesiska koordinater. För varje instruktion där Leona går framåt eller bakåt och pennan är nedsänkt ska du alltså konstruera ett linjesegment från punkten (x_1, y_1) där Leona startar till punkten (x_2, y_2) där Leona stannar.

För att göra detta ska du utföra följande uppgifter:

1. Konstruera en formell grammatik för Leona-språket. För att hjälpa till på traven med detta ger vi ett förslag på hur indata borde styckas upp i tokens nedan. Du behöver inte följa detta förslag om du inte vill men om du har problem att få din lösning korrekt rekommenderas att du följer denna vägledning. Grammatiken behöver uttryckas på BNF-form eller liknande för godkänt. Senare ska en rekursiv medåknings-parser skrivas enligt grammatiken, så försök se till att grammatiken är lämpad för detta (annars kommer den antagligen behöva modifieras).

2. Rita ett korrekt och läsligt parseträd för testfall 12 i din grammatik. Lägg trädet på Git och var beredd på att förklara det under redovisningen av labben.

3. Som första steg i en parser för din grammatik, skriv en *lexikal analysator* som delar upp indatafilen i tokens.

4. Skriv en *parser* för Leona-språket med rekursiv medåkning. Parsern ska ta sekvensen av tokens som produceras av den lexikala analysatorn, och producera ett syntaxträd.

5. Skriv kod för att *exekvera* det givna programmet genom att översätta det syntax-träd som produceras av parsern till en lista med linjesegment.

6. Slå ihop lexikal analys, parsning, och exekvering till ett fullständigt program som läser ett Leona-program och konstruerar linjesegmenten. Se nedan för detaljerad information om hur indata ska läsas och utdata skrivas.

Observera att dessa del-uppgifter ska ses som krav. Du **ska** konstruera en grammatik för språket, du **ska** skriva en parser som använder rekursiv medåkning, och du **ska** separera de olika stegen (lexikal analys, parsning, exekvering).

Vägledning

Trigonometri: För att göra översättningen behöver du kunna beräkna vilken position Leona befinner sig på. Låt oss påminna om följande grundläggande trigonometriska faktum: om Leona befinner sig på koordinaterna (x, y), är vänd i riktning v (antal grader moturs från rakt högerut), och går d punkter framåt, så kommer Leonas nya position att vara $(x + d\cos(\pi v/180), y + d\sin(\pi v/180))$

Lexikal analys: Ett förslag på token-typer att använda för tokenisering (lexikal analys) av indata är att använda följande 13 token-typer:

- FORW, BACK, LEFT, RIGHT, DOWN, UP, COLOR, REP: tokens som representerar respektive kommando.
- PERIOD, QUOTE: token som representerar punkt respektive citat-tecken.
- DECIMAL: token som representerar ett positivt heltal.
- HEX: token som representerar en hex-färg.
- ERROR: allt annat innehåll – token som representerar ett syntax-fel på lexikal nivå.

Det kan vara lockande att även ha med en token-typ för whitespace (eftersom detta måste finnas på vissa ställen), men detta tenderar att leda till obegripliga buggar. Istället kan tokens för de kommandon som har argument definieras som "kommando-namnet följt av ett whitespace". En sträng som "`FORW 12`" skulle då ge token-sekvensen FORW, NUMBER, medan strängen "`FORW12`" skulle ge ett ERROR-token. Kravet på whitespace mellan r-argumentet till `REP` och `<REPS>`-delen kan hanteras på ett liknande sätt men är lite mer komplicerat. Eftersom vi i den lexikala analysen inte ser skillnad på ett tal som är ett argument till t.ex. `FORW` och ett tal som är ett argument till `REP` så måste vi tillåta att ett tal följas av andra saker än whitespace, men vissa kombinationer som t.ex. "`42"`" och "`42F`" borde resultera i ett ERROR-token.

Rekursiv medåkning: Kom ihåg att en rekursiv medåkningsparser ska ha en funktion för varje icke-slutsymbol i din grammatik. Dessa funktioner ska inte ta några argument, och returnera ett parseträd. Om du börjar skriva funktioner som tar in olika argument som håller reda på någonting om vad som hänt hittills (t.ex. en boolean-flagga som anger huruvida vi befinner oss inne i en REP-sats eller inte) så har du gjort fel, och kommer behöva göra om din lösning! Detsamma gäller om du istället för funktions-argument har lite globala variabler eller klass-variabler som håller reda på något slags tillstånd. Det *enda* tillstånd (state) som får finnas i parsern är ett Lexer-objekt som håller reda på vilket indata-token vi för närvarande befinner oss på.

Gör delarna i rätt ordning: Det kan vara lockande att angripa den här uppgiften genom att helt enkelt börja koda. Tips: gör inte det! Om du börjar i rätt ände, med att konstruera en grammatik och göra uppgiften "by the book", löper du mycket mindre risk att fastna på någon av de *många* detaljer och knepigheter som finns i uppgiften, vilket kommer bespara dig tid i det långa loppet.

Indata

Indata består av ett Leona-program och ges på standard input (`System.in` i Java – vid behov, se Kattis-hjälpen för information om vad detta betyder).

Du kan anta att *om* det givna programmet är syntaktiskt korrekt så kommer alla alla tal som är obegränsade i språkdefinitionen (avståndsparametrar d och repetitionsparametrar r) vara högst 10^5, och att det totala antalet instruktioner som utförs när programmet körs kommer vara högst $2 \cdot 10^5$ (dessa är alltså garantier på indata, inget du behöver kontrollera).

Indatafilen är högst 1 MB stor och teckenkodningen är utf-8.

Utdata

- Om det givna Leona-programmet är syntaktiskt felaktigt ska följande meddelande skrivas ut, där r är den rad på vilken (första) syntaxfelet finns:

  `Syntaxfel på rad` r

Se förtydliganden i exempel-fallen nedan om vilken rad som anses vara raden för första syntaxfelet.

- Annars, om det givna programmet är syntaktiskt korrekt, ska en lista med linjesegment som ritas av programmet skrivas ut. Segmenten ska skrivas ut i samma ordning som de ritas av Leona, och varje segment skrivs ut på en ny rad, på följande format:

  $c\ x_1\ y_1\ x_2\ y_2$

  Här är c färgen linjesegmentet har (i hex-format precis som i språket), (x_1, y_1) är startpunkten för linjesegmentet (den punkt där Leona började när segmentet ritades) och (x_2, y_2) är slutpunkten för linjesegmentet (den punkt där Leona slutade). Koordinaterna ska vara korrekta upp till en noggrannhet på 10^{-3} (om `double`-variabler används och svaret skrivs ut med 4 eller fler decimaler ska det inte bli avrundningsfel).

- Teckenkodningen på utdatan ska vara utf-8.

Sample Input 1

```
% Det här är en kommentar
% Nu ritar vi en kvadrat
DOWN.
FORW 1. LEFT 90.
FORW 1. LEFT 90.
FORW 1. LEFT 90.
FORW 1. LEFT 90.
```

Sample Output 1

```
#0000FF 0.0000 0.0000 1.0000 0.0000
#0000FF 1.0000 0.0000 1.0000 1.0000
#0000FF 1.0000 1.0000 0.0000 1.0000
#0000FF 0.0000 1.0000 0.0000 0.0000
```

Sample Input 2

```
% Space runt punkt valfritt.
DOWN  . UP.DOWN.  DOWN.
% Rader kan vara tomma

% radbrytning/space/tabb för
% att göra koden mer läslig.
REP 3 "COLOR #FF0000.
       FORW 1. LEFT 10.
       COLOR #000000.
       FORW 2. LEFT 20."
% Eller oläslig
          COLOR
% färgval på gång
  #111111.
REP 1 BACK 1.
```

Sample Output 2

```
#FF0000 0.0000 0.0000 1.0000 0.0000
#000000 1.0000 0.0000 2.9696 0.3473
#FF0000 2.9696 0.3473 3.8356 0.8473
#000000 3.8356 0.8473 5.3677 2.1329
#FF0000 5.3677 2.1329 5.8677 2.9989
#000000 5.8677 2.9989 6.5518 4.8783
#111111 6.5518 4.8783 6.5518 3.8783
```

Sample Input 3

```
% Syntaxfel: felaktig färgsyntax
COLOR 05AB34.
FORW 1.
```

Sample Output 3

```
Syntaxfel på rad 2
```

Sample Input 4

```
% Oavslutad loop
REP 5 "DOWN. FORW 1. LEFT 10.
```

Sample Output 4

```
Syntaxfel på rad 2
```

Sample Input 5

```
% Syntaxfel: ej heltal
FORW 2,3.
```

Sample Output 5

```
Syntaxfel på rad 2
```

Sample Input 6

```
%&(CDH*(
FORW
#123456.
&C(*N&(*#NRC
```

Sample Output 6

```
Syntaxfel på rad 3
```

Sample Input 7

```
% Måste vara whitespace mellan
%   kommando och parameter
DOWN. COLOR#000000.
```

Sample Output 7

```
Syntaxfel på rad 3
```

Sample Input 8

```
% Syntaxfel: saknas punkt.
DOWN
% Om filen tar slut mitt i ett kommando
% så anses felet ligga på sista raden
   % i filen där det förekom någon kod
```

Sample Output 8

```
Syntaxfel på rad 2
```

Sample Input 9

```
% Måste vara space mellan argument
REP  5"FORW 1."
% Detta inte OK heller
REP   5FORW 1.
```

Sample Output 9

```
Syntaxfel på rad 2
```

Sample Input 10

```
% Ta 8 steg framåt
REP 2 REP 4 FORW 1.
REP% Repetition på gång
2% Två gånger
"%Snart kommer kommandon
DOWN% Kommentera mera
.% Avsluta down-kommando
FORW 1
LEFT 1. % Oj, glömde punkt efter FORW-kommando
"
```

Sample Output 10

```
Syntaxfel på rad 9
```

Sample Input 11

```
% Nästlad loop 1
REP 2 "UP. FORW 10. DOWN. REP 3 "LEFT 120. FORW 1.""
% Nästlad loop 2
REP 3 "REP 2 "RIGHT 2. FORW 1."
       COLOR #FF0000. FORW 10. COLOR #0000FF."
% COLOR #000000. % Bortkommenterat färgbyte
BACK 10.
% Upper/lower case ignoreras
% Detta gäller även hex-tecknen A-F i färgerna i utdata,
% det spelar ingen roll om du använder stora eller små
% bokstäver eller en blandning.
color #AbcdEF. left 70. foRW 10.
```

Sample Output 11

```
#0000FF 10.0000 0.0000 9.5000 0.8660
#0000FF 9.5000 0.8660 9.0000 0.0000
#0000FF 9.0000 0.0000 10.0000 0.0000
#0000FF 20.0000 0.0000 19.5000 0.8660
#0000FF 19.5000 0.8660 19.0000 0.0000
#0000FF 19.0000 0.0000 20.0000 0.0000
#0000FF 20.0000 0.0000 20.9994 -0.0349
#0000FF 20.9994 -0.0349 21.9970 -0.1047
#FF0000 21.9970 -0.1047 31.9726 -0.8022
#0000FF 31.9726 -0.8022 32.9671 -0.9067
#0000FF 32.9671 -0.9067 33.9574 -1.0459
#FF0000 33.9574 -1.0459 43.8601 -2.4377
#0000FF 43.8601 -2.4377 44.8449 -2.6113
#0000FF 44.8449 -2.6113 45.8230 -2.8192
#FF0000 45.8230 -2.8192 55.6045 -4.8983
#0000FF 55.6045 -4.8983 45.8230 -2.8192
#ABCDEF 45.8230 -2.8192 51.1222 5.6613
```

Sample Input 12

```
DOWN .
% OBS! Denna fil har ett
% tabb-tecken ('\t') i första
% REP-satsen.  Om du kör den
% genom att göra copy-paste
% kommer tabb-tecknet inte
% testas korrekt.
REP     2 REP % raaadbryyyt
1 "REP 2 REP 2 "FORW 1."
 LEFT 45."
```

Sample Output 12

```
#0000FF 0.0000 0.0000 1.0000 0.0000
#0000FF 1.0000 0.0000 2.0000 0.0000
#0000FF 2.0000 0.0000 3.0000 0.0000
#0000FF 3.0000 0.0000 4.0000 0.0000
#0000FF 4.0000 0.0000 4.7071 0.7071
#0000FF 4.7071 0.7071 5.4142 1.4142
#0000FF 5.4142 1.4142 6.1213 2.1213
#0000FF 6.1213 2.1213 6.8284 2.8284
```