Linköpings universitet

TDP019

Projekt: Datorspråk

CodEng

Author
Eric Jönsson
Vidar Siwe

Examinator
Jonas Wallgren

May 14, 2019

Contents

1 Inledning

TDP019 Projekt: datorspråk är ett av de projektarbeten som görs av studenter som läser första året på programmet Innovativ programmering vid Linköpings universitet. Projektet utförs under hela den andra terminen och redovisas i maj. Projektet går ut på att ett eget programmeringsspråk ska konstrueras och implementeras.
Det konstruerade språket programmeras i programmeringsspråket Ruby och till hjälp finns en redan programmerad parser kallad rdparse. Det programmeringsspråk som vi har skapat kallas för "CodEng". Språket går ut på att i stor del kunna skrivas som "normal" text på engelska.

2 Användarhandledning

"CodEng" som ord kommer ifrån en sammansättning av delarna "cod" från "code" och "eng" från "english". Språket fick detta namn för att det ska vara intuitivt angående vad språkets grundidé är. Språket är ett imperativt programmeringsspåk och går ut på att det i stor utsträckning ska fungera att skriva ut programmeringskoden som engelsk text. För att kunna skriva språket som ren text behövs förkunskaper i programmering, eftersom de ord som är godkända att använda är baserade på hur "normal" programmering ser ut. Språket riktar sig mot de personer som vill att koden de skriver ser mer ut som normal text än kod i sina programmeringsfiler.

2.1 Installation

CodEng kräver linux med senaste versionen av programmeringsspråket Ruby. I skrivande stund är det version 2.6.3. Ladda ner "CodEng" till valfri plats och öppna sedan terminalen i undermappen **bin**. Skriv sedan följande kommando i terminalen för att slutföra installationen:
`echo "PATH=$PATH:$(pwd)" >> <konfigurationsfil>`
Ersätt <konfigurationsfil> med sökvägen till skalets konfigurationsfil. För bash är detta `~/.bashrc`. Starta om terminalen eller ladda konfigurationsfilen igen för att slutföra installationen.

2.2 Körning

CodEng kan köras på två sätt, interaktivt eller med en fil. Skriv följande kommando i terminalen för att köra en fil:
`CodEng <sökväg-till-fil>`
Ifall ingen fil anges körs CodEng i interaktivt läge. I interaktivt läge får användaren själv mata in kommandon på en rad som sedan parsern tolkar. Ett returvärde skrivs sedan ut i terminalen och användaren får möjlighet att mata in igen.

2.3 Datatyper

Det finns fem stycken olika datatyper i CodEng. Datatyperna är CEBool, CEFloat, CEInteger, CEString och CEVariable.
CEBool är den datatyp i språket som hanterar "true"- och "false"-värden. Detta betyder att om ett sant eller falskt värde skapas i språket kommer det värdet att sparas som ett objekt av typen CEBool. Denna sortens sparning gäller generellt för alla de olika datatyperna men värdena som sparas i dem skiljer sig från varandra.
I CEFloat sparas decimaltal som skrivs i koden eller som beräknas av programmets körning. CEInteger sparar på samma sätt som CEFloat tal. Skillnaden är att CEInteger endast sparar heltal. CEString är den datatyp som sparar textsträngar i språket till exempel sparas "3" som en CEString men 3 sparas som en CEInteger. Skillnaden för att det ska sparas som en textsträng är att det som ska sparas skrivs inom citattecken. I datatypen CEVariable sparas de variabler som sätt i koden.

2.4 Operationer

I språket hanteras tre stycken olika typer av operationer. Aritmetiska operationer, relationsoperationer och logiska operationer.
I de aritmetiska operationerna används operatorerna: +, -, *, /, ** och () för att strukturera upp de aritmetiska uttryck som kan tänkas utföras med språket. I språket motsvaras också operatorerna av engelska textversioner. Några exempel på aritmetiska uttryck och hur de kan skrivas i CodEng på olika sätt visas i figur 1.

```
3 + 4
=> 7
5 plus 4
=> 9
6 - 4
=> 2
92 minus 12
=> 80
5 * 5
=> 25
3 times 7
=> 21
24 / 3
=> 8
30 over 5
=> 6
4 ** 2
=> 16
2 to the power of 8
=> 256
-30 plus 15
=> -15
3 * -6
=> -18
(5 + 5) * 2
=> 20
((10 / 5) ** 4) + ((5 + 7) * 3)
=> 52
```

Figure 1: Aritmetiska uttryck och hur de kan skrivas i CodEng.

Den andra typen av operationer som har implementerats i programmeringsspråket var relationsoperationer. Operatorerna för relationsoperationer är: $<, <=, >$ $, >=$, $==$, $!=$ och de används för att hitta hur olika värden relaterar i förhållande till varandra. Exempelvis används de ofta inom iterationer för att se om loop är färdig eller inte. I CodEng finns det speciella uttryck som motsvarar dessa operatorer. Olika typer av relationsuttryck visas med exempel i figur 2.

```
2 < 3
=> true
4 less than 8
=> true
4 <= 4
=> true
5 less than or equal to 4
=> false
2 > 3
=> false
1 greater than 4
=> false
4 > 2
=> true
5 >= 6
=> false
5 >= 5
=> true
5 >= 4
=> true
5 greater than or equal to 9
=> false
3 == 3
=> true
7 equal to 6
=> false
3 != 3
=> false
5 not equal to 4
=> true
```

Figure 2: Relationsoperationer och hur de kan skrivas i CodEng.

Logiska operatorer används i logiska uttryck för att se om värdet blir sant eller falskt. Operatorerna är: &&, || och ! som motsvaras av “and”, “or” och “not” i CodEng. Exempel på hur logiska uttryck kan användas i språket syns i figur 3. Vid användning av logiska operatorer på siffror räknas alla siffror som inte är 0 som sanna värden, endast 0 räknas som falskt.

```
true && true
=> true
true && false
=> false
false and true
=> false
false and false
=> false
true || true
=> true
true || false
=> true
false or true
=> true
false or false
=> false
!true
=> false
not false
=> true
0.1 and true
=> true
0.0 and true
=> false
1 and true
=> true
0 and true
=> false
```

Figure 3: Logiska uttryck och hur de kan skrivas i CodEng.

2.5 Funktioner

Funktioner i CodEng är kodstycken som har tilldelats variabelnamn. Funktionerna kan också bearbeta argument som skickats in i dem. Argumenten i funktioner till exempel vara någon av de datatyper som nämndes i under del 2.3. Funktioner kan anropas med ett funktionsanrop. Vid ett anrop utförs det kodstycke som tillhör just den funktionen på den plats i koden där anropet skrevs. I figur 4 visas en funktion skriven i textversionen av CodEng. När funktionen i figuren körs beräknas upphöjt till 3 för den siffra som skickades in till funktionen som argument och skriver ut svaret på beräkningen. Den gör sedan samma sak igen för de nio följande heltalen genom en “while-loop”. Det senaste uttrycket som kördes innan funktionen avslut är det värde som funktionen skickar ut.

```
define foo with number do
loops is number plus 10
while number less than loops do
x is number to the power of 3
writeln(x)
number is number plus 1
stop stop

call foo with 1
```

Figure 4: En funktionsdeklaration och motsvarande anrop.

2.6 Iterationer

I CodEng finns endast en typ av iteration, “while”-loopar. “While”-loopen används för att köra ett kodstycke om och om igen tills oändligheten eller ett speciellt krav har uppfyllts. Själva loopen har sant respektive falskt läge. I det sanna läget körs kodstycket som står inne i loopen och i det falska läget stängs loopen av. I figur 5 visas ett exempel på en “while”-loop och hur de fungerar i språket. “While”-loopen körs på grund av att loop variabeln som satts till att vara “true” har placerats som loopens tillstånd.

```
i = 0
loop = true
while loop do
  if i < 7 then
    i = i + 1
  else
    writeln(i)
    loop = false
  stop
stop
```

Figure 5: En “while”-loop som visar på looping och val i loopar

2.7 Val

De val som har implementerats i språket är de klassiska “if”- och “else”-satserna, översatt till svenska blir det om- och annars-satser. Satserna fungerar genom att koden inom just den “if”- eller “else”-satsen endast körs om det uttryck som tillhör satsen är sant. “If”-delen kollas först och om den delen är sann hoppas “else”-delen över, och tvärtom. Detta illustreras i samma figur 5, där uttrycket för “if”-delen är sann för siffrorna 0-6 men falsk när siffran blir 7. I detta läge körs istället koden för “else”-delen och loop-variabeln sätts till falsk, vilket gör att hela loopen stannar.

2.8 Variabler och “Scope”

Variabler i CodEng består av bokstäver, siffror och understreck. Variabelnamn kan inte bestå av endast siffror. Variablerna sparas med sitt värde i ett “scope”. CodEng har dynamiskt “scope”. Det innebär att variablens värde alltid är det första den hittar i sitt nuvarande scope.
Ett exempel på detta kan ses i figur 6 till höger. Figuren innehåller 3 “scopes”, ett som omsluter hela programmet och sedan har båda funktionerna varsitt “scope”. När foo kallas i bar kommer den att skriva ut “number”. Först letar foo i sitt eget scope efter “number” men hittar ingen deklarerad variabel med det namnet. Eftersom den kallades i funktionen bar letar den nu i dess scope och hittar att den har värdet 20. Om det hade varit statiskt “scope” istället för dynamiskt så kommer foo att leta efter variabeln i “scopet” för hela programmet istället, då det är där funktionen är definerad, och värdet blir då 10.

```
number is 10

define foo
write(number)
stop

define bar
number is 20
call foo
stop

call bar
```

Figure 6: Exempel på hur “scope” fungerar i CodEng.