Róbótinn Karel

Guðmundur Björn Birkisson

Iðnaðarverkfræði-, vélaverkfræði- og tölvunarfræðideild
Háskóli Íslands
2013

RÓBÓTINN KAREL

Guðmundur Björn Birkisson

60 ECTS eininga ritgerð sem er hluti af
Magister Scientiarum gráðu í Hugbúnaðarverkfræði

Leiðbeinendur
Snorri Agnarsson
Hjálmtýr Hafsteinsson

Prófdómari
Freyja Hreinsdóttir

Iðnaðarverkfræði-, vélaverkfræði- og tölvunarfræðideild
Verkfræði- og náttúruvísindasvið
Háskóli Íslands
Reykjavík, Maí 2013

Róbótinn Karel
60 ECTS eininga ritgerð sem er hluti af M.Sc. gráðu í Hugbúnaðarverkfræði

Iðnaðarverkfræði-, vélaverkfræði- og tölvunarfræðideild
Verkfræði- og náttúruvísindasvið
Háskóli Íslands
Harðarhaga 6
107 Reykjavík

Sími: 525 4000

Skráningarupplýsingar:
Guðmundur Björn Birkisson, 2013, Róbótinn Karel, meistararitgerð,
Iðnaðarverkfræði-, vélaverkfræði- og tölvunarfræðideild, Háskóli Íslands.

Prentun: Háskólaprent
Reykjavík, Maí 2013

Abstract

In recent years, discussion about teaching younger students programming has increased. Many ideas have been tested in this field and there exists some variety of programs that take different approaches to this challenge. Karel the Robot is one of those ideas that first surfaced in the nineteen eighties. That system was designed for beginner courses in computer science. It was immensely popular and proved successful for what it was designed for. Now the system has been redesigned for Icelandic students, but is aimed at younger students. Karel the Robot provides a simple and fun environment to solve problems that support the student in learning programming. A new programming language has also been designed specifically to be as understandable and easy as possible, but also powerful enough to be able to solve a wide variety of problems. The system also provides a platform so that anyone can to create the problems to solve, which makes it ideal for teachers. A manual also comes with the system that fully describes each element of it and provides examples of problems and the solutions to them. Karel the Robot is free and the code for the system is open source.

Útdráttur

Áhugi og umræða hefur aukist um hvernig kenna skuli yngri nemendum forritun. Margar hugmyndir um betrumbætur á þessu sviði hafa komið fram og til eru mörg forrit sem taka mismunandi nálganir á kennsluna. Róbótinn Karel er ein slík hugmynd sem á rætur sínar að rekja til níunda áratugarins. Það kerfi var á sínum tíma að mestu beint að nemendum í grunnnámi í tölvunarfræði eða sambærilegu námi. Kerfið naut mikilla vinsælda og hefur reynst vel á þessum vettvangi. Núna er búið að endurgera þetta kerfi fyrir íslenska nema en er þó hugsað fyrir lægra menntunarstig en háskólanám. Róbótinn Karel býður upp á auðvelt og skemmtilegt umhverfi til þess að læra forritun með því að leysa verkefni. Nýtt forritunarmál hefur verið hannað með það í huga að hafa það eins skiljanlegt og auðið er. Það er þó nógu öflugt til þess að geta leyst stóra flóru af verkefnum. Kerfið býður líka upp á að hver sem er getur búið til verkefni til þess að leysa, sem gerir það tilvalið til kennslu. Að lokum fylgir handbók sem lýsir kerfinu og dæmi um verkefni og lausnir á þeim. Róbótinn Karel er ókeypis og kóðinn fyrir kerfið er opinn öllum.

Efnisyfirlit

Myndaskrá

Töfluskrá

Þakkir

Ég vill þakka leiðbeinanda mínum Snorra Agnarssyni fyrir góða leiðsögn og stuðning, bæði í gegn um námið og verkefnið sjálft. Ég vil líka þakka öllum sem hjálpuðu til við prufukeyrslur og annað sem tengdist verkefninu. Einnig þakka ég Skarphéðni Garðarssyni fyrir prófarkarlestur. Að lokum vil ég þakka fjölskyldunni minni fyrir allan þann stuðning sem hún hefur veitt mér í gegn um námið undanfarin ár.

1. Inngangur

Tölvur gegna stærra og stærra hlutverki í daglegu lífi fólks. Fjölmiðla og afþreyingu af flestu tagi er hægt að nálgast í gegnum tölvuna. Það sama gildir um þjónustu fyrirtækja á borð heimabanka, umsóknir og upplýsingamiðlun svo eitthvað sé nefnt. Sumar af þessum þjónustum, sem boðið er upp á, eru verr nýttar af notendum en aðrar. Sem dæmi má nefna RSS efnisstrauma sem fréttamiðlar nota til þess að birta fréttir á vefnum. Atvinnuveitendur hafa einnig notað RSS strauma til þess að auglýsa störf (Sigurðsson, 2012).

Þessi aukna tölvuvæðing þarfnast þess að notendur skilji og kunni að nota tölvur til þess að nýta sér þá þjónustu og hraða sem tölvur hafa upp á að bjóða. Það þarf að stuðla að því að fólk almennt, nálgist notkun á tölvunni með snjöllum hætti (e. smart users of technology) og þetta er m.a. ástæðan fyrir því að í Eistlandi er byrjað að kenna börnum í fyrsta bekk að forrita (Olson, 2012). Við þurfum að efla tölvufærni næstu kynslóðar, læra að vinna með tölvuna en ekki bara vinna á hana (Sölvadóttir, 2013). Af þessum ástæðum hefur fólk verið að stíga fyrstu skrefin hér á landi til þess að kenna börnum á grunnskólastigi forritun. Sérhæfð fyrirtæki hafa haldið námskeið handa börnum með hjálp tóla sem koma víðsvegar að.

Það eru skiptar skoðanir á því hvernig á að kenna forritun. Hægt er að deila um hvaða forritunarmál eða kerfi sé best í stakk búið til þess að hjálpa nemendum að læra forritun. Margir kennarar eru þó á þeirri skoðun að það sé æskilegast að einblína á lausn á verkefnum (e. problem solving) og koma því á framfæri að forritunarmálið sem notað er sé aðeins tól til þess að hjálpa við lausn á verkefnunum (Labuscagne, 2008). En hver er þá besta aðferðin við það að kenna nýliðum í forritun þennan hæfileika? Það hefur sýnt sig að þegar nemendum er kennt að leysa verkefni með hjálp leikja, eða herma, getur það hjálpað við skilning, aukið ánægju nemenda og ýtt undir áhuga (Liu et al., 2011).

2. Saga Karel

Árið 1981 gaf Richard E. Pattis út bókina „Karel The Robot: A Gentle Introduction to the Art of Programming". Þar kynnti hann til sögunnar róbótann Karel en nafnið fékk hann frá tékkneska höfundinum Karel Čapek sem stuðlaði að því, með skrifum sínum, að orðinu „robot" var bætt við enska tungumálið.

Hugmyndin var að kenna nemendum forritun án þess að þurfa að fara í þær flækjur sem fylgdu forritunarmálum á þessum tíma. Þetta gerði hann með því að hanna kerfi sem gerði nemendum kleift að forrita róbótann Karel, á auðveldan hátt, til þess að leysa ýmis verkefni. Karel naut þó nokkurra vinsælda og var notað í grunnáföngum í tölvunarfræði við marga skóla í Bandaríkjunum sem varð til þess að bók Pattis var seld í vel yfir 100.000 eintökum (Roberts, 2005).

2.1. Upprunalega kerfið

Róbótinn Karel býr í heimi sem er búinn til með ferhyrningum sem mynda net, líkt og skákborð. Karel hefur 5 grunn aðgerðir:

- move (Karel gengur einn reit áfram í áttina sem hann snýr í).
- turnleft (Karel snýr sér 90° til vinstri).
- putbeeper (Karel setur niður boðtæki þar sem hann stendur).
- pickbeeper (Karel tekur upp boðtæki þar sem hann stendur).
- turnoff (Karel slekkur á sér og forrit hættir keyrslu).

Auk þessara aðgerða eru nokkrar fyrirspurnar aðgerðir til þess að kanna umhverfi Karel. Þar má nefna aðgerð til að athuga hvort Karel standi ofan á boðtæki og hvort það sé veggur fyrir framan hann. Auk innbyggðu aðgerðanna var hægt að skilgreina ný stef og nota stjórn segðir á borð við '*ef*' og '*meðan*' í málinu. Málið sem notað var til þess að

stjórna Karel var undir sterkum áhrifum af Pascal. Hérna má sjá dæmi um forrit í þessu máli sem stjórnar Karel:

```
BEGINNING-OF-PROGRAM

  DEFINE turnright AS
  BEGIN
    turnleft;
    turnleft;
    turnleft
  END

  BEGINNING-OF-EXECUTION
    ITERATE 3 TIMES
    BEGIN
      turnright;
      move;
    END
    turnoff
  END-OF-EXECUTION

END-OF-PROGRAM
```

Pattis ásamt fleirum notaði þetta mál til kennslu fram undir miðjan 10. áratuginn en þá var útfærslan á kerfinu orðin úreld og það hætti að virka (Roberts, 2005).

2.2. Afsprengi Karel

Það hafa nokkur afsprengi komið af þessu kerfi. Þar má nefna *Karel++* sem notar forritunarmál líkt C++ og Java til þess að kenna hlutbundna forritun, *Guido van Robot*, sem er tól til þess að kenna Python og *rbKarel* sem er úrfærsla á kerfinu í málinu REALbasic.

Kerfi sem er kallað „*Karel the Robot learns Java*" er annað afsprengi upprunalega Karel og er notað í dag við Stanford háskólann. Það er notað í grunnáfanganum „CS106A: Programming Methodology". Það kerfi kennir hlutbundna forritun í Java og því eru hugtök á borð við klasa, aðferðir og erfðir í hávegum höfð. Sama forrit og að ofan, í „*Karel the Robot learns Java*", gæti litið einhvern veginn svona út:

```
1   import stanford.karel.*;
2
3   public class KarelExample extends Karel {
4
5     public void turnright() {
6       turnleft();
7       turnleft();
8       turnleft();
9     }
10
11    public void run() {
12      for(int i = 0; i < 3; i++) {
13        turnright();
14        move();
15      }
16    }
17  }
```

2.3. Karel fyrir byrjendur

Notkun á Róbótanum Karel hefur reynst vel við kennslu á mismunandi hugtökum í forritun. Kerfið hjálpar kennurum að útskýra efni á borð við algríma með því að útvega dæmum samhengi sem nemandinn á auðvelt með að skilja (Larason, 1995).

Hingað til hefur þessum kerfum oftast verið beint að nemendum í grunnnámi í tölvunarfræði eða einhverju sambærilegu. Það gerir það að verkum að forritin, á borð við þau sem dæmin að ofan sýna, eru illskiljanleg fyrir þann sem aldrei hefur snert forritun áður. Þessi kerfi þarfnast skilnings á forritunarmálum á borð við Pascal eða Java sem ekki voru hönnuð til kennslu. Þessi mál hafa því mun fleiri eiginleika og hærra flækjustig en þarf við kennslu í forritun á lægra stigi. Til dæmis má sjá að dæmin að ofan þarfnast skilnings á hugtökum á borð við erfðir sem Java hefur. Annar ókostur er að notendur þessara kerfa þurfa að þýða forritin sjálfir með þýðendum og nota önnur tól, s.s. Eclipse umhverfið, sem ekki eru auðskiljanleg fyrir byrjendur.

Karel kerfið hefur verið notað við kennslu hjá börnum í Þýskalandi sem eru á grunnskólaaldri með misgóðum árangri (Kiesmüller, 2009). Því virðist svo vera að hindranir þess að nota Karel við kennslu á lægri kennslu stigum séu tvískiptar. Annars vegar eru málin sem notuð eru til þess að stýra róbótanum ekki næginlega einföld. Hins vegar er erfitt fyrir byrjendur að keyra forritin sín. Ef þessar hindranir eru fjarlægðar ætti Karel að vera tilvalið til kennslu á lægri stigum.

3. Róbótinn Karel

Sem hluti af þessu verkefni hef ég hannað kerfi frá grunni sem útfærir hugmyndina um róbótann Karel. Þetta kerfi samanstendur af tveim megin hlutum. Annars vegar sérsmíðuðu forritunarmáli sem notað er við lausnir á verkefnum. Hins vegar notendaviðmóti til þess að keyra forrit, sjá aðgerðir Karel og búa til verkefni fyrir nemendur.

Í byrjun verkefnisins voru ákveðin lykilmarkmið sett sem kerfið átti að uppfylla. Þau voru fengin með því að skoða í samráði við leiðbeinanda fyrri útfærslur á róbótanum Karel og finna það sem betur mátti fara í þeim. Þessi markmið voru eftirfarandi:

- Auðveld uppsetning
- Auðflytjanlegt milli stýrikerfa
- Nota einfalt notendaviðmót
- Einfalt forritunarmál
- Góð meðhöndlun á villum

Fyrstu tvö markmiðin í listanum voru þau, sem þurfti fyrst að taka ákvörðun um. Þar sem hraði á bæði reikniaðgerðum og kerfinu í heild sinni var ekki grunn markmið var góður kostur að forrita kerfið í forritunarmálinu Java.

Java er ætlað til þess að gera forriturum kleift að skrifa kóða sem keyrir á mörgum mismunandi stýrikerfum. Þessi eiginleiki hefur verið kallaður WORA sem stendur fyrir „write once, run anywhere". Java bíður einnig upp á að pakka forritum í svokallaðar *Jar* skrár sem er auðveldlega hægt að keyra á öllum tölvum sem hafa Java sýndarvélina uppsetta.

Annar kostur við notkun á Java sýndarvélinni er að forritunarmálið Morpho keyrir ofan á Java. Morpho er létt mál sem bætir sveigjanleika og virkni við Java (Agnarsson, 2012a). Morpho hefur sína eigin sýndarvél sem getur tekið við Morpho smalamáli og keyrt það, sem leyfir kerfinu að þýða og keyra kóða, án erfiðleika fyrir notendur. Því varð Java fyrir valinu.

3.1. Forritunarmálið

Málið sem gert var fyrir kerfið hefur ekki hlotið neitt sérstakt nafn og er bara kallað „Karel forritunarmálið". Eins og kom fram var í upphafi reynt að gera málið einfalt til þess að notendur ættu auðvelt með að nota það en þó nógu öflugt til þess geta leyst flóknari dæmi.

Þetta mál er alíslenskt í þeim skilningi að öll lykilorð í málinu eru á íslensku, en líka að það er ekki hlutmengi af einhverju öðru máli heldur er það alveg nýtt. Málið tekur frá Java, Python og Morpho. Hér má sjá sama dæmi og var tekið í kafla 2 í þessu nýja máli:

```
1   stef hægri() {
2      vinstri()
3      vinstri()
4      vinstri()
5   }
6
7   endurtaka 3 {
8      hægri()
9      áfram()
10  }
```

Í þessum kafla er talað um fræðilegu hliðina á bak við forritunarmálið. Til þess að fá tæmandi útlistun á málfræði og eiginleikum málsins skal skoða Karel handbókina.

3.1.1. Stef

Eins og má sjá í dæminu hér að ofan er hægt að bæði skilgreina og kalla í stef. Hægt er að kalla í stef sem eru skilgreind neðar í forritstexta án þess að skilgreina frumgerð (e. function prototype). Þetta mál er bálkmótað að því leytinu til að það er hægt að skilgreina stef inní stefjum. Það er einnig hægt að kalla á stef og breytur í sömu eða efri földunarhæðum eins og önnur bálkmótuð mál leyfa. Í raun eru öll stef sem notandi skilgreinir lokanir (e. closure). Það er þó ekki hægt að skila stefi úr öðrum stefjum eða geyma lokanir í breytum. Þetta er gert til þess að einfalda málið. Bæði að einfalda málfræði þess og minnka þann skilning sem er nauðsynlegur til þess að nota málið.

Stef mega vera endurkvæm. Hérna er dæmi um notkun á endurkvæmni í útfærslu á fibo stefi. Þetta fibo stef tekur inn heila tölu n, sem ekki er neikvæð, og skilar n-tu fibonacci tölunni.

```
stef fibo(n) {
  ef n < 2 {
    skila 1
  }
  annars {
    skila fibo(n - 1) + fibo(n - 2)
  }
}
```

3.1.2. Breytur

Málið er einnig með breytur. Eins og áður kom fram geta breytur ekki innihaldið lokanir. Þær geta innihaldið eftirfarandi gildi:

- Tölur
- Strengi
- Boolean gildi
- EKKERT (NULL) gildið

Öll gildi eru í raun bendar á Java hluti. Tölurnar eru í raun Java Double hlutir og því eru tæknilega ekki til heiltölur í málinu. Hérna er dæmi um notkun á breytum í útfærslu á fibo stefi.

```
stef fibo(n) {
  ef n < 2 {
    skila 1
  }
  breyta a = 1
  breyta b = 1
  breyta temp = 0
  endurtaka n {
    temp = b
    b = a
    a = a + temp
  }
  skila a
}
```

3.1.3. Segðir og virkjar

Forritunarmálið býður upp á þær segðir sem flest mál bjóða upp á. Þær segðir eru meðal annarra *ef*, *meðan* og *skila* segðirnar. Þessar segðir virka eins og menn hafa vanist í öðrum málum. Það er þó einn munur á. Það er ekki leyfilegt að skilgreina breytur né stef í stjórnsegðum líkt og *ef* og *meðan*. Þetta er gert til þess að bæði einfalda málið og þulusmíði.

Málið er einnig með sömu grunnvirkja og önnur mál. Það skal taka fram að virkjarnir *og* (&&) og *eða* (||) nota ekki skammhlaups ákvörðun (e. Short-circuit evaluation). Með öðrum orðum eru ávallt allar innri segðir sem eru hluti af stærri boolean segðum reiknaðar.

3.2. Þýðandinn

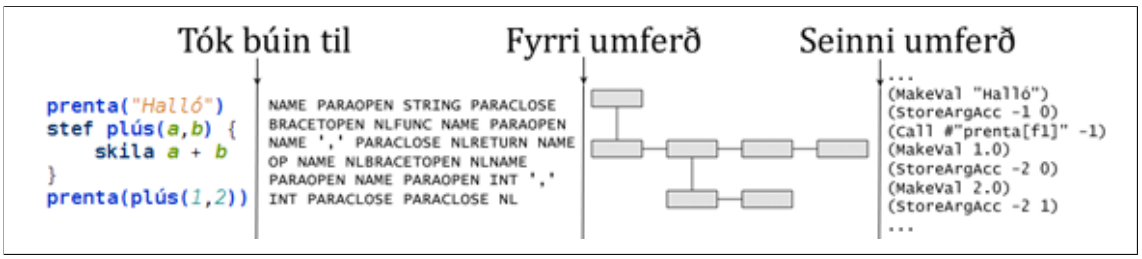
Þýðandinn fyrir Karel málið er skrifaður í Java. Hann er að mestu skrifaður frá grunni en hann notar JFlex (e. Fast Lexical Analyzer Generator for Java) tólið til lesgreiningar forritstexta.

Fyrri útgáfur af þýðandanum notuðu tólið BYACC/J (e. Berkeley v1.8 YACC-compatible parser generator for Java) við hjálp á þulusmíði. Það er þekkt að nota Flex og Yacc saman til þess að búa til þýðendur og þess vegna urðu þessi tól fyrir valinu. Það kom þó í ljós að BYACC/J hentaði ekki fyrir þetta verkefni, aðallega vegna þess að BYACC/J býður ekki upp á jafn góða stjórn á meðhöndlun á villum, eins og þýðandi sem er skrifaður frá grunni gerir. Það fór því þannig að þýðandinn var skrifaður upp á nýtt þótt sá gamli gæfi frá sér þulu sem virkaði.

Nýi þýðandinn er skrifaður með endurkvæmni ofanferð (e. Recursive descent). Þar sem BYACC/J býr til LALR þýðanda var ekki hægt að nota gömlu mállýsinguna fyrir nýja þýðandann. Það var því gerð ný LL mállýsing fyrir Karel forritunarmálið.

3.2.1. Ferill við smíð smalamáls

Í raun er nýi þýðandinn LL(1) þýðandi sem fer tvær umferðir við þulusmíð. Mynd 3.1 sýnir hvernig ferlið fer fram.



Mynd 3.1: Skref sem þýðandi tekur við þulusmíð

Tók búin til

Þetta er fyrsta skrefið í þulusmíðinni. Í þessu skrefi er forritstextinn brotinn upp í svokölluð tók. Þessi tók standa fyrir lykilorð, tölur, strengi og fleira sem við kemur málinu. Það skal þó tekið fram að þetta er gert samhliða fyrri umferð í þýðandanum en ekki á undan eins og mætti lesa úr mynd 3.1.

Fyrri umferð

Í þessari umferð tekur þýðandinn við tókum frá svokölluðum ”tokenizer”. Með þessum tókum er síðan smíðað tré, sem er kallað milliþula, sem inniheldur allar skilgreiningar og skipanir sem eru í forritstextanum.

Seinni umferð

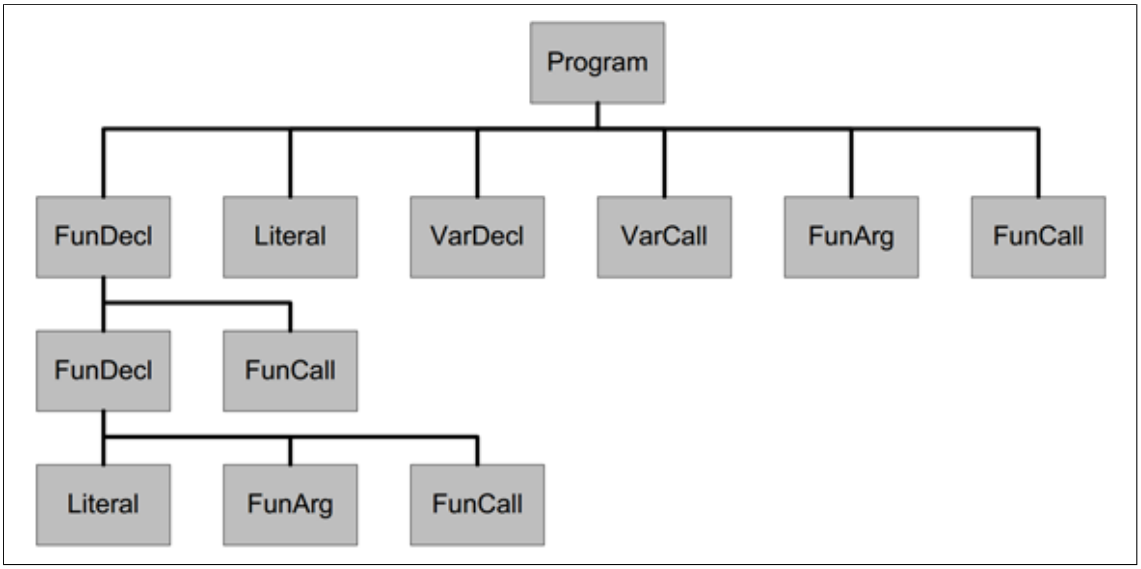
Í þessari umferð er síðan gengið endurkvæmt í gegn um milliþuluna. Fyrir hvern hnút í trénu er viðeigandi Morpho smalamálsskipun bætt við þuluna. Ef hnúturinn er kall á breytu eða stef, er leitað frá viðeigandi hnút og upp eftir trénu að viðeigandi skilgreiningu. Þetta leyfir málinu að ná fram sömu földunarreglum á stefjum og breytum og í öðrum bálkmótuðum málum.

Milliþulan

Eins og fram hefur komið smíðar þýðandinn milliþulu í fyrri umferð sinni. Milliþulan er ekkert annað en tré af skilgreiningum. Hver hnútur í trénu á eitt foreldri en sumir hnútar geta átt mörg börn. Við skulum íhuga eftirfarandi forritstexta:

```
1  stef b() {
2     stef c() {
3        prenta( "Halló" )
4     }
5     c()
6  }
7  breyta a = "Heimur"
8  prenta(a)
```

Milliþulan sem þýðandinn býr til við þýðingu á þessum texta er sýnd á mynd 3.2. Við sáum að þeir hnútar sem eiga börn eru í raun skilgreiningar á stefjum. Það má sjá að efsti hnúturinn í trénu er kallaður *Program* og er í raun „main" stefið í forritinu. Þýðandinn passar að hafa skilgreiningar á stefjum sem fyrstu börnin í hverri földunarhæð. Þetta tryggir að það sé búið að skilgreina viðeigandi lokun í Morpho sýndarvélinni áður en það er reynt að kalla á hana (því öll stef eru í raun lokanir).



Mynd 3.2: Dæmi um milliþulu

3.2.2. Smalamálið

Þýðandinn notar aðeins tólf Morpho smalamálsskipanir við þulusmíði á Karel forritunarmálinu. Morpho sýndarvélin býður upp á þó nokkuð fleiri skipanir (Agnarsson, 2012b). En skipanirnar í töflu 3.1 nægðu til þess að ná fram þeim eiginleikum sem sóst var eftir í Karel forritunarmálinu. Það skal þó tekið fram að *'_X:'* er tæknilega ekki skipun. Athuga skal að í töflunni táknar *AC* svokallaðan „Accumulator" í sýndarvél Morpho.

Tafla 3.1: Smalamálsskipanir sem þýðandinn notar og lýsing þeirra.

Nafn skipunar	Lýsing
_X:	Vistfang X í smalamálsþulu, þar sem X er jákvæð heiltala.
Call X Y	Kall á víðvært stef X með vakningarfærslu Y.
CallClosure X Y	Kall á lokun með X viðföngum sem er í *AC* með vakningarfærslu Y.
Fetch L P	Ná í gildi eða lokun í sæti P. Talan L táknar földunarhæð.
Go T	Hoppa á vistfang T.
GoFalse T	Hoppa á vistfang T ef gildi í *AC* er ósatt.
MakeClosure T L X	Búa til nýja lokun þar sem fyrsta skipun er í vistfangi T. Lokunin hefur X mörg viðföng. Talan L segir til um földunarhæð lokunarinnar.
MakeVal V	Setur gildið V í *AC*.
Return	Snýr til baka úr kalli.
Store L P	Vista gildi eða lokun í sæti P. Talan L táknar földunarhæð.
StoreArgAcc A P	Vista gildi í *AC* í sæti P í vakningarfærslu A.
StoreArgVal A P V	Vista gildi V í sæti P í vakningarfærslu A.

Það er kannski erfitt fyrir þann, sem hefur ekki komið að þulusmíði eða forritun í smalamáli, að sjá hvernig hægt er að búa til mál með aðeins þessum skipunum. Hér fyrir neðan má sjá lítinn forritsbút og smalamálið sem þýðandinn skilar þegar hann þýðir forritsbútinn.

```
1  stef heilsa() {
2     prenta("Góðan daginn "+nafn)
3  }
4  breyta nafn = "Guðmundur"
5  heilsa()
```

```
1  (MakeClosure _0 0 1)
2  (Go _1)
3  _0:
4  (MakeVal "Góðan daginn ")
5  (StoreArgAcc -2 0)
6  (Fetch 1 1)
7  (StoreArgAcc -2 1)
```

```
8   (Call #"+[f2]" -2)
9   (StoreArgAcc -1 0)
10  (Call #"prenta[f1]" -1)
11  (MakeVal null)
12  (Return 0)
13  _1:
14  (Store 0 0)
15  (MakeVal "Guðmundur")
16  (Store 0 1)
17  (Fetch 0 0)
18  (CallClosure 0 -1)
19  (Return 0)
```

Þulan byrjar á því í línu 1 að búa til lokun sem bendir á vistfangið _0. Í línu 2 hoppar þulan svo yfir skilgreininguna á búki stefsins **heilsa**. Skilgreiningin á stefinu er frekar skýr. Það sést þó í línu 6 að náð er í breytu í sæti 1 í einni földunarhæð fyrir ofan með skipuninni (Fetch 1 1). Þetta er kallið í breytuna ***nafn***. Það má líka sjá að í línum 11 og 12 skilar stefið NULL. Það gera öll stef þar sem ekki er tekið fram hvaða gildi skal skila. Í línu 14 sést að lokunin er vistuð í sæti 0 í núverandi földunarhæð (sem er í raun efsta földunarhæðin). Línur 14 - 18 eru síðan skilgreiningin á breytunni ***nafn*** og kall í stefið (sem er tæknilega séð lokun) **heilsa**.

Tekið skal fram að hægt er að opna sérstakan glugga til þess að sjá smalamálið sem þýðandinn býr til. Hann er opnaður með því að keyra *karel.jar* með eftirfarandi hætti.

```
1   java -jar karel.jar -debug
```

3.2.3. Stuðningur fyrir „Taka Skref"

Notendaviðmótið býður upp á að „Taka Skref". Þessu er lýst nánar í handbók Karel. Til þess að styðja þennan eiginleika setur þýðandinn inn eftirfarandi bút í smalamálsþuluna fyrir hverja <*almennsegð*> (Sjá handbók fyrir skilgreiningu):

```
1   (StoreArgVal -1 0 X)        ;;;Stepping
2   (Call #"karelStep[f1]" -1)  ;;;Stepping
```

Þarna stendur X (eitt viðfangið í skipuninni StoreArgVal) einfaldlega fyrir þá línu sem segðin er í. Fallið *karelStep* þjónar tvíþættum tilgangi. Annars vegar lætur það þráðinn sem forrit notanda keyrir á bíða þar til ýtt er á "Taka Skref". Undantekningin á þessu er þegar forritið er keyrt án þess að vera að taka skref. Hins vegar hjálpar stefið við að stöðva forrit í keyrslu, hvort sem þau virka sem skyldi eða eru með einhverja kvilla eins

og endalausa lykkju.

Þessa forritsbúta má sjá reglulega í smalamáli á forritum notanda. Ég hef þó tekið þá út í dæmunum í þessu riti til þess að gera þau skilvirkari.

3.2.4. Betrumbætur

Þrátt fyrir að þýðandinn sé vel heppnaður er hann þó ekki fullkominn. Það hefur tekist nokkuð vel að meðhöndla villur og smalamáls þulan er nokkuð stílhrein sem þýðandinn skilar. Það eru þó nokkur atriði sem mætti laga við smíði þulu. Við skulum sem dæmi skoða eftirfarandi forritsbút:

```
1   stef hrun() {
2      skila hrun()
3   }
```

Þessi forritsbútur er í sjálfu sér órökréttur og gerir það að verkum að ef kallað er í stefið **hrun** fer forritið í endalausa „lykkju". Smalamálið sem þýðandinn býr til fyrir þetta stef bendir okkur á tvö atriði sem betur mættu fara í þulusmíði. Hérna má sjá smalamálið:

```
1   (MakeClosure _0 0 1)
2   (Go _1)
3   _0:
4   (Fetch 1 0)
5   (CallClosure 0 -1)
6   (Return 0)
7   (MakeVal null)
8   (Return 0)
9   _1:
10  (Store 0 0)
```

Fyrra atriðið má greinilega sjá í línum 6 - 8. Í línu 6 má sjá að stefið skilar gildi úr stefinu og línur 7 - 8 eru aldrei keyrðar. Þetta gerir þýðandinn til þess að tryggja að NULL sé skilað nema annað sé tekið fram. Þýðandinn mætti sleppa línum 7 - 8 í aðstæðum sem þessum. Þótt þetta geri svo sem ekkert til gagnvart keyrslu á forritum bætir þetta óþarfa skipunum í þuluna.

Seinna atriðið er að ef kallað yrði í þetta stef, myndi kösin í Morpho sýndarvélinni fyrr eða síðar fyllast af vakningafærslum. Það er vegna þess að Karel forritunarmálið gerir ekki sérstakar ráðstafanir varðandi halaendurkvæmni. Það væri þó hægt að bæta því við með því að nota smalamálsskipunina *BecomeClosure* þar sem það á við í stað *CallClosure*.

Eitt atriði en hefur komið upp sem mætti lagfæra. Hérna sést forritsbútur og smalamálsþulan fyrir hann:

```
1  breyta a = "Halló heimur"
2  prenta(a)
```

```
1  (MakeVal "Halló heimur")
2  (Store 0 0)
3  (Fetch 0 0)
4  (StoreArgAcc -1 0)
5  (Call #"prenta[f1]" -1)
```

Þarna má sjá að í línu 3 er hlaðið gildi inn í *AC* sem er þar nú þegar. Þessi segð er því óþörf og mætti fjarlægja.

3.3. Notendaviðmót

Frá fyrsta degi var lagt upp með að hafa viðmótið eins einfalt og unnt væri. Það var gert til þess að auðvelda notendum að keyra forrit og sjá bæði aðgerðir Karel og heiminn hans. Viðmótið skiptist gróflega upp í tvo hluta:

Keyrslu Umhverfi

Þessi hluti viðmótsins gerir notendum kleift að skrifa og keyra forrit, sjá villur frá þýðanda og sjá heim Karel. Í þessum hluta er textaritill sem býður upp á þá eiginleika sem nútíma ritlar bjóða upp á. Þar má nefna allar grunn aðgerðir eins og *klippa* og *líma* en einnig svokallað „syntax highlighting" sem litar forritstexta notenda til þess að gera hann auðlesanlegri.

Borðvinnsla

Þetta er sá hluti viðmótsins sem hjálpar notendum að skilgreina heim Karel. Heimurinn er skilgreindur með borði. Borðvinnslan gerir notendum auðvelt fyrir að bæta inn veggjum, kössum og útgöngum í borðið. Hugsa má borðvinnsluna sem eins konar teikniforrit því að það þarf aðeins að nota músina til þess að breyta borðum. Hún leyfir síðan að vista borðin og því er hægt að senda borð á milli manna og tölva.

Hægt er að lesa nánar um viðmótið og notkun á því í handbókinni.

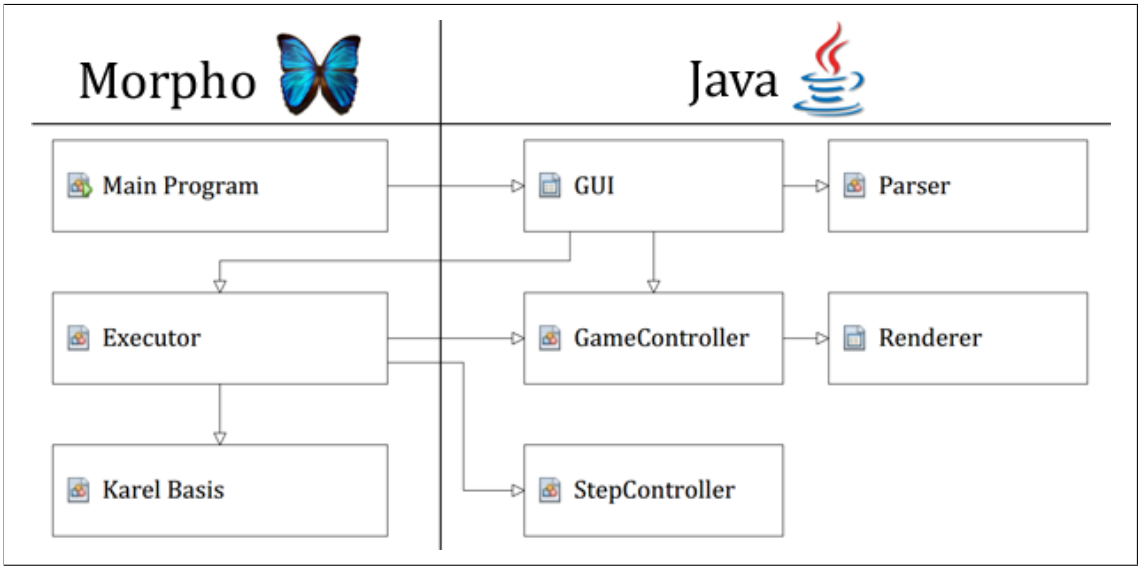
3.4. Uppbygging kerfisins

Sjálft kerfið er töluvert veigamikið og er rúmlega 10.000 línur af kóða (með athugasemdum). Kerfinu má gróflega skipta upp í eftirfarandi hluta:

Tafla 3.2: Meginhlutar kerfisins

Nafn skipunar	Lýsing
Main Program	Megin forrit kerfisins sem er ræst við upphaf keyrslu.
Executor	Tekur við Morpho smalamáli og keyrir það.
Karel Basis	Skilgreiningar á innri föllum og virkjum Karel forritunarmálsins.
GUI	Notendaviðmót kerfisins.
Parser	Þýðandi kerfisins.
GameController	Stjórnar leiknum sjálfum. Tekur við skipunum frá *Executor* og stjórnar *Renderer*.
StepController	Hjálpar við að stoppa forrit og að "Taka skref".
Renderer	Teiknar heim Karel.

Mynd 3.3 lýsir gróflega hvernig kerfið er uppbyggt. Eins og áður hefur komið fram er kerfið skrifað í tveimur forritunarmálum. Þeir hlutar kerfisins sem eru skrifaðir í Morpho eru í vinstri dálki, en hinir sem skrifaðir eru í Java eru í hægri dálki. Þegar keyrsla hefst á kerfinu er búið til tilvik af Morpho sýndarvélinni sem keyrir síðan „Main Program".



Mynd 3.3: Gróf mynd af uppbyggingu kerfisins

Samskipti milli kerfishluta er einnig gróflega lýst á mynd 3.3. Þegar ör bendir frá *A* til *B* gefur það til kynna að *A* noti *B* á einn eða annan hátt. Sem dæmi um þetta notar *GUI* hlutinn *Parser* til þess að þýða forrit notenda yfir í Morpho smalamál. Þar næst sendir *GUI* smalamálið til *Executor* sem þýðir það yfir í vélamál og keyrir það. Það má segja að málin skipti með sér hlutverkum. Þeir hlutar kerfisins sem koma að viðmóti eru skrifaðir í Java en sá hluti sem gerir okkur kleift að keyra forrit er skrifað í Morpho.

Mismunandi hönnunarmynstur voru höfð í huga við gerð kerfisins. Singelton mynstrið var notað fyrir *GameController* og *StepController* til þess að geta kallað á þá úr *Executor* á auðveldann hátt. Sjálfur *Renderer* hluti kerfisins er forritaður eftir skili (e. interface) sem þýðir að auðvelt væri að breyta heimi Karel yfir í þrívíðan heim.

Allur kóði kerfisins er útgefinn með „Apache License 2.0” leyfinu. Það þýðir að allir geta nálgast og breytt kóðanum. Hann má nálgast á vefsíðu verkefnisins:

```
http://morpho.cs.hi.is/karel
```

4. Framtíð Karel

Þetta kerfi sem hefur verið hugarfóstur hitt í rúmt ár er nú orðið að veruleika. Það stenst allar þær væntingar sem ég lagði til þess. Það má nú samt segja að kerfið sé ekki alveg fullþroskað. Ef segja ætti hvar á líftíma sínum (e. release life cycle) kerfið væri mætti segja að það væri í svokölluðum ”BETA” fasa. Sá fasi lýsir sér þannig að kerfið uppfyllir alla þá eiginleika sem ætlast er til að það hafi en það á eftir að framkvæma veigamiklar prófanir, í höndum notenda, fyrir villum og öðrum ófyrirsjáanlegum atriðum. Einnig á eftir að prófa að fullu hvernig íslenskir nemendur taka við þessu umhverfi og hvernig þeim gengur að leysa verkefni.

Þetta kerfi er ekki smíðað með gróða í huga og þess vegna er þetta opið hugbúnaðarkerfi (e. open source). Það veldur því að þeir aðilar sem búa yfir réttri kunnáttu geta breytt, bætt og aðlagað kerfið eftir aðstæðum. Þess vegna eru engar hömlur á kerfinu og möguleikar þess miklir.

Nú er komið að þeim aðilum sem standa að kennslu barna, hvort sem þeir eru opinberir eða einkaaðilar, að prófa Róbótann Karel. Ég hef fulla trú á því að bæði kennarar og nemendur muni hafa gaman af því að búa til og leysa verkefni með hjálp Karel. Ég bind miklar vonir við að Róbótanum Karel verði sýndur áhugi hér á landi því ég veit að hann getur lagt sitt af mörkum til þess að betrumbæta forritunarkennslu á lægri kennslustigum.

Heimildir

Agnarsson, S. (2012a). Morpho Manual. Óbirt efni.

Agnarsson, S. (2012b). The Morpho Virtual Machine. Óbirt efni.

Kiesmüller, U. (2009). Diagnosing Learners' Problem-Solving Strategies Using Learning Environments with Algorithmic Problems in Secondary Education. *ACM Transactions on Computing Education (TOCE)*, 9(3):1–26.

Labuscagne, C. (2008). How to teach programming to novices. *Communications of the ACM*, 51(6):11.

Larason, K. (1995). Using Karel the robot as a classroom motivator. *3C ON-LINE*, 2(4):6.

Liu, C.-C., Cheng, Y.-B., and Huang, C.-W. (2011). The effect of simulation games on the learning of computational problem solving. *Computers & Education*, 57(3):1907–1918.

Olson, P. (2012). Why Estonia Has Started Teaching Its First-Graders To Code. Sótt 23. janúar 2013 af `http://www.forbes.com/sites/parmyolson/2012/09/06/why-estonia-has-started-teaching-its-first-graders-to-code/`.

Roberts, E. (2005). Karel the Robot learns Java. Sótt 18. febrúar 2013 af `http://www.stanford.edu/class/cs106a/handouts/karel-the-robot-learns-java.pdf`.

Sigurðsson, O. (2012). RSS straumar og atvinnuleit. Sótt 19. apríl 2013 af `http://www.visir.is/rss-straumar-og-atvinnuleit/article/2012711299957`.

Sölvadóttir, R. (2013). Endurforritun menntunar. Sótt 17. apríl 2013 af `http://www.advania.is/um-advania/blogg-advania/blogg/2013/04/10/Endurforritun-Menntunar/`.

A. Handbók

A.1. Karel kerfið

A.1.1. Uppsetning

Karel notar Java sýndarvélina og því þarf Java keyrslu umhverfið að vera uppsett á vélinni sem á að keyra Karel. Hægt er að ná í Java keyrslu umhverfið á eftirfarandi slóð:

```
http://www.java.com/
```

Engin frekari uppsetning er nauðsynleg til þess að keyra Karel umhverfið. Það þarf þó að niðurhlaða Karel keyrsluskránni af vefnum inn á vélina og keyra síðan viðeigandi skrá. Slóð vefsíðunnar sem hýsir allar skrár tengdar Karel er:

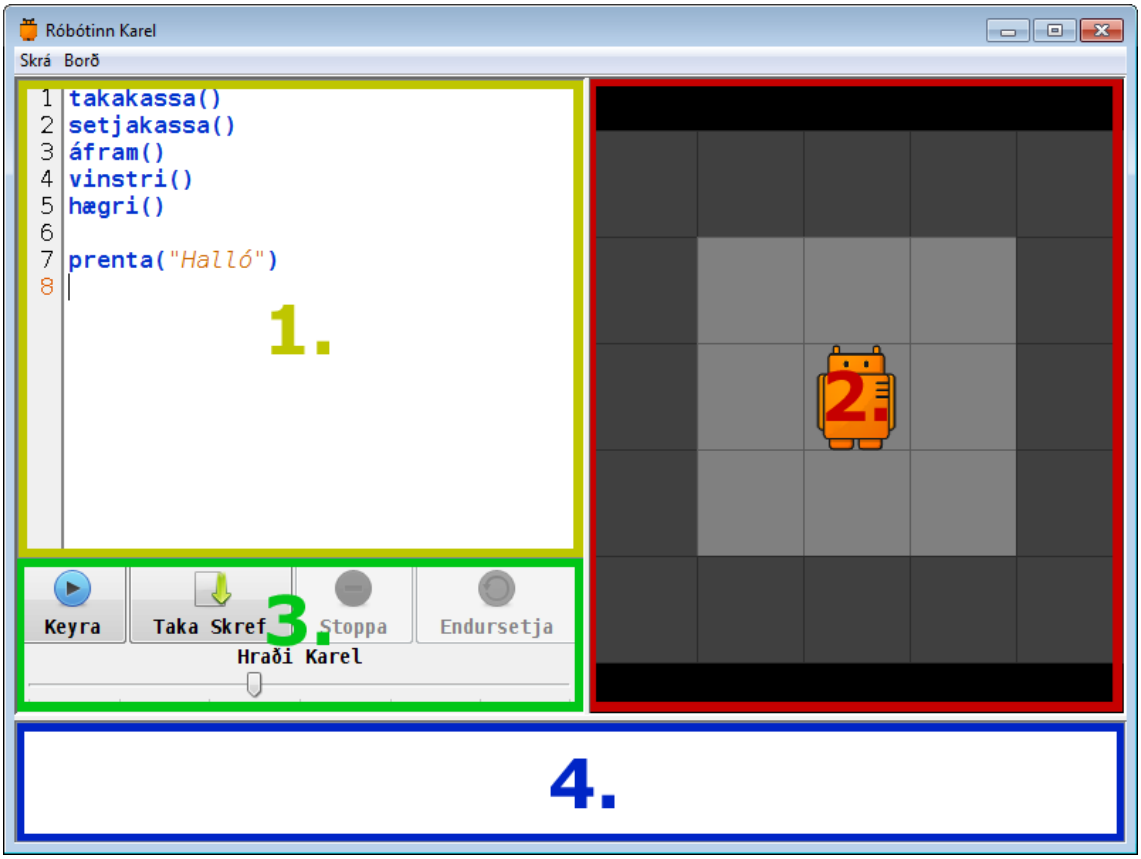
```
http://morpho.cs.hi.is/karel
```

Hægt er að ná í keyrsluskrána á vefsíðunni með því að smella á "Hlaða niður Karel".

A.1.2. Keyrslu umhverfi

Sjálft Karel umhverfið skiptist niður í tvo hluta. Annars vegar keyrsluumhverfið og hins vegar borðvinnslu. Þegar keyrsluumhverfið er opnað í fyrsta sinn fer glugginn á sjálfgefinn stað á skjánum með sjálfgefna stærð. Hins vegar man umhverfið síðustu stærð og staðsetningu sem notuð var og munu síðari keyrslur á kerfinu taka mið af því.

Keyrslu umhverfið í Karel hefur fjóra megin hluta. Mynd A.1 telur þessa hluta upp með lituðum númerum og ferhyrningum. Næstu fjórir undirkaflar lýsa hlutverki hvers kerfishluta nánar.



Mynd A.1: Karel keyrsluumhverfið

1. Textaritill

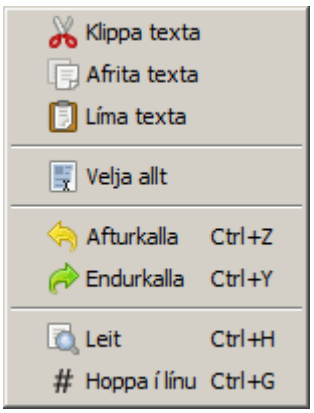
Textaritill kerfisins er svæði númer 1 á mynd A.1. Hann er notaður til þess að skrifa þann kóða sem skal keyra á róbótanum Karel.

Textaritillinn notar svokallað „syntax highlighting". Það virkar þannig að lykilorð, breytunöfn, fastar og fleiri atriði í setningarbyggingu málsins er litað með einkennandi lit. Þetta er eiginleiki sem hjálpar notendum að lesa kóða betur. Til þess að hjálpa við villugreiningu á forritum notenda sýnir textaritillinn númer hverrar línu vinstra megin við forritstexta. Einnig lýsir textaritillinn upp þann textabút sem inniheldur villu. Sjá má ítarlega lýsingu á meðhöndlun á villum í kafla A.1.7.

Hægt er að nota grunn flýtileiðir sem hinn almenni notandi hefur vanist í öðrum textaritlum. Þær eru:

Nýtt forrit (Ctrl + N)	Opna forrit (Ctrl + O)
Vista forrit (Ctrl + S)	Klippa texta (Ctrl + X)
Afrita texta (Ctrl + C)	Líma texta (Ctrl + V)
Afturkalla (Ctrl + Z)	Endurkalla (Ctrl + Y)
Snögg leit (Ctrl + F)	Leit (Ctrl + H)
# Hoppa í línu (Ctrl + G)	Loka Karel (Ctrl + Q)

Hægt er að hægri smella með músarhnappi á textaritil til þess að fá upp valmynd af aðgerðum sem hægt er að framkvæma.



Mynd A.2: Valmynd í textaritil

2. Borð

Borð kerfisins er svæði númer 2 á mynd A.1. Það sýnir þann heim eða umhverfi sem Karel býr í. Þegar notandinn keyrir forrit getur hann séð þær aðgerðir sem Karel framkvæmir í borðinu. Sjá kafla A.1.4 fyrir nánari lýsingu á heimi Karel.

Þegar kerfið er fyrst keyrt er opnað sjálfgefið tómt borð. Athuga skal að ekki er hægt að keyra forrit þegar ekkert borð er opið. Notandi getur opnað önnur borð með því að velja í valmynd Borð -> Opna Borð. Annar möguleiki er að opna borðvinnslu kerfisins með því að velja í valmynd Borð -> Borðvinnsla. Sjá nánar kafla A.1.3 um notkun á borðvinnslu.

3. Stjórntæki

Stjórntæki kerfisins eru í svæði númer 3 á mynd A.1. Þar eru fjórir takkar sem notaðir eru við keyrslu á forriti og endursetning á borði. Þar er líka sleði sem stjórnar hraðanum

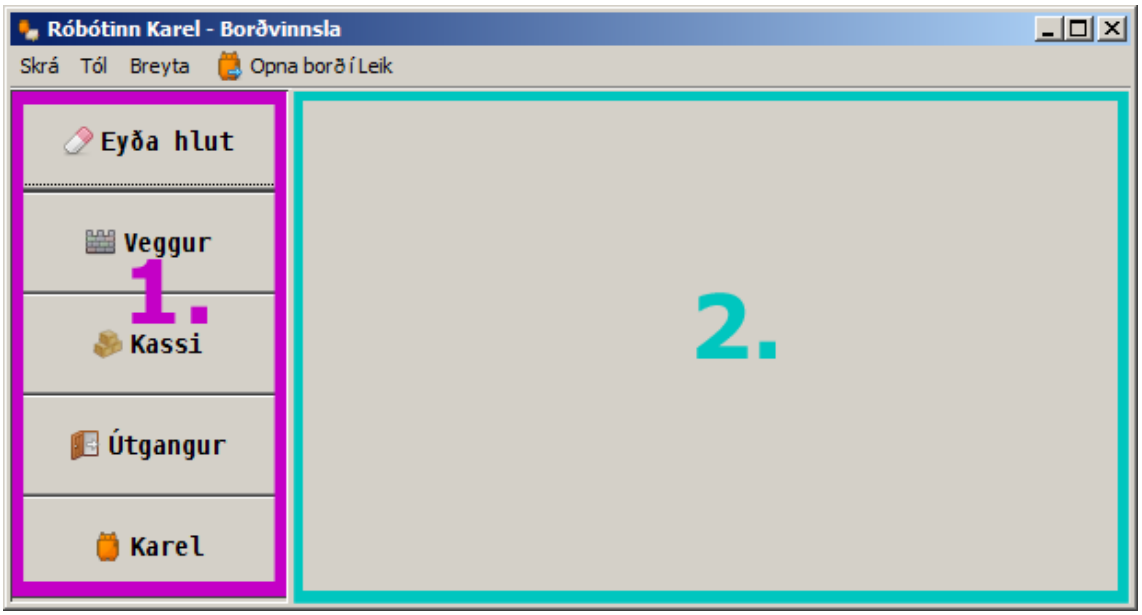
á aðgerðum Karel í borðinu. Sjá kafla A.1.6 fyrir nánari lýsingu á því hvernig forrit eru keyrð í kerfinu.

4. Skilaboðagluggi

Skilaboðaglugginn er svæði númer 4 á mynd A.1. Allar villur eða önnur skilaboð sem kerfið þarf að senda notenda eru birt í skilaboðaglugganum. Einnig skal tekið fram að það sem er sent innbyggða stefinu *prenta* birtist í skilaboðaglugganum.

A.1.3. Borðvinnsla

Borðvinnsla er sá hluti kerfisins sem gerir notendum kleift að búa til sín eigin borð fyrir Karel eða breyta borðum sem aðrir notendur hafa búið til.



Mynd A.3: Karel borðvinnsla

Borðvinnslan hefur eftirfarandi flýtileiðir:

Nýtt borð (Ctrl + N)	Opna borð (Ctrl + O)
Vista borð (Ctrl + S)	Afturkalla (Ctrl + Z)
Endurkalla (Ctrl + Y)	Loka (Ctrl + Q)

Viðmót borðvinnslunnar skiptist upp í tvo hluta og eins og sést á mynd A.3. Svæði númer 1 á myndinni inniheldur tiltæk tól en svæði númer 2 er borðið sjálft.

Í stuttu máli skilgreinir borð umhverfið sem Karel er í. Nánari lýsingu á borðunum má lesa í kafla A.1.4.

Til þess að breyta borðinu velur notandinn viðeigandi tól. Þegar tól er valið þá verður takki viðeigandi tóls gráleitur. Þá er hægt að beita valda tólinu á borðið með því að vinstrismella á borðið sjálft. Hér að neðan kemur lýsing á hverju tóli fyrir sig. Flýtileiðir á lyklaborði fyrir hvert tól eru sýndar í sviga.

- *Eyða hlut* (Ctrl + 1). Tekur út vegg, kassa, útgang eða Karel.
- *Veggur* (Ctrl + 2). Setur niður vegg.
- *Kassi* (Ctrl + 3). Setur niður *X* kassa þar sem *X* er tala sem notandi gefur upp. Einnig er hægt að hafa tólið valið og nota skrunhjól á mús til þess að bæta við eða taka í burtu kassa.
- *Útgangur* (Ctrl + 4). Setur niður útgang.
- *Karel* (Ctrl + 5). Setur niður Karel. Aðeins einn Karel getur verið í borðinu.

Athuga skal að undir öllum kringumstæðum er hægri músartakki jafngildur tólinu *Eyða hlut*.

Þegar notandi er búinn með borðið sem hann er að vinna í getur hann annað hvort valið að *Vista borð* eða *Opna borð í Leik*. Það skal athuga að Karel þarf alltaf að vera til staðar í borðinu til að hægt sé að vista eða opna borð í leik.

A.1.4. Heimur Karel

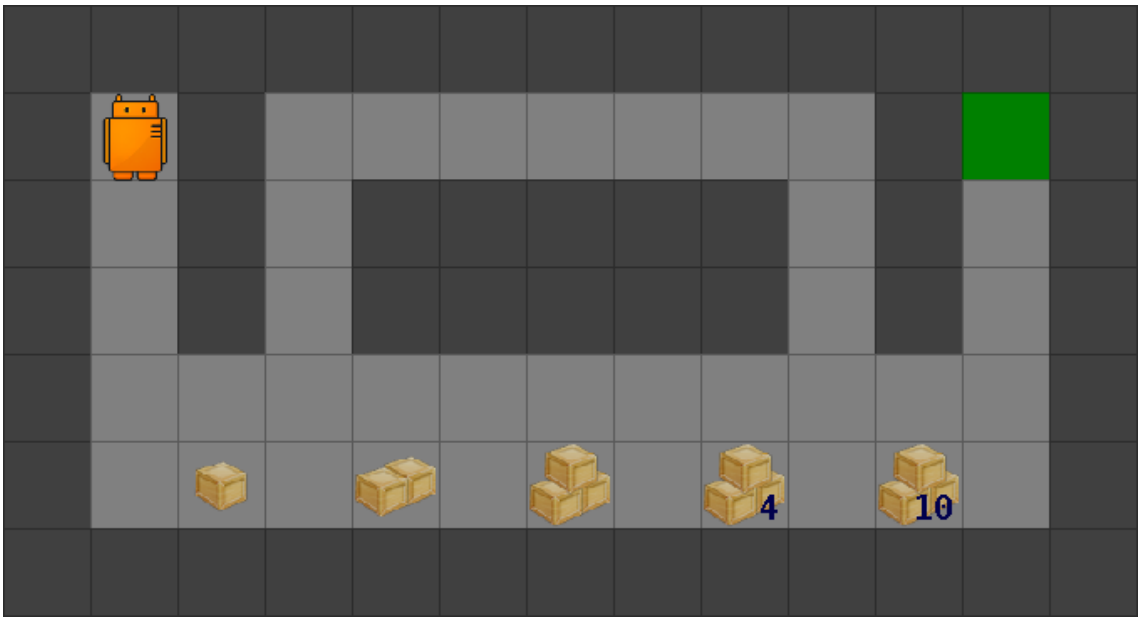
Heimurinn eða umhverfið sem Karel býr í er skilgreint með borði. Öll borð hafa vegg sem umlykur borðið. Þetta er til þess að Karel komist ekki út úr borðinu. Því er minnsta mögulega borðið 3 x 3 reitir að stærð. Stærsta mögulega borðið er 100 x 100 reitir. Allar stærðir þar á milli eru leyfilegar. Athuga skal að breidd og hæð borðsins þarf ekki að vera sú sama.

Á mynd A.4 má sjá dæmigert borð. Hver reitur í borðinu getur táknað einn af eftirfarandi hlutum:

Tafla A.1: Mismunandi reitir í heimi Karel

Tómt svæði	Veggur	Kassi	Útgangur	Karel
				

Það geta verið fleiri en einn kassi á hverjum reit. Þá kemur mynd af fleiri kössum upp að tölunni 3. Ef fleiri en 3 kassar eru á hverjum reit er aðeins mynd af 3 kössum en talan í reitnum segir þá til um fjölda kassa. Þetta sést á mynd A.4.



Mynd A.4: Dæmi um Karel borð

A.1.5. Aðgerðirnar hans Karel

Karel hefur 5 grunnaðgerðir. Þessar aðgerðir gera notenda kleift að leysa verkefni með Karel. Í kafla A.3 eru listuð upp þau innbyggðu stef sem kalla þarf í til þess að framkvæma þessar aðgerðir. Aðgerðirnar eru:

- Áfram
  - Karel tekur eitt skref í þá átt sem hann snýr. Hann getur gengið yfir allar gerðir af reitum nema veggi.

- Vinstri og Hægri
  - Karel getur snúið sér annaðhvort til vinstri eða hægri. Hann snýr sér 90° í senn.
- Taka upp kassa
  - Karel tekur upp einn kassa ef hann stendur ofan á einum eða fleiri kössum. Ef hann stendur ekki ofan á kassa gerist ekki neitt.
- Setja niður kassa
  - Karel setur niður einn kassa. Engar hömlur eru á því hvað Karel getur sett niður marga kassa. Karel getur þó ekki sett niður kassa á útgang.

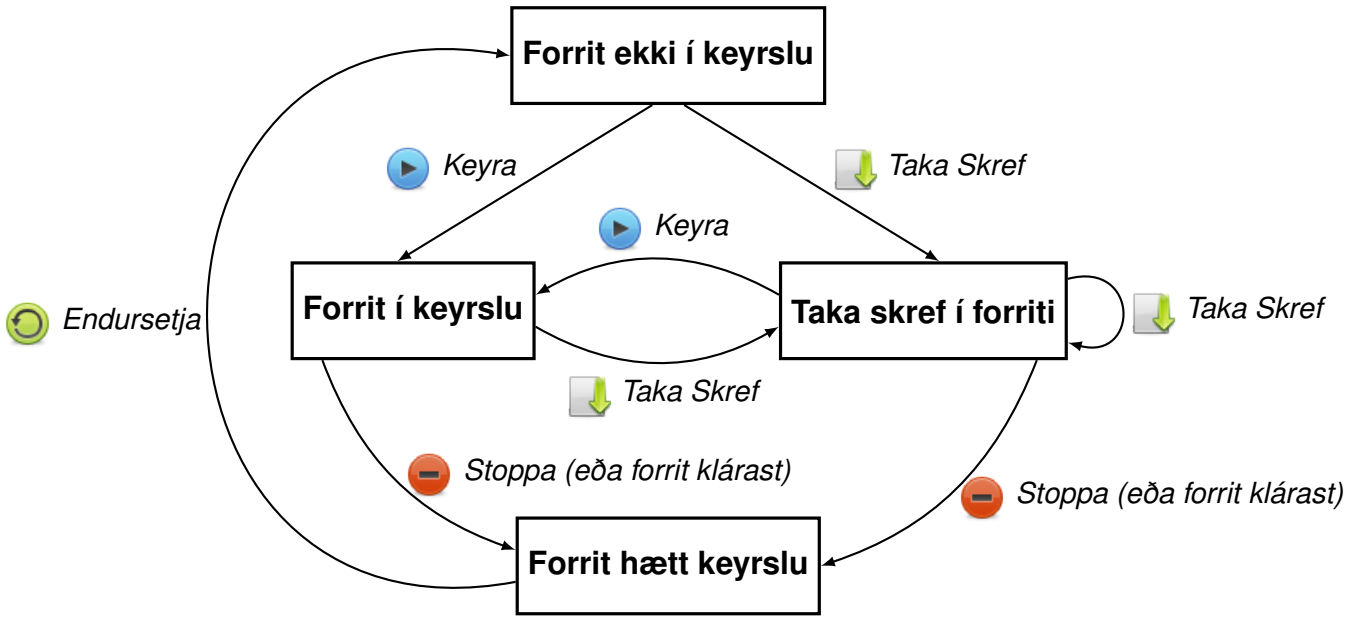
Þegar kerfið er keyrt opnast sjálfgefið forrit sem kallar einmitt á þessar fimm grunnaðgerðir.

```
1  takakassa()
2  setjakassa()
3  áfram()
4  vinstri()
5  hægri()
6
7  prenta( "Halló" )
```

A.1.6. Keyrsla á forriti

Þegar notandi vill keyra forrit notar hann stjórntæki, sem lýst var stuttlega í kafla A.1.2. Best er að lýsa keyrslu á forriti með stöðuvél. Hana má sjá á mynd A.5.

Efst uppi er staða sem heitir „Forrit ekki í keyrslu". Þetta er byrjunarstaða, þ.e.a.s. sú sem tekur á móti notanda þegar ekkert hefur verið átt við stjórntækin. Ef smellt er á takkann *Keyra* þá keyrir forritið og stöðuvélin er þá komin í stöðu „Forrit í keyrslu". Af sama skapi fer stöðuvélin í stöðu „Taka Skref í forriti" er smellt er á takkann *Taka Skref* í stöðu „Forrit ekki í keyrslu". Hver staða hefur ákveðna eiginleika.



Mynd A.5: Stöðuvél sem lýsir keyrslu á forriti

Forrit ekki í keyrslu

Þetta er upphafsstaðan í stöðuvélinni. Í henni er aðeins hægt að ýta annað hvort á takkann *Keyra* eða takkann *Taka Skref*. Ef engar villur eru í þeim kóða, er forritið keyrt og stöðuvélin skiptir um stöðu samkvæmt mynd A.5. Athuga skal að aðeins hægt að nota textaritilinn þegar forrit er ekki í keyrslu.

Forrit í keyrslu

Í þessari stöðu er forritið keyrandi óhindrað. Hægt er að smella á tvo takka meðan á keyrslu stendur. Ef smellt er á takkann *Stoppa* hættir forritið keyrslu. Hins vegar ef smellt er á takkann *Taka Skref* hættir forritið að keyra óhindrað og stöðuvélin fer í stöðu „Taka skref í forriti".

Taka skref í forriti

Þessi staða býður upp á að taka skref í forritinu. Í hvert sinn sem smellt er á takkann *Taka Skref* fer forritið eina línu áfram. Sú lína sem næst verður framkvæmd er yfirstrikuð í textaritli með gráum lit. Línur sem innihalda ekki segðir eru ekki teknar með. Ef smellt er á *Keyra* mun forritið keyra óhindrað. Í þessari stöðu er líka hægt að stoppa keyrslu með því að smella á *Stoppa* til þess að stöðva keyrslu á forriti.

Forrit hætt keyrslu

Í þessari stöðu er aðeins hægt að smella á takkann *Endursetja* sem fer með stöðuvélina í upphafsstöðu og endursetur heiminn hjá Karel.

A.1.7. Meðhöndlun á villum

Þegar villur eru í forriti notanda mun kerfið benda á villurnar. Á mynd A.6 má sjá dæmigerða villu.

```
1 prenta("Segð í lagi")
2 prentaa("Segð ekki í lagi")
```

```
Forritið er með villur:
Lína 2: Stef að nafni 'prentaa' með 1 viðfang ekki skilgreint!
```

Mynd A.6: Dæmi um villu

Í þessu dæmi er fyrst kallað í innbyggða stefið 'prenta'. Næst er kallað á stef með nafninu 'prentaa' sem er ekki skilgreint. Þá sýnir kerfið notandanum villuna með því að bæði yfirstrika hana með rauðu í textaritli og lýsa villunni í skilaboðaglugga. Þetta gerir kerfið fyrir allar villur sem það finnur.

A.2. Karel forritunarmálið

Fyrir róbótann Karel er til sérhannað forritunarmál. Málið hefur ekki fengið neitt einkennandi nafn og er aðeins kallað *Karel forritunarmálið*. Þetta forritunarmál er alíslenskt og tekur mið af málum á borð við Java og Python.

A.2.1. Grunneiningar málsins

Lykilorð

Forritunarmálið hefur nokkur lykilorð sem ekki má nota sem nöfn á stefjum eða breytum.

*'**stef**', '**breyta**', '**ef**', '**annars**', '**annars ef**', '**meðan**', '**endurtaka**', '**skila**', 'satt', 'ósatt', 'ekki', 'eða', 'og'*

Öll þessi lykilorð hafa sinn tilgang sem kemur fram í næstu köflum.

Tákn

Táknin sem forritunarmálið notar eru eftirfarandi:

'(', ')', '{', '}', '#', ',', NL

Þessi tákn eru notuð til þess að aðskilja ýmisa þætti í málinu. Það skal athuga að NL stendur fyrir ný lína, en sem dæmi má taka að hver segð þarf að byrja á nýrri línu.

Athugasemdir

Skrifa má athugasemdir í forritstexta. Kerfið hundsar allar athugasemdir og má því skrifa hvað sem er í þeim. Athugasemd í einni línu skal byrja með táknunum '##' en athugasemd sem á að spanna margar línur skal byrja á táknunum '#{' og enda á '}#'. Þær er hægt að gera á eftirfarandi hátt:

```
1  ## Þetta er athugasemd í einni línu
2  #{
3  Þetta er athugasemd
4  sem spannar
5  margar línur
6  }#
```

A.2.2. Gildi

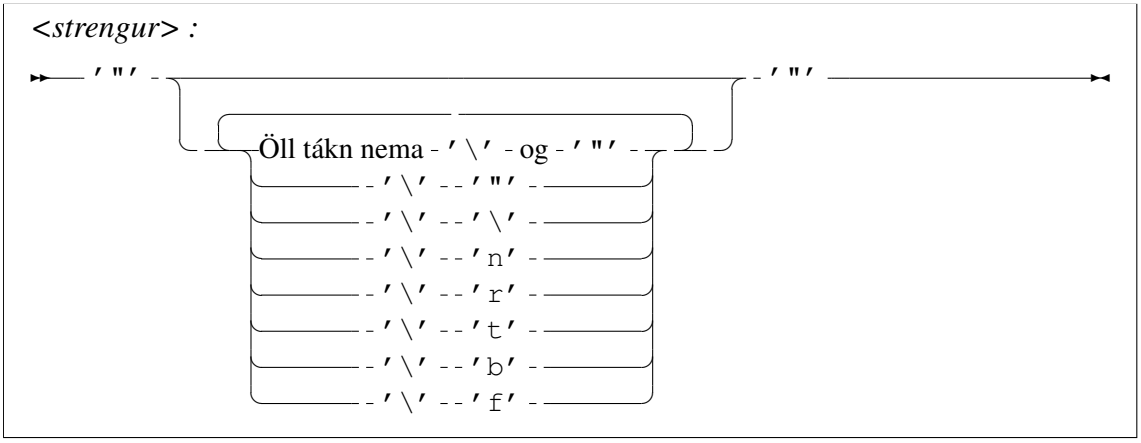
Gildin í Karel forritunarmálinu eru strengir, tölur, sanngildi og EKKERT. Þeim er lýst í þessum kafla.

Strengir

Strengir eru einn af þremur lesföstum sem notaðir eru í málinu. Þeir eru til þess að tákna einhvern texta og eru appelsínugulir á litinn í textaritil. Dæmi um notkun á þeim væri:

```
1  "Halló"  ## Strengur
2  "Heimur"  ## Annar strengur
```

Hægt er að nota algengustu „sérstaka stafi" eins og til dæmis '\n' sem gerir nýja línu í strengunum.



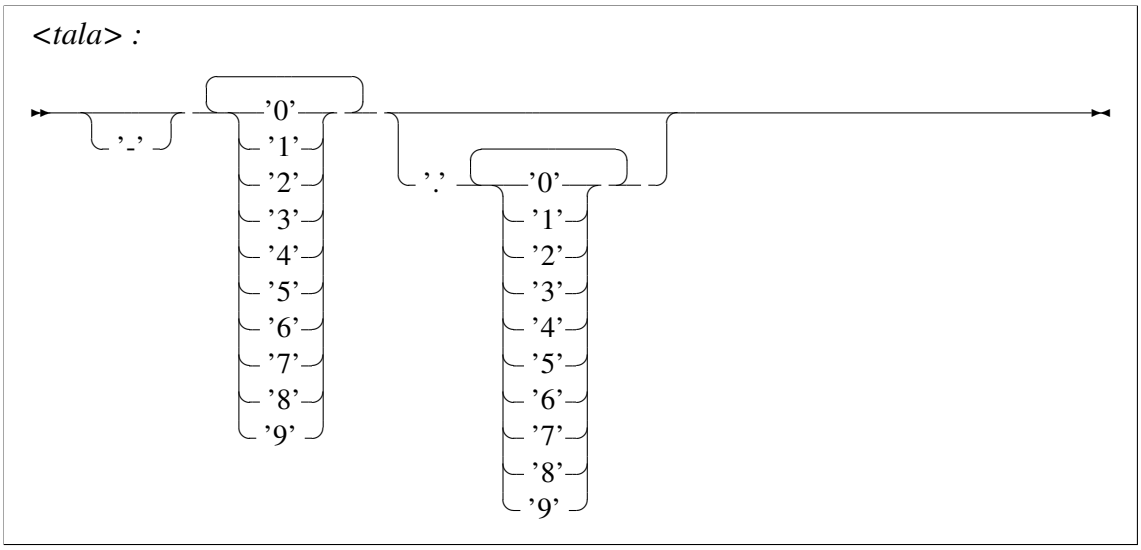
Mynd A.7: Járnbrautarrit fyrir <strengur> .

Ef skoðaða er mynd A.7 er hægt að ímynda sér að þetta séu járnbrautarteinar og að það sé lest á þeim sem er staðsett uppi til vinstri. Þegar kerfið les forrit notanda er það í sjálfu

sér að aka þessari lest eftir teinunum. Ef forrit notanda kemur með röð tákna sem brýtur í bága við járnbrautarritið kemur villa.

Tölur

Tölur eru annar lesfastinn í málinu. Þær eru grænbláar á litinn í forritstexta og má skilgreina eins og sýnt er á mynd A.8.



Mynd A.8: Járnbrautarrit fyrir <tala> .

Dæmi um lesfasta fyrir tölur gæti til dæmis verið:

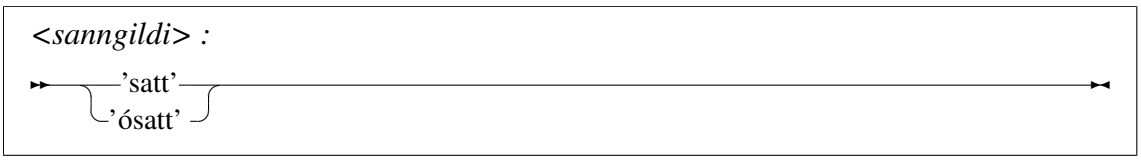
```
42  ## Tala
-15  ## Neikvæð tala
3.1415  ## Talan PI
```

Sanngildi

Sanngildin eru þriðji lesfastinn í málinu. Þau eru annaðhvort 'satt' eða 'ósatt'.

Dæmi um sanngildi:

```
satt      ## Sanngildið satt
ósatt     ## Sanngildið ósatt
```



Mynd A.9: Járnbrautarrit fyrir <sanngildi> .

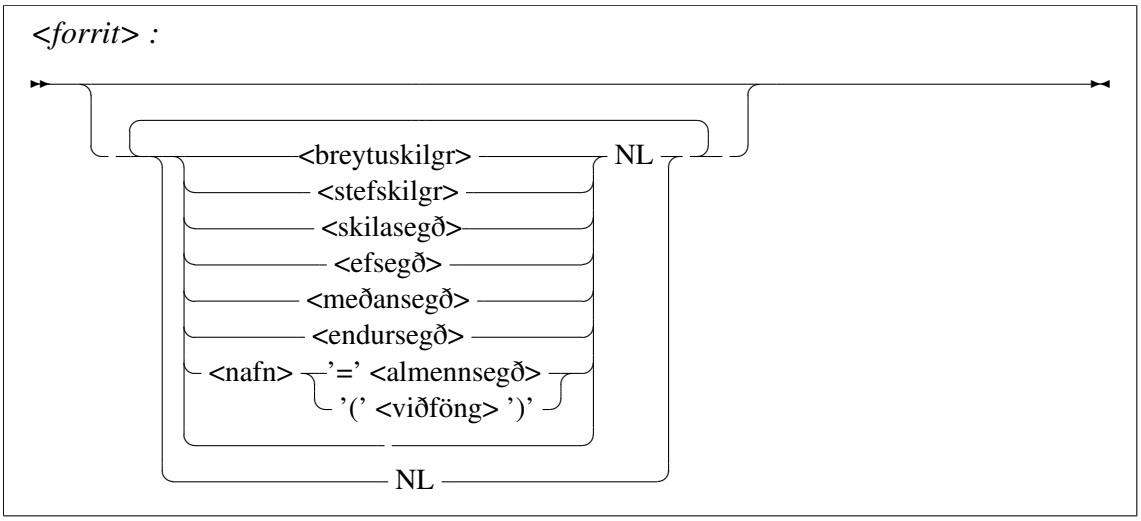
Það skal tekið fram að öll gildi í málinu, þar með talið tölur og strengir, eru í raun sönn nema EKKERT og *ósatt* gildið.

EKKERT gildið

Það skal tekið fram að *EKKERT* gildið er líka til staðar í málinu. Því er skilað úr stefjum ef ekki er tekið fram hverju stefið skilar. Það kemur einnig út úr reiknisegðum sem nota virkja ekki rétt. Það er þó ekki hægt að nota *EKKERT* gildið í forrits texta.

A.2.3. Forrit

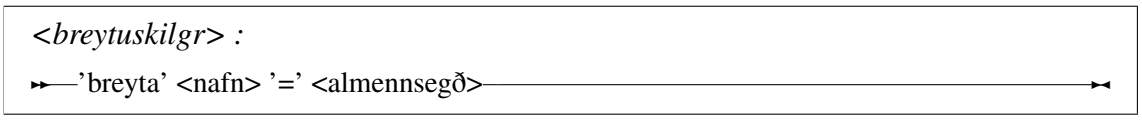
Til þess að búa til forrit sem leysa verkefni þarftu ýmsar segðir. Karel forritunarmálið býður því upp á þær grunnsegðir sem önnur mál hafa.



Mynd A.10: Járnbrautarrit fyrir <forrit>:

Á mynd A.10 sést járnbrautarrit af grunnmállýsingu forritunarmálsins. Því má til dæmis sjá á mynd A.10 að forrit má vera tómt. Það má líka innihalda eina eða fleiri segðir en NL (ný lína) þarf alltaf að vera á milli þeirra.

A.2.4. Breytur



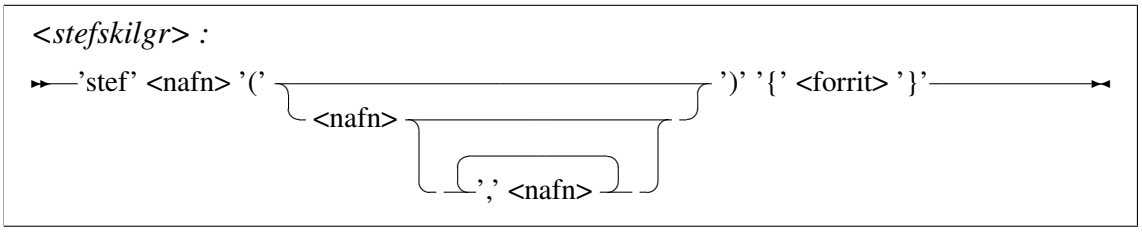
Mynd A.11: Járnbrautarrit fyrir breytu skilgreiningar segðir.

Breytur sem ekki eru skilgreindar í stefjum eru víðværar. Það þýðir að hægt er að nálgast þær hvar sem er í forritstextanum. Hins vegar má skilgreina breytur inn í stefi og ef það er gert er aðeins hægt að nálgast þær innan þess stefs. Dæmi um notkun á breytum:

```
1  breyta x = 42
2  breyta y = "Halló"
3  x = x + 3
4  y = "Halló" + "Heimur"
```

Í þessu dæmi má sjá breytuskilgreiningu. Eins og var sýnt neðst á mynd A.10 má frumstilla breytu með hvaða *<almennsegð>* sem er. Að sjálfsögðu geta breytur í málinu innihaldið öll leyfileg gildi málsins, en það er það eina sem þær geta innihaldið. Kerfið gefur upp villu ef sama breytunafnið er skilgreint oftar en einu sinni (í sömu földunarhæð).

A.2.5. Stef



Mynd A.12: Járnbrautarrit fyrir skilgreiningar segðir stefja.

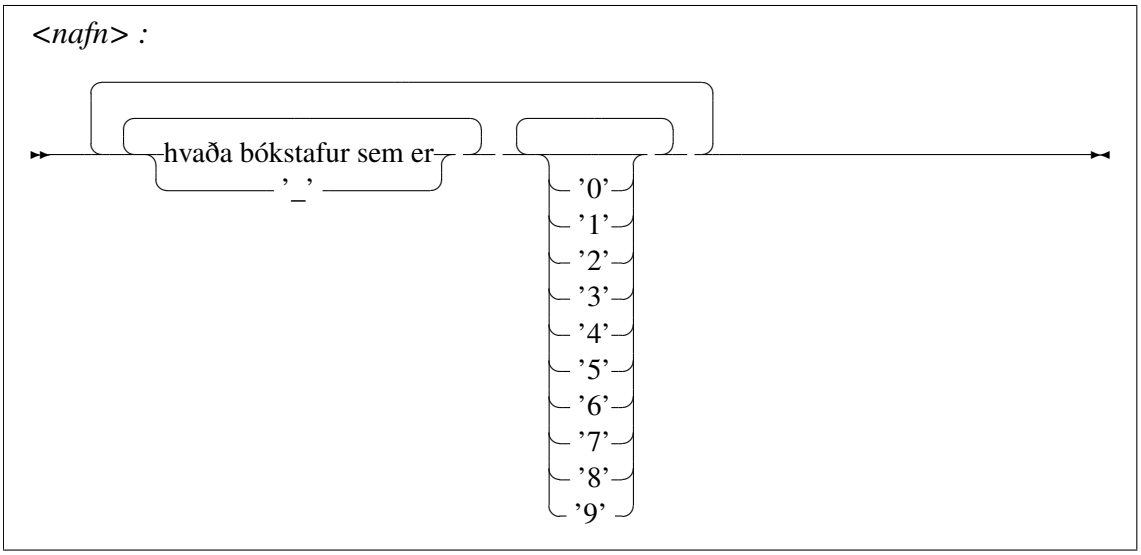
Stef í Karel forritunarmálinu geta tekið núll eða fleiri viðföng. Athuga skal að það er hægt að skilgreina stef inní stefi. Stef skila alltaf gildi. Ef það er ekki tekið fram skilar það *EKKERT* gildinu. Dæmi um skilgreiningu á stefi gæti verið:

```
1  stef heilsa(nafn) {
2     prenta("Halló "+nafn)
3  }
```

Þetta stef tekur inn viðfang sem gæti t.d. verið nafnið *"Karel"* og prentar á skilaboða glugga *Halló Karel*. Stefið gerir það með því að kalla í stefið 'prenta'. Prenta er innbyggt stef í kerfinu. Til þess að sjá útlistun á innbyggðum stefjum skal sjá kafla A.3. Köllum í stef er lýst málfræðilega neðst á mynd A.10.

A.2.6. Nöfn

Eins og sást hér að ofan nota bæði breytu- og stefskilgreiningar <*nafn*> . Öll nöfn þurfa að byrja á bókstaf eða '_' en síðan mega fylgja bókstafir, '_' og/eða tölustafir



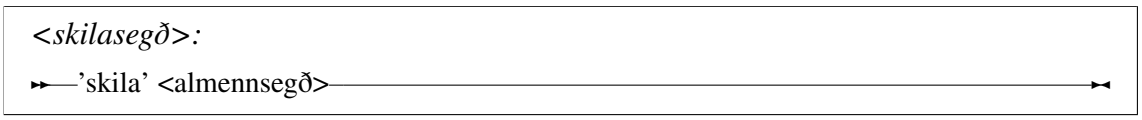
Mynd A.13: Járnbrautarrit fyrir <nafn> .

Dæmi um skilgreiningar á nöfnum eru til dæmis:

```
1  nafn
2  nafn2
3  _nafn3
4  nafn_4
```

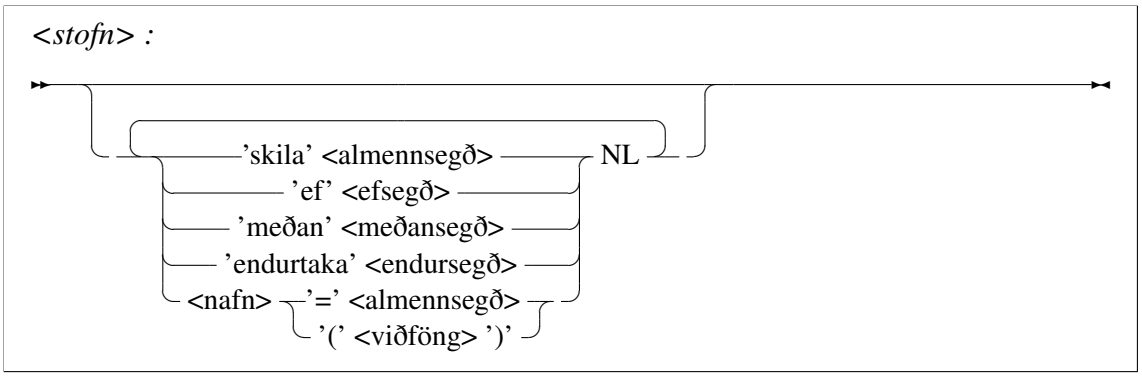
A.2.7. Skila segðin

Þessi segð er notuð til þess að skila gildum út stefjum. Hana má því aðeins nota í stefjum.



Mynd A.14: Járnbrautarrit fyrir 'skila' segðina.

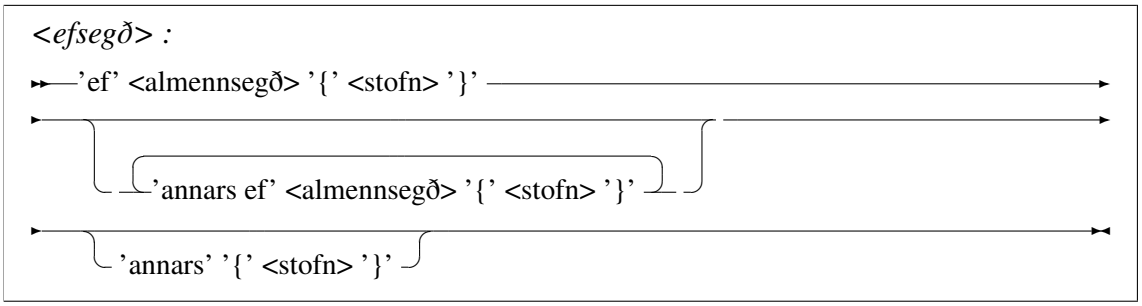
A.2.8. Stjórn segðir



Mynd A.15: Járnbrautarrit fyrir <stofn> .

Stjórn segðirnar nota <*stofn*> sem líkist verulega <*forrit*> en leyfir ekki breytu- og stefskilgreiningar. Skilgreiningu á <*stofn*> má sjá á mynd A.15. Þar sem allar stjórn segðir nota <*stofn*> eru breytu- og stefskilgreiningar ekki leyfðar inn í þeim.

Ef segðin



Mynd A.16: Járnbrautarrit fyrir 'ef' segðina.

Þessi segð gerir notanda kleift að taka ákvarðanir í forritinu. Mynd A.16 sýnir hvernig hún er skilgreind málfræðilega. Hún byrjar á lykilorðinu *'ef'* og einhverju skilyrði. Ef það skilyrði er uppfyllt (er *satt*) er kóðinn í <*stofn*> þess skilyrðis keyrður og hoppað er yfir önnur skilyrði ef þau eru tekin fram. Hægt er að nota *'annars ef'* núll eða oftar til þess taka fram fleiri skilyrði. Það má síðan enda segðina á *'annars'* og <*stofn*> þess er keyrður ef ekkert efri skilyrða er uppfyllt. Dæmi um notkun á segðinni væri:

```
1   breyta aldur = 15
2
3   ef aldur < 6 {
4       prenta("Leikskóli")
5   }
6   annars ef aldur < 16 {
7       prenta("Grunnskóli")
8   }
9   annars ef aldur < 20 {
10      prenta("Framhaldsskóli")
11  }
12  annars {
13      prenta("Vinna")
14  }
```

Meðan segðin

```
<meðansegð> :
►►──'meðan' <almennsegð> '{' <stofn> '}'──────────────────────────────►◄
```

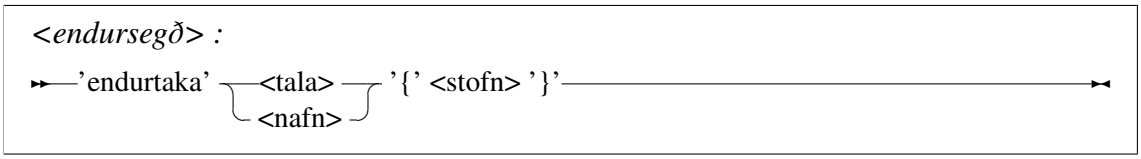
Mynd A.17: Járnbrautarrit fyrir 'meðan' segðina.

Meðan segðin er notuð til þess að gera endurtekningar í forritinu. Hún endurtekur <*stofn*> sinn svo lengi sem skilyrðið er uppfyllt. Dæmi um notkun á segðinni sem prentar út 5 sinnum *Kakan er lygi* í skilaboðaglugga.

```
1   breyta umferðir = 5
2
3   meðan umferðir > 0 {
4       prenta("Kakan er lygi")
5       umferðir = umferðir - 1
6   }
```

Í þessu dæmi er skilyrðið umferðir > 0. Í hverri endurtekningu er talan umferðir minnkuð um einn með segðinni umferðir = umferðir - 1. Þar sem talan umferðir var 5 til þess að byrja með mun forritið prenta út "Kakan er lygi" fimm sinnum.

Endurtaka segðin



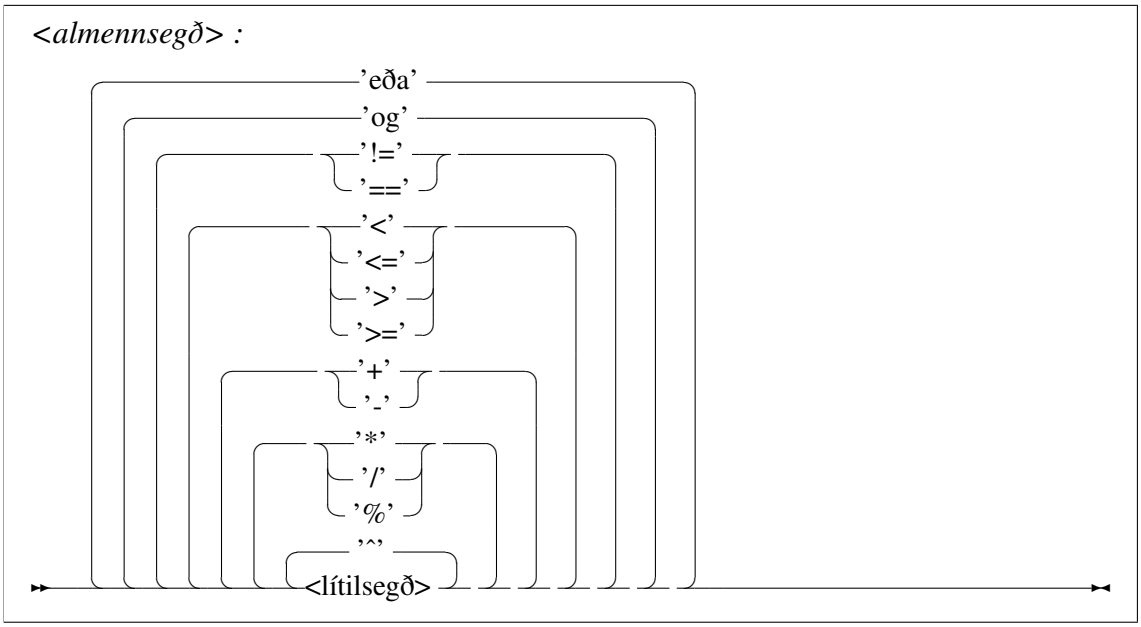
Mynd A.18: Járnbrautarrit fyrir 'endurtaka' segðina.

Endurtaka segðin er í raun bara einfölduð *'meðan'* segð. Í stað þess að fara eftir skilyrði er hún endurtekin X sinnum þar sem X getur annaðhvort verið heiltala eða breyta með heiltölugildi. Hérna er forritsbútur sem framkvæmir sömu aðgerðir og dæmið fyrir *'meðan'* segðina

```
1  endurtaka 5 {
2     prenta("Kakan er lygi")
3  }
```

Ef breytan sem er notuð í *'endurtaka'* segðina er strengur eða sanngildi, er hún aldrei keyrð og ef breytan inniheldur tölu sem er ekki heil tala það er námundað niður.

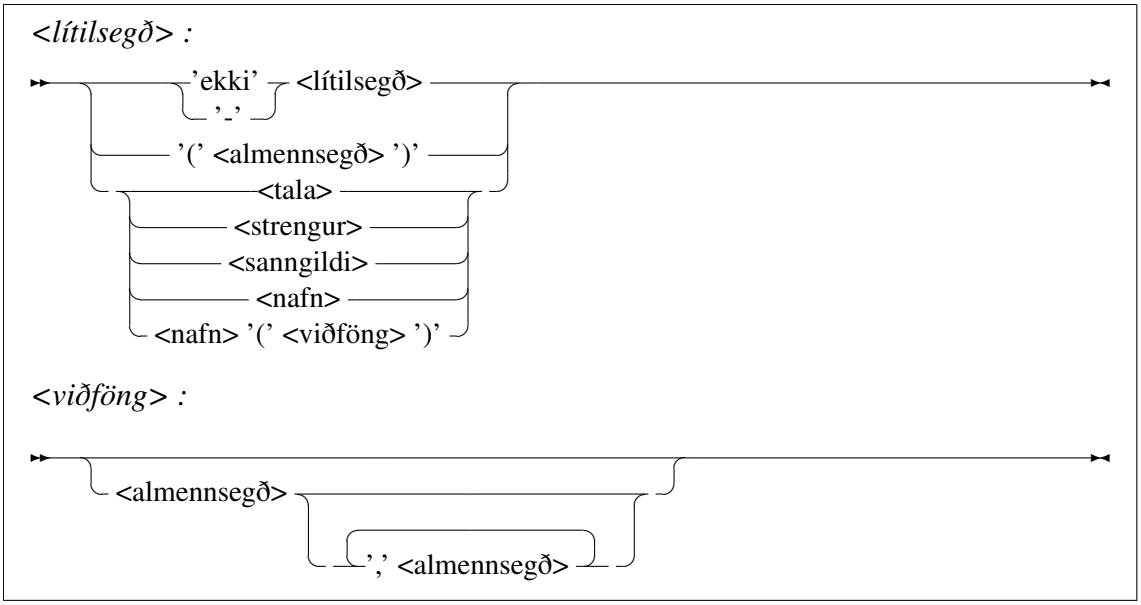
A.2.9. Virkjar, viðföng og búkur



Mynd A.19: Járnbrautarrit fyrir <almennsegð> .

Hingað til hafa flestar segðir innihaldið segð að nafni <*almennsegð*>. Þetta er hin almenna reiknisegð. Hún inniheldur alla þá virkja sem málið hefur uppá að bjóða. Mynd A.19 gæti verði villandi í fyrstu en það sem hún er að segja okkur er að virkinn 'eða' hefur minnstan forgang og virkinn ˆ hefur mestan forgang. Þessir virkjar eru allir tvíundarvirkjar.

Allir þessir virkjar vefjast utan um <*lítilsegð*>. Á mynd A.20 má sjá málfræðibyggingu hennar. Í henni eru einundarvirkjarnir 'ekki' og '-'. Hún gerir okkur líka kleift að hafa almennar segðir inn í svigum og kalla á gildi, breytur og stef. Ef virkjar eru notaðir með ólöglegum viðföngum skila þeir EKKERT gildinu.



Mynd A.20: Járnbrautarrit fyrir <lítilsegð> og <viðföng> .

Eins og við sáum í kafla A.2.5 geta stef tekið 0 eða fleiri viðföng. Viðföngin geta verið hvaða <*almennsegð*> sem er. Ef viðföngin eru fleiri en eitt skal vera ',' á milli þeirra.

A.3. Virkjar og innbyggð stef

Í þessum kafla eru taldir upp þeir virkjar og innbyggð stef sem eru í kerfinu. Með þeim fylgir notkunarlýsing sem útskýrir hvernig skal nota þá. Ef þeir er notaðir á einhvern veg sem brýtur í bága við notkunarlýsingu þeirra munu þeir annað hvort skila EKKERT gildinu eða einhverju órökréttu gildi. Ólögleg notkun þeirra getur einnig valdið því að forrit „keyrir" án þess að nokkuð gerist. Því skal gæta þess að fylgja forskilyrðum virkjanna vel. Það sama gildir um innbyggðu stefin.

A.3.1. Virkjar

eða

Notkun:	***x*** *=* ***y*** *eða* ***z***
Forskilyrði:	***y*** *og* ***z*** *er hvaða gildi sem er.*
Eftirskilyrði:	*Ef* ***y*** *og* ***z*** *eru EKKERT gildið eða ósatt gildið er* ***x*** *= ósatt,* ***x*** *= annars satt.*

og

Notkun:	***x*** *=* ***y*** *og* ***z***
Forskilyrði:	***y*** *og* ***z*** *er hvaða gildi sem er.*
Eftirskilyrði:	*Ef* ***y*** *eða* ***z*** *eru EKKERT gildið eða ósatt gildið er* ***x*** *= ósatt,* ***x*** *= annars satt.*

!=

Notkun:	***x*** *=* ***y*** *!=* ***z***
Forskilyrði:	***y*** *og* ***z*** *er hvaða gildi sem er.*
Eftirskilyrði:	***x*** *er satt ef* ***y*** *og* ***z*** *eru ekki eins, annars er* ***x*** *ósatt.*

==

Notkun:	*x = y == z*
Forskilyrði:	*y og z er hvaða gildi sem er.*
Eftirskilyrði:	*x er satt ef y og z eru af sama tagi og með sama gildi, annars er x ósatt.*

<

Notkun:	*x = y < z*
Forskilyrði:	*y og z er hvaða gildi sem er.*
Eftirskilyrði:	*x er satt ef y og z eru af sama tagi og y er minna en z, annars er x ósatt.*

<=

Notkun:	*x = y <= z*
Forskilyrði:	*y og z er hvaða gildi sem er.*
Eftirskilyrði:	*x er satt ef y og z eru af sama tagi og y er minna eða jafnt og z, annars er x ósatt.*

>

Notkun:	*x = y > z*
Forskilyrði:	*y og z er hvaða gildi sem er.*
Eftirskilyrði:	*x er satt ef y og z eru af sama tagi og y er stærra en z, annars er x ósatt.*

>=

Notkun:	*x = y >= z*
Forskilyrði:	*y og z er hvaða gildi sem er.*
Eftirskilyrði:	*x er satt ef y og z eru af sama tagi og y er stærra eða jafnt og z, annars er x ósatt.*

+

Notkun:	*x = y + z*
Forskilyrði:	*#1: y og z eru tölur.* *#2: Annaðhvort y eða z er strengur og hitt er eitthvað gildi.*
Eftirskilyrði:	*#1: x er útkoma samlagningar á y og z.* *#2: x er nýr strengur sem fenginn með því að skeyta saman y og z.*

-

Notkun:	*x = y - z*
Forskilyrði:	*y og z eru tölur.*
Eftirskilyrði:	*x er útkoma frádráttar z af y.*

*****

Notkun:	*x = y * z*
Forskilyrði:	*y og z eru tölur.*
Eftirskilyrði:	*x er margfeldi z og y.*

/

Notkun:	***x*** *=* ***y*** */* ***z***
Forskilyrði:	***y*** *og* ***z*** *eru tölur,* ***z*** *má ekki vera 0.*
Eftirskilyrði:	***x*** *er útkoma deilingar* ***y*** *með* ***z****.*

%

Notkun:	***x*** *=* ***y*** *%* ***z***
Forskilyrði:	***y*** *og* ***z*** *eru heilar tölur.* ***y*** *verður að vera >= 0.*
Eftirskilyrði:	***x*** *er leifar þegar* ***y*** *er deilt með* ***z****. 0 <=* ***x*** *< | z |.*

^

Notkun:	***x*** *=* ***y*** *^* ***z***
Forskilyrði:	***y*** *og* ***z*** *eru tölur. Þær mega vera neikvæðar.*
Eftirskilyrði:	***x*** *er* ***y*** *haft upp í veldi* ***z****.*

ekki (einundarvirki)

Notkun:	***x*** *= ekki* ***y***
Forskilyrði:	***y*** *er hvaða segð sem er.*
Eftirskilyrði:	*Ef* ***y*** *er ósatt eða EKKERT þá er* ***x*** *= satt, en annars* ***x*** *= ósatt.* ***x*** *er andstæða* ***y****.*

- (einundarvirki)

Notkun:	***x*** *=* ***-*** ***y***
Forskilyrði:	***y*** *er tala.*
Eftirskilyrði:	***x*** *= 0 -* ***y****.*

A.3.2. Innbyggð stef

prenta

Notkun:	***prenta(x)***
Forskilyrði:	***x*** *er hvaða gildi sem er.*
Eftirskilyrði:	*Búið er að prenta* ***x*** *í skilaboða glugga og meðfylgjandi nýrri línu.*

áfram

Notkun:	***x*** *=* ***áfram()***
Forskilyrði:	*ekkert*
Eftirskilyrði:	*Karel hefur tekið eitt skref áfram ef hann gat það.* ***x*** *==* *satt* *ef skrefið tókst annars er* ***x*** *==* *ósatt.*

vinstri

Notkun:	***vinstri()***
Forskilyrði:	*ekkert*
Eftirskilyrði:	*Karel hefur snúið sér 90° til vinstri (rangsælis).*

hægri

Notkun:	***hægri()***
Forskilyrði:	*ekkert*
Eftirskilyrði:	*Karel hefur snúið sér 90° til hægri (réttsælis).*

takakassa

Notkun:	***x*** *=* ***takakassa()***
Forskilyrði:	*ekkert*
Eftirskilyrði:	*Karel hefur tekið upp einn kassa ef hann stendur ofan á kassa hrúgu.* ***x*** *== 1 ef tókst að taka upp kassa, annars er* ***x*** *== 0.*

setjakassa

Notkun:	***setjakassa()***
Forskilyrði:	*ekkert*
Eftirskilyrði:	*Karel hefur sett niður einn kassa á reitinn sem hann stendur á ef sá reitur er ekki útgangur.*

kassi

Notkun:	***x*** *=* ***kassi()***
Forskilyrði:	*ekkert*
Eftirskilyrði:	***x*** *== satt ef Karel stendur ofan á 1 eða fleiri kössum, annars er* ***x*** *== ósatt.*

veggur

Notkun:	***x*** *=* ***veggur()***
Forskilyrði:	*ekkert*
Eftirskilyrði:	***x*** *== satt ef Karel getur ekki tekið eitt skref áfram (veggur fyrir), annars er* ***x*** *== ósatt.*

útgangur

Notkun:	***x*** = ***útgangur()***
Forskilyrði:	*ekkert*
Eftirskilyrði:	***x*** == *satt ef Karel stendur á útgangi, annars er* ***x*** == *ósatt.*

átt

Notkun:	***x*** = ***átt()***
Forskilyrði:	*ekkert*
Eftirskilyrði:	***x*** *er strengur sem segir til um áttina á Karel.*

hnitx

Notkun:	***z*** = ***hnitx()***
Forskilyrði:	*ekkert*
Eftirskilyrði:	***z*** *er x hnit á staðsetningu Karel í borðinu. x hnitið er 1 ef Karel er alveg vinstra megin í borðinu en hækkar síðan ef Karel færir sig til hægri.*

hnity

Notkun:	***z*** = ***hnity()***
Forskilyrði:	*ekkert*
Eftirskilyrði:	***z*** *er y hnit á staðsetningu Karel í borðinu. y hnitið er 1 ef Karel er alveg efst í borðinu en hækkar síðan ef Karel færir sig til neðar í borðið.*

stopp

Notkun:	***stopp()***
Forskilyrði:	*ekkert*
Eftirskilyrði:	*Keyrsla á forritinu stoppar.*

heil

Notkun:	***x*** = ***heil(y)***
Forskilyrði:	***y*** *er tala.*
Eftirskilyrði:	***x*** *er heiltöluhluti tölunnar* ***y****. Með öðrum orðum er* ***x*** *talan sem kemur út þegar búið er að skafa af allar tölur fyrir aftan kommu í tölunni* ***y****.*

A.4. Dæmi

Dæmi sem hægt er að spreyta sig á og lausnir við þeim er hægt að nálgast á vefsíðu kerfisins:

```
http://morpho.cs.hi.is/karel
```