LINKÖPINGS UNIVERSITET

TDP019
Projekt: Datorspråk

Bean Storm

Författare

Viktor Rösler, vikro653@student.liu.se
Jim Teräväinen, jimte145@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Vårterminen 2021
Version 1.0

11 maj 2021

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
1.0	Första versionen av rapporten färdigställd efter korrekturläsning	21-05-11
0.7	Första versionen av systemdokumentation och reflektioner färdig	21-05-10
0.6	Första versionen av allt från Variabler och tilldelning till Scope färdig	21-05-09
0.5	Installationsmanual för Bean Storm samt datatyper skriven i första version	21-05-07
0.4	Installationsmanual för Ruby skriven i första version	21-05-04
0.3	Arbetsmetodik skriven i första version	21-04-29
0.2	Inledning och Syfte skrivna i första version	21-04-28
0.1	Grundversion med titlar och grov plan	21-04-07

2 Sammanfattning

Rapporten beskriver språket Bean Storm som utvecklats som en del av kursen TDP019 vid Linköpings universitet. Språket är ett försök att skapa ett turingkomplett objektorienterat språk som använder de delar utvecklarna gillar mest från C++, Ruby och Python. Språket är utvecklat i Ruby och erbjuder många funktioner från simpel aritmetik och styrsatser till funktioner och klasser. I rapporten finns uttömmande beskrivningar och flera exempel på hur språket och dess funktioner kan användas vilket gör språket tillgängligt för de som inte spenderat mycket tid med programmering. Språket är utvecklat med hjälp av Ruby och programmet RDparse vilket var givet till utvecklarna som en del av kursen. Resultatet av utvecklingen blev ett i stora delar lyckat objektorienterat språk med endast ett par buggar och få av de planerade funktionerna kom inte med.

Innehåll

1 Revisionshistorik **1**

2 Sammanfattning **1**

3 Inledning **3**
3.1 Syfte och frågeställningar . . . 3
3.2 Arbetsmetodik och verktyg . . . 3

4 Användarhandledning **3**
4.1 Installation . . . 3
4.1.1 Operativsystem . . . 4
4.1.2 Ruby . . . 4
4.1.3 Bean Storm . . . 4
4.2 Hello World . . . 4
4.3 Grundläggande syntax . . . 5
4.4 Datatyper . . . 5
4.4.1 Booleska värden . . . 5
4.4.2 Heltal . . . 5
4.4.3 Flyttal . . . 6
4.4.4 Strängar . . . 6
4.4.5 Listor . . . 6
4.5 Variabler och tilldelning . . . 6
4.5.1 Nyckelordet auto . . . 6
4.6 Operatorer . . . 7
4.7 In- och ut-matning . . . 7
4.8 Villkorssats . . . 8
4.9 Repetitionssatser . . . 9
4.10 Funktioner . . . 9
4.11 Klasser . . . 10
4.12 Scope . . . 12

5 Systemdokumentation **12**
5.1 Lexikalisk analys . . . 13
5.2 Parsning . . . 13
5.2.1 Grammatik . . . 13
5.3 Objektträd . . . 17

6 Erfarenheter och reflektion **17**

7 Programkoden **19**
7.1 bean_storm.rb . . . 19
7.2 program.rb . . . 20
7.3 types_variables_expressions.rb . . . 25
7.4 conditions_ifs_loops.rb . . . 32
7.5 functions.rb . . . 36
7.6 classes.rb . . . 41
7.7 parser.rb . . . 47
7.8 rdparse.rb . . . 56

3 Inledning

Denna rapport beskriver ett programspråk som utvecklas av Viktor Rösler och Jim Teräväinen, härmed refererade till som utvecklarna. Arbetet utförs som en del av kursen TDP019 som läses andra terminen på programmet innovativ programmering vid Linköpings universitet. Programspråket heter Bean Storm och är ett objektorienterat, turingkomplett och generellt språk som kombinerar författarnas favoritaspekter från andra programspråk. Språket nyttjar avancerade funktioner som hårt typade variabler och måsvingar från C++ samtidigt som bekväm syntax från Ruby som till exempel att parenteser på inbyggda funktioner är valbara.

3.1 Syfte och frågeställningar

Syftet med att utveckla programspråket är att få en djupare förståelse för hur existerande programspråk fungerar, samt att ge författarna möjligheten att i framtiden lättare förstå och utveckla nya programspråk. Syftet med detta dokument är att ge läsaren en inblick i språkets funktion och hur författarna motiverat val i funktionaliteten.

Språket är från början tänkt att vara ett generellt och turingkomplett språk som kombinerar de delar av python, ruby och C++ som utvecklarna finner mest attraktiva. Attraktiviteten av en del bestäms utifrån hur naturligt konceptet känns för utvecklarna och blir därför en unik och intressant blandning. Det fanns även planer att implementera en gimmick funktion i form av att kunna tilldela enheter och storheter till värden för att enklare kunna konvertera saker så som recept eller ritningar. Inbyggt stöd för enheter implementerades aldrig då utvecklarna valde att fokusera på mer användbara funktioner.

3.2 Arbetsmetodik och verktyg

Utvecklarna har tillsammans skapat en plan för att arbetet ska följa tidsplanen och kunna lämnas in i tid till deadline given av beställaren. Planen är att snabbt diskutera vad utvecklarna förväntar sig av kursen och språket så att det är klart för alla inblandade vilken ambitionsnivå projektet bör ligga på. Nästa steg är att undersöka vilka krav beställaren har så att de kan tas i åtanke vid planeringen. Därefter planerar utvecklarna språkets struktur med BNF-notation. I nästa fas börjar utvecklarna att skapa Bean Storm samtidigt som de dokumenterar processen i form av dagbok och ändringar i BNF-notationen. Utvecklarna träffas varje måndag för att planera veckan och uppskatta antalet timmar som kommer nyttjas av andra kurser, resterande tid används till utveckling av språket. Utvecklarna har som mål att arbeta från 8:15 till 17:00 varje dag men kan skala upp eller ned om arbetet skulle behöva mer eller mindre tid.

För utvecklingen av språket används programspråket Ruby och programmet RDparse. Ruby används för logiken och för att skapa klasser som sedan kan exekveras för att ge Bean Storm den funktion som utvecklarna önskar. RDparse är ett program skrivet i Ruby som hanterar funktionen av en lexer och en parser. Det vill säga RDparse sköter uppdelningen och klassificeringen av text samt att skapa rätt objekt beroende på det som skrivits. Reglerna som RDparse följer utformas dock av utvecklarna.

4 Användarhandledning

Följande del ämnar att utbilda läsaren i användningen av Bean Storm på en grundläggande nivå. Däribland hur man installerar språket samt skriver och exekverar en enkel fil.

4.1 Installation

För att använda Bean Storm krävs det att användaren också har Ruby installerat på sin maskin. För att kunna installera Ruby behöver användaren vara administratör eller ha godkännande från administratör för maskinen. Denna del vägleder användaren igenom att installera Ruby och köra Bean Storm, samt tar upp några krav på användarens maskin för optimal funktion.

4.1.1 Operativsystem

Bean Storm är testat och utvecklat på Ubuntu 20.04.2 LTS och Windows 10 version 10.0.19042. Att användaren har något av dessa operativsystem eller likvärdigt är inget krav men utvecklarna garanterar ingen funktion på andra operativsystem eller tidigare versioner av de två ovan nämna.

4.1.2 Ruby

För att Bean Storm ska kunna köra krävs det att användaren först installerar programspråket Ruby. Språket utvecklas med Ruby 2.7.2 och utvecklarna garanterar ingen funktion om användaren inte installerar version 2.7.2 eller senare och likvärdig version av Ruby. Steg för steg instruktioner för att installera Ruby på Windows 10 och Ubuntu följer nedan.

Windows 10:

1. Navigera via en webbläsare till `https://rubyinstaller.org/downloads/`
2. Hitta den senast släppta versionen av Ruby 2.7.X i listan till vänster. Det bör stå `Ruby+Devkit 2.7.z-1 (xYY)` där 'z' kan vara '2' eller högre och 'YY' är '64' eller '86' och bör väljas utefter vilket operativsystem du använder.
3. Klicka på den version som du hittat utefter dina förutsättningar och godkänn nedladdningen av filen som då börjar.
4. När filen laddats ned navigerar du till filen på din maskin och kör den.
5. Följ instruktionerna som installationsprogrammet ger dig.
6. För att kontrollera att installationen lyckats öppna CMD och skriv `ruby -version`. Du bör få en utskrift med versionsnumret som resultat.

Ubuntu:

1. Öppna en terminal.
2. Skriv `sudo apt update` i terminalen för att se till att du får senaste versionen av Ruby.
3. Skriv `sudo apt install ruby-full` och följ sedan instruktionerna som terminalen skriver ut.
4. Nu bör Ruby vara installerat och du kan kontrollera att det gått rätt igenom att skriva `ruby --version` i terminalen.
5. Om terminalen skickar tillbaka en utskrift med versionsnummer så är du klar!

4.1.3 Bean Storm

Bean Storm är ett program som inte behöver installeras, istället kör du en fil med hjälp av Ruby. Användningen är så enkel som att köra filen `bean_storm.rb`. Följande del antar att läsaren har tillgång till källfilerna för Bean Storm.

1. Navigera till mappen där `bean_storm.rb` är sparad, se till att filen ligger i samma mapp som alla andra filer den kom med.
2. Öppna en terminal och skriv `ruby bean_storm.rb` om terminalen nu skriver ut 'Welcome to Bean Storm' så har du startat interpretatorn och kan skriva kommandon!

4.2 Hello World

Denna del ämnar att visa användaren grunderna i filinläsning och användning av Bean Storm i form av ett 'Hello World' program.

1. Börja med att skapa en ny textfil, metoden kan variera mellan operativsystem men målet är att få en fil med ett namn som liknar detta exempel: `hello.bean`
2. Öppna filen med valfri textredigerare eller utvecklingsmiljö. Tyvärr finns i nuläget inget stöd för automatisk formatering av koden.
3. Skriv följande i filen: `print "Hello world"`
4. Spara filen och navigera till mappen där du sparat `bean_storm.rb`
5. Öppna en terminal i mappen och skriv `ruby bean_storm.rb`
6. I interpretatorn som nu öppnats skriver du `load` följt av sökvägen till din `hello.bean` fil.
7. Programmet bör nu köras och terminalen skriver ut 'Hello world'!

4.3 Grundläggande syntax

När användaren vill skriva kod i Bean Storm kommer den finna att reglerna är lösa och tillåter flera olika stilval. Ett exempel på ett enkelt program kan se ut som kodexempel 1. Eftersom att varje kommando får plats på en rad var så skulle detta program kunna skrivas direkt i interpretatorn. När användaren vill skriva mer komplicerad kod får den istället skriva ned den i en fil då interpretatorn försöker räkna ut resultatet av ett kommando vid varje ny linje.

Här vill utvecklarna påpeka för användaren att Bean Storm är helt oberoende av alla blankstegstecken förutom retur, även benämnt ny linje. Användaren kan placera valfritt antal mellanslag och indragningar i koden utan att funktionen påverkas. Detta leder även till att styrtecken som `{}`(måsvingar) inte behöver placeras på någon speciell plats, de behöver endast stå i rätt ordning.

Om användaren skulle önska att lämna kommentarer eller att Bean Storm ska hoppa över att läsa en rad finns det ett kommentarstecken. Skriver användaren `#`(hashtag) hoppar Bean Storm över att läsa resten av raden från där tecknet står.

```
# skriv ut numret 4
int a = 3
++a
print a
```

Kodexempel 1: Ett exempel på Bean Storm kod som skriver ut siffran 4.

4.4 Datatyper

Bean Storm är ett hårt typat språk. Det betyder att värden inte kan sparas i vilken variabel som helst. När användaren vill spara ett värde måste de därmed identifiera av vilken typ värdet är och skapa en variabel av rätt datatyp. Detta kan hjälpa användaren att skriva felsäker kod då det förväntade resultatet måste stämma överens med det faktiska resultatet för att undvika en krasch.

4.4.1 Booleska värden

Ett booleskt värde är binärt och har därför bara 2 godkända lägen, sant och falskt. I Bean Storm skrivs dessa värden som `True` eller `False`, det som bör noteras är att begynnelsebokstaven är en versal i båda värdena. Dessa värden återfinns oftast i villkorssatser som beskrivs senare.

4.4.2 Heltal

Heltal definieras som alla tal, positiva eller negativa, som inte har några andelar. Ett exempel skulle vara att `2` är ett heltal samtidigt som `2.2` inte är det. Heltalen återfinns i enklare matematik och ofta för att räkna repetitioner i repetitionssatser.

4.4.3 Flyttal

Flyttal är som heltal men tillåter även andelar i talet. Ett exempel skulle vara att `2.2` är ett flyttal samtidigt som `2` inte är det. I Bean Storm kan ett flyttal högst ha nio decimaler.

4.4.4 Strängar

Bean Storm kan även hantera text och oavsett om det är en ensam bokstav eller en hel paragraf används strängar. För att urskilja vanlig text från kod noteras all vanlig text med `'' ""`(citattecken). Användaren kan välja mellan enkla eller dubbla citattecken men måste ange samma variant i slutet som de gjorde i början.

4.4.5 Listor

Bean Storm erbjuder även sammansatta datastrukturer som listor, där mer än ett värde kan sparas och sedan hämtas. Listor i Bean Storm kräver inte att alla objekt är av samma datatyp och är därmed löst typade, användaren kan alltså spara t.ex både heltal och strängar i samma lista. För att hämta ett värde på en viss plats i en lista används hakparenteser, utvecklarna vill även påpeka att listor är noll-indexerade och därmed har första elementet i en lista plats noll och inte ett. Ett exempel på att hämta det första värdet ur en lista hade kunnat vara: `listnamn[0]`.

4.5 Variabler och tilldelning

Bean Storm stödjer att användaren kan spara värden i variabler för att användas senare. Variabler är hårt typade och kan endast spara värden av den typ som de förväntar sig. I kodexempel 2 visas hur samtliga datatyper kan skapas samt ett par olika sätt de kan tilldelas värden. Variabler som skapas utan att bli tilldelade ett värde får ett standardvärde.

```
1  bool condition      #Booleska värden, standardvärdet är False
2  condition = True
3
4  int number = 3      #Heltal, standardvärdet är 0
5
6  float part = 2/3    #Flyttal, standardvärdet är 0.0
7
8  string msg = "Bean Storm!" #Strängar, standardvärdet är ""
9  msg = 'Storm Bean!' #Variabler kan byta innehåll!
10
11 list dev = ["Jim", 21, 1.70] #Listor, standardvärdet är []
12 dev[1] = 22         #Det går att komma åt specifika delar av listan med hakparenteser.
```

Kodexempel 2: Exempel på hur man skapar variabler av alla datatyper och hur de kan tilldelas värden.

4.5.1 Nyckelordet auto

Bean storm erbjuder även nyckelordet `auto` utöver att skapa variabler med datatypen. Detta nyckelord tillåter Bean Storm att automatiskt härleda vilken datatyp en variabel bör använda om användaren tillhandahåller ett värde. Nyckelordet `auto` fungerar för samtliga inbyggda datatyper och ersätter namnet på datatypen i syntaxen, men fungerar endast om det finns ett värde att härleda från. I kodexempel 3 demonstreras nyckelordet `auto`, både hur det kan och inte kan användas. Utvecklarna vill även påpeka att en variabel skapad med `auto` fortfarande kommer få en datatyp precis som vilken annan variabel som helst.

```
1 #auto kan användas om det finns ett värde att härleda ifrån!
2 auto heltal = 3
3 #Datatypen för dessa blir fortfarande 'int' och 'string' respektive.
4 auto text = "Hej!"
5
6 #auto fungerar inte om den inte har något att härleda ifrån!
```

```
auto flyttal
#Raden över kommer ge ett fel och avbryta programmet.
```

Kodexempel 3: Exempel på hur nyckelordet auto kan och inte kan användas.

4.6 Operatorer

Om användaren vill nyttja de variabler och värden som Bean Storm hanterar så erbjuder språket även flertalet operatorer. Med operatorerna kan användaren manipulera värden, göra matematiska uträkningar, skriva komplicerade villkor och mycket mer. I tabell 1 visas alla matematiska operatorer tillsammans med ett exempel av dess grundläggande funktionalitet. Alla operatorer exekveras i enlighet med prioriteringsreglerna om inte användaren manipulerar ordningen med parenteser `()`. För specialfallet av pre- och post-fix operatorer så vill utvecklarna påpeka att skillnaden mellan dem är när värdet manipuleras. Prefix operatorer manipulerar värdet innan det returneras i motsats till postfix operatorer som manipulerar värdet efter det returnerats.

Exempel	Resultat	Beskrivning
2 + 1	3	Addition, returnerar summan.
2 - 1	1	Subtraktion, returnerar differensen.
4 * 2	8	Multiplikation, returnerar produkten.
4 / 2	2	Division, returnerar kvoten.
5 // 2	2	Heltalsdivision, returnerar kvoten rundat nedåt.
5 % 2	1	Modulo, returnerar resten av en division.
++1	2	Prefix addition, lägger till 1 på värdet och returnerar.
`--1`	0	Prefix subtraktion, tar bort 1 från värdet och returnerar.
1++	2	Postfix addition, lägger till 1 på värdet efter det returnerats.
`1--`	0	Postfix subtraktion, tar bort 1 från värdet efter det returnerats.

Tabell 1: Tillgängliga matematiska operatorer med enkla exempel.

Språket hanterar även logiska operatorer för jämförelse och invers. Dessa operatorer kommer alltid returnera booleska värden eftersom att de evaluerar villkoret som användaren skrivit. Tabell 2 visar alla tillgängliga logiska operatorer tillsammans med enkla exempel för sanna och falska utfall. Logiska villkor används med fördel i villkorssatser och repetionssatser.

Exempel == True	Exempel == False	Beskrivning
1 == 1	1 == 2	Likhet, kontrollerar om båda sidor är lika
1 != 2	1 != 1	Olikhet, kontrollerar om sidorna är olika
!False	!True	Invers, vänder resultatet till motsatt
True && True	True && False	Och, returnerar sant om båda sidor är sanna
True || False	False || False	Eller, returnerar sant om en av sidorna är sann
1 <2	1 <1	Strikt mindre än, kontrollerar om vänsterledet är strikt mindre än högerledet
2 >1	2 >2	Strikt större än, kontrollerar om vänsterledet är strikt större än högerledet
1 <= 1	1 <= 0	Mindre än, kontrollerar om vänsterledet är mindre än högerledet
2 >= 2	2 >= 3	Större än, kontrollerar om vänsterledet är större än högerledet

Tabell 2: Tillgängliga logiska operatorer med enkla exempel.

4.7 In- och ut-matning

För att interagera med slutanvändaren av ett program kan användaren av Bean Storm utnyttja den in- och ut-matning som finns i språket. `Print` och `input` funktionerna tillåter användaren att skriva information till terminalen eller hämta information skriven i terminalen. Utmatning är enkelt och använder `print`

funktionen. Det som användaren skriver efter `print` skrivs ut i terminalen. Inmatning är mer komplicerat, `input` funktionen pausar programmet tills slutanvändaren trycker på retur. Om slutanvändaren har matat in text eller siffror innan retur trycks så kommer `input` funktionen att returnera det. I kodexempel 4 visas en representation av interpretatorn och exempel på `print` och `input`. Utvecklarna vill även påpeka att med `print` och `input` används inga parenteser `()`.

```
Welcome to Bean Storm!
How have you bean?
>> print 2
2
>> print "Print hanterar mycket!"
Print hanterar mycket!
>> print "Vad heter du?"
Vad heter du?
>> string answer = input
Jag heter Jim.
>> print answer
Jag heter Jim.
```

Kodexempel 4: Exempel på användning av in- och ut-matning i terminalen.

4.8 Villkorssats

Bean Storm erbjuder flera olika styrsatser och den mest grundläggande är villkorssatsen. Med om-satsen kan användaren exekvera kod endast om ett villkor uppfylls. Utöver det erbjuder språket även 'eller-' och 'eller om-' satser. För att använda en av satserna skriver användaren ett av nyckelorden `if, else, elif` följt av ett logiskt villkor som evalueras till sant eller falskt. Efter villkoret placeras två måsvingar (`{}`) och all kod innanför dem hör då till villkorssatsen. I kodexempel 5 finns exempel på användningen av villkorssatserna.

Om-satsen använder nyckelordet `if` och exekverar den tillhörande koden endast om villkoret evaluerar till sant.

Eller-satsen använder nyckelordet `else` och kan endast användas tillsammans med 'om-' och 'eller om-satsen'. Nyckelordet `else` placeras efter måsvingen som markerar slutet av en annan villkorssats och exekverar den tillhörande koden om ingen av de föregående satserna evaluerade till sant.

'Eller om'-satsen använder nyckelordet `elif` och kan placeras efter en om-sats eller en annan 'eller om'-sats. Satsen fungerar likadant som en om-sats förutom att i en kedja av flera kommer endast den första satsen vars villkor evaluerar till sant köras och resten hoppas över.

Alla inbyggda typer i Bean Storm har ett sanningsvärde. Standardvärdet för inbyggda typer har sanningsvärdet falskt, och alla andra värden har sanningsvärdet sant.

```
if 1.0
{ den här koden hade körts } #kom ihåg att måsvingar kan placeras fritt
else
{ men inte den här }


if (1 == 2)#parenteser kan placeras kring villkoret för tydlighet
{
    Den här koden hade inte kört eftersom 1 == 2 blir falskt
}
elif True { den här koden hade körts }
elif True { och då hoppas denna över }
else { Samma för denna }
```

Kodexempel 5: Exempel på användning av villkorssatser.

4.9 Repetitionssatser

Bean Storm erbjuder två olika repetitionssatser, `while` och `for` loopar. Repetitionssatser tillåter användaren att köra tillhörande kod flera gånger utan att behöva skriva samma kod flera gånger i rad. På samma sätt som villkorssatserna används måsvingar (`{}`) för att definiera vilken kod som tillhör repetitionssatsen. I kodexempel 6 visas praktiska exempel för båda repetitionssatserna.

Repetitionssatsen 'så länge' använder nyckelordet `while` och exekverar så länge och endast om dess villkor evaluerar till sant. Efter att den tillhörande koden har exekverats så om-evalueras villkoret och om det blir falskt hoppas den tillhörande koden över och programmet går vidare.

Repetitionssatsen 'för' använder nyckelordet `for` och exekverar så länge och endast om dess villkor evaluerar till sant vilket gör den identisk till 'så länge'-satsen. Däremot så har 'för'-satsen en inbyggd funktionalitet i form av att den kan manipulera en styrvariabel automatiskt varje repetition. Syntaxen för 'för' är speciell och är följande: `for (variabel:villkor:ändring){kod}`. `for` är nyckelordet för satsen och står alltid med. `variabel` är den variabel som kommer styra satsen, oftast definieras den direkt på plats. `villkor` är det villkor som måste evaluera till sant för att koden ska köras. `ändring` är det uttryck som kommer köras en gång efter varje repetition och bör manipulera styrvariabeln så att det inte repeterar i oändlighet.

```
print "godtar du reglerna? y/n?"
string answer = input
while (answer != y && answer != Y)
{
    print "Du får inte fortsätta utan att godta reglerna!"
    print "godtar du reglerna? y/n?"
    answer = input
    # Denna kommer köra tills användaren matar in 'y' eller 'Y'
}

for (int i = 0 : i < 10 : ++i)
{
    print i
    # Denna kommer köra 10 gånger och skriva ut talen 0-9
}
```

Kodexempel 6: Exempel på användning av repetitionssatser

4.10 Funktioner

Bean Storm inkluderar även funktionalitet för funktioner som minskar behovet av att skriva samma kod flera gånger. Funktioner i Bean Storm erbjuder ett sätt att ge ett kodblock ett namn och sedan tillkalla den koden med olika attribut när användaren behöver den. Syntaxen för funktioner är: `datatyp namn(attribut){kod}` och efter att den är definierad används den med `namn(attribut)`. Utvecklarna har valt att ha typade funktioner eftersom att det hjälper användaren att skriva felfri och väl igenomtänkt kod. Typade funktioner tvingar användaren att returnera rätt sorts värde från funktionen.

För funktioner finns även en extra datatyp kallad `void`, den kan användas för att definiera en funktion som inte ska returnera något, t.ex en funktion för utskrifter. I kodexempel 7 definieras ett par olika funktioner för att visa på funktionaliteten. Utvecklarna vill påpeka att en funktion inte behöver vara definierad tidigare i koden för att kunna användas eftersom att alla funktioner läses in först, användaren kan alltså definiera en funktion efter att de använt den i en fil. Dessutom kan funktioner överladdas så att flera funktioner med samma namn men olika attribut kan existera samtidigt.

```
int add(int a, int b)
{
    return a + b
}

int add(int a, int b, int c)# Detta är en överladdning.
```

```
{
    return add(a, b) + c
}

add(1, 2) # Kommer returnera 3

void hello() {print "Hello world"} # void funktioner kräver ingen retur.

bool test(bool a, bool b)
{
    if (bool a && bool b)
    {
        hello()
        return True
    }
    else { return False }
}
```

Kodexempel 7: Exempel på användning av funktioner

4.11 Klasser

För mer avancerade program kan Bean Storm nyttja klasser och objektorienterad programmering. För att skapa en klass i Bean Storm används nyckelorder `class` följt av ett namn och måsvingar, t.ex `class Namn {innehåll}`. Klasser stödjer variabler, funktioner och konstruktionsfunktioner så att användaren kan skapa objekt lika dem i C++. Konstruktionsfunktionerna följer syntaxen: `Klassnamn(attribut){kod}` och körs när ett objekt av klassen skapas med rätt mängd och sorts attribut. Klasser får automatiskt en grundläggande konstruktionsfunktion så att det går att skapa objekt av en klass även om inga konstruktionsfunktioner skrivits. För att skriva över den grundläggande konstruktionsfunktionen behöver användaren definiera en egen konstruktionsfunktion utan attribut.

För att skapa ett objekt av en klass använder användaren klassen som att det vore en datatyp, t.ex `Klassnamn objektnamn` hade skapat ett objekt. Om användaren vill tillkalla en konstruktionsfunktion lägger den till parenteser efter objektnamnet med attribut som matchar en konstruktionsfunktion i klassen. Utöver detta stödjer även klasserna arv, för att definiera en barnklass skrivs den enligt syntaxen: `class Barnklass < Förälder {innehåll}`. När en klass ärver av en annan klass så får den tillgång till samtliga klassvariabler och funktioner som ligger i föräldern, men inte konstruktionsfunktionerna utanför dess egen konstruktionsfunktioner. I kodexempel 8 visas det lite enkla exempel på att skapa klasser och arv samt skapa och använda objekt från dessa klasser. Utvecklarna vill påpeka att alla klassnamn ska ha en versal som begynnelsebokstav.

```
class Person
{
    int age
    string sex
    float money

    Person(int n_age, string n_sex, float n_money)
    {
        age = n_age
        sex = n_sex
        money = n_money
    }

    float check_balance()
    {
        return money
    }
    void birthday()
    {
```

```
        ++age
    }
}

class Student < Person
{
    int hp
    string program

    Student(int n_age, string n_sex, float n_money, int n_hp, string n_program)
    {
        Person(n_age, n_sex, n_money)
        hp = n_hp
        program = n_program
    }

    void csn()
    {
        money = money + 10000
    }
}

Person mandy(40, "female", 700000)

mandy.birthday()
mandy.age # Detta kommer nu returnera 41

Student bill(22, "non-binary", 40000, 80, "Innovativ programmering")
bill.csn()
bill.money # Detta kommer nu returnera 50000
bill.birthday()
bill.age # Detta kommer nu returnera 23
```

Kodexempel 8: Exempel på användning av funktioner

Klasser i Bean Storm har stöd för statiska variabler och åtkomstmodifierare. Statiska variabler är variabler som alla objekt av samma klass har gemensamt. Om ett klassobjekt ändrar värdet på en statisk variabel ändras också värdet på den medlemmen för alla andra klassobjekt av samma klass.

Åtkomstmodifierare hanterar vilka medlemmar som går att komma åt utanför klassen. Medlemmar som finns under åtkomsten `private:` är endast tillgängliga i klassen. Medlemmar under åtkomsten `protected:` går att komma åt i dess klass, och eventuella underklasser. Medlemmar som är `public:` är tillgängliga både i klassen, dess underklasser, och via punktnotation. Alla medlemmar är publika om inget annat anges, och alla konstruktorer är publika.

I kodexempel 9 visas exempel på statiska variabler samt åtkomstmodifierare

```
class Person
{

    Person(string n_address, int n_age, string n_name)
    {
        address = n_address
        age = n_age
        name = n_name
    }

private:
    string address
protected:
    int age
public:
    string name
}
```

```
18
19 class Friend < Person
20 {
21     Friend(string n_address, int n_age, string n_name)
22     {
23         Person(n_address, n_age, n_name)
24     }
25
26     int get_age() {return age}
27
28     string get_address() {return address}
29
30     static bool can_dance
31 }
32
33 Friend orkla("beanstreet", 55, "Orkla")
34 Friend bill("1stAve", 32, "Bill")
35
36 bill.can_dance = True  # Detta sätter can_dance till True för både orkla och bill
37
38 # tillåtet: name har publik åtkomst
39 orkla.name
40
41 # inte tillåtet: age är protected
42 orkla.age
43
44 # tillåtet: get_age är en publik funktion,
45 # och kan komma åt protected medlemmar i dess basklass
46 orkla.get_age()
47
48 # inte tillåtet: address är private
49 orkla.get_address()
```

Kodexempel 9: Exempel på användning statiska variabler och åtkomstmodifierare

4.12 Scope

Bean Storm använder sig av statisk variabel och funktions åtkomst vilket betyder att varje funktion, klass och styrsats endast observerar sin egen och sina föräldrars variabler och funktioner. Som exempel på detta så kan en variabel som skapas direkt i grunden av en fil kommas åt var som helst senare i filen, dessa kallas för 'globala' variabler. En variabel som skapas i den tillhörande koden för en 'om'-sats kan däremot inte kommas åt av en separat 'om'-sats. Däremot kan en 'om'-sats inuti den första 'om'-satsen komma åt alla variabler som skapats i dess överliggande styrsats. Detta gäller även för funktioner som användaren definierar. Denna hantering är viktig för inkapslingen av variabler och garanterar en mer stabil och korrekt funktion i programmet.

5 Systemdokumentation

Bean Storm använder programmet RDparse för att göra lexikalisk analys och parsa kod skriven i språket. Vid parsningen skapas ett objektträd, där varje objekt representerar en konstruktion i språket. När parsningen är klar anropas en funktion på objektet längst upp i trädet som evaluerar trädet. Evalueringen sker rekursivt, genom att varje objekt i trädet evaluerar de objekt som det består av.

Ett Bean Storm program som består av mer än en rad kod resulterar i ett objektträd med ett `Stmt_List` objekt längst upp i trädet. Ett `Stmt_List` består av, och evaluerar, alla satser i programmet. Först evalueras funktion- och klassdefinitioner, och sedan alla andra satser. Det resulterar i att funktioner och klasser kan användas var som helst i koden, oavsett var de är definierade. För att alla klassdefinitioner evalueras i samma skede måste underklasser vara definierade efter sina basklasser.

5.1 Lexikalisk analys

Den lexikaliska analysen skapar tokens från programkod. Varje token är ett eller flera tecken som utgör accepterade tecken och ord i språket. Tokeniseringen sker utifrån en rad regler, och en token skapas enligt den första regeln den uppfyller.

De första reglerna skapar tokens för `{` och `}` som representerar början, samt slut, på ett scope. Båda dessa regler äter upp blankstegstecken vid sidan om sig för att göra dem oberoende av radbrytningar. Sedan kommer regler för kommentarer (`#`), radbrytning, och blanksteg. Kommentarer och blanksteg reglerna skapar inte till några tokens, och radbrytning skapar en token som markerar radavslut. Regler för literaler, nyckelord, typer, namn, och operatorer kommer sist, och de skapar alla en token som består av samma tecken som reglerna kräver.

Varje token används sedan vid parsningen av ett program.

5.2 Parsning

Parsningen kontrollerar att uppsättningen av, och ordningen på, tokens från den lexikaliska analysen bildar korrekt syntax för språket. Parsningen sker enligt reglerna som finns i grammatiken för språket, och varje delsekvensen av tokens från sekvensen av tokens ur den lexikaliska analysen matchar på den första grammatikregeln som uppfylls.

Varje regel i grammatiken definierar en konstruktion i språket. Vid parsningen av ett program skapas objekt för de olika konstruktionerna i språket. Dessa består av varandra och utgör tillsammans objektträdet för ett program.

5.2.1 Grammatik

```
---PROGRAM---
<program> ::= <eol> <stmt_list> <eol>
            | <eol> <stmt_list>
            | <stmt_list> <eol>
            | <stmt_list>

<stmt_list> ::= <stmt> <eol> <prog_stmt_list> | <stmt>

<eol> ::= /^@eol$/

<stmt> ::= /^load$/ /\S*.bean/
         | /^print$/ <value>
         | <input>
         | <class_def>
         | <func_def>
         | <loop_stmt>
         | <if_stmt>
         | <declare_stmt>
         | <assign_stmt>
         | <value>

<input> ::= /^input$/

<declare_stmt> ::= <built_in_type> <name> = <input>
                 | <built_in_type> <name> = <value>
                 | <built_in_type> <name> = & <var>
```

```
                | <built_in_type> <name>
                | <class_name> <name> ( <func_args> )
                | <class_name> <name> ( )
                | <class_name> <name> = & <var>
                | <class_name> <name> = <func_call>
                | <class_name> <name> = <var>
                | <class_name> <name>


<built_in_type> ::= /^auto$/
                  | /^bool$/
                  | /^int$/
                  | /^float$/
                  | /^string$/
                  | /^list$/

<name> ::= /\b?!auto\b|bool\b|int\b|float\b|string\b|list\b|void\b|
               if\b|elif\b|else\b|print\b|input\b|while\b|for\b|break\b|
               continue\b|return\b|class\b|static\b)^[a-z]\w*\b/x


<assign_stmt> ::= <list_index> = <input>
                | <list_index> = <value>
                | <name> = & <input>
                | <name> = <input>
                | <name> = <value>

<value> ::= <bool> | <expr>


<expr> ::= <expr> + <term> | <expr> - <term> | <term>

<term> ::= <term> // <un_opr>
         | <term> * <un_opr>
         | <term> / <un_opr>
         | <term> % <un_opr>
         | <un_opr>

<un_opr> ::= -- <atom>
           | ++ <atom>
           | <atom> --
           | <atom> ++
           | - <atom>
           | <atom>

<atom> ::= <func_call>
         | <list_index>
         | <var>
         | <string>
         | <list>
         | <float>
```

```
         | <int>
         | ( <expr> )


<var> ::= <class_member_access> | <name>

<float> ::= /\d+\.\d+/
<int> ::= /^\d+$/
<string> ::= /"[^\"]*"/ | /'[^\']*'/
<bool> ::= /^True$/ | /^False$/
<list> ::= [ <list_nested_objects> ] | [ ]

<list_nested_objects> ::= <value> , <nested_objects> | <value>

<nested_objects> ::= <value> , <nested_objects> | <value>

<list_index> ::= <name> <index>

<index> ::= [ <expr> ] <index> | [ <expr> ]

or_cond ::= <or_cond> || <and_cond> | <and_cond>

and_cond ::= <and_cond> && <comparison> | <comparison>

<comparison> ::= <value> <rel_opr> <val>
               | ! ( <or_cond> )
               | ( <or_cond> )
               | ! <condition_value>
               | <condition_value>

<rel_opr> ::= == | != | >= | <= | > | <

<condition_value> ::= <literal>
                    | <func_call>
                    | <list_index>
                    | <var>

<literal> ::= <int> | <float> | <bool> | <string> | <list>


<if_stmt> ::= /^if$/ <or_cond> { <nested_stmt_list> } <if_rule>
            | /^if$/ <or_cond> { <nested_stmt_list> } <eol> <if_rule>
            | /^if$/ <or_cond> { <nested_stmt_list> }

<if_rule> ::= | /^elif$/ <or_cond> { <nested_stmt_list> } <if_rule>
              | /^elif$/ <or_cond> { <nested_stmt_list> } <eol> <if_rule>
              | /^elif$/ <or_cond> { <nested_stmt_list> }
              | /^else$/ { <nested_stmt_list> }


<loop_stmt> ::= /^while$/ <or_cond> { <nested_stmt_list> }
```

```
              | /^for$/ ( <declare_stmt> : <or_cond> : <expr> ) { <nested_stmt_list> }
              | /^for$/ ( <declare_stmt> : <or_cond> : <assign_stmt> ) { <nested_stmt_list> }
              | /^for$/ ( <var> : <or_cond> : <expr> ) { <nested_stmt_list> }
              | /^for$/ ( <var> : <or_cond> : <assign_stmt> ) { <nested_stmt_list> }

<nested_stmt_list> ::= <nested_stmt> <eol> <nested_stmt_list>
                     | <nested_stmt> <eol>
                     | <nested_stmt>

<nested_stmt> ::= /^break$/ | /^continue$/ | <return_stmt> | <stmt>


<func_def> ::= <func_type> <name> ( <param_list> ) { <func_stmt_list> }
             | <func_type> <name> ( ) { <func_stmt_list> }

<func_type> ::= /^void$/ | <type>

<type> ::= <built_in_type> | <class_name>

<param_list> ::= <type> <name> , <param_list> | <type> <name>

<func_stmt_list> ::= <func_stmt> <eol> <func_stmt_list>
                   | <func_stmt> <eol>
                   | <func_stmt>

<func_stmt> ::= <return_stmt> | <stmt>

<return_stmt> ::= /^return$/ <value> | /^return$/

<func_call> ::= <name> ( <func_args> )
              | <name> ( )
              | <class_name> ( <func_args> )
              | <class_name> ( )

<func_args> ::= & <var> , <func_args>
              | <value> , <func_args>
              | & <var>
              | <value>


<class_def> ::= /^class$/ <class_name> < <class_name> { <class_def_body }
              | /^class$/ <class_name> { <class_def_body> }

<class_name> ::= /\b(?!True\b|False\b)^[A-Z]\w*\b/

<class_def_body> ::= <class_member_def> <eol> <class_def_body>
                   | <class_member_def> <eol>
                   | <class_member_def>

<class_member_def> ::= <func_def>
                     | <class_constructor>
```

```
                    | <class_variable>
                    | <access_mod>

<class_constructor> ::= <class_name> ( <param_list> ) { <func_stmt_list> }
                      | <class_name> ( ) { <func_stmt_list> }

<class_variable> ::= /^static$/ <declare_stmt> | <declare_stmt>

<access_mod> ::= /^public:$/ | /^private:$/ | /^protected:$/

<class_member_access> ::= <name> . <class_member_access> | <name> . <member>

<member> ::= <func_call> | <assign_stmt> | <list_index> | <name>
```

5.3 Objektträd

Objekten som skapas av parsern består av varandra och bildar ett träd av objekt. Varje objekt i trädet hanterar evalueringen av en konstruktion i språket. När ett Bean Storm program körs evaluerar objekten i objektträdet varandra och producerar resultatet av körningen.

Exempelvis representeras en while-loop i språket av ett `While_Loop` objekt. `While_Loop` objektet innehåller ett `Binary_condition` objekt, och ett `Nested_Stmt_List` objekt. Ett `Binary_condition` objekt innehåller ett villkor, och retunerar sanningsvärdet av detta villkor när objektet evalueras. `Nested_Stmt_List` objekt är rekursiva behållare som kan lagra alla sorters satser som finns i ett scope. När ett `Nested_Stmt_List` evalueras retunerar det en array med alla satser i den rekursiva strukturen. När ett `While_Loop` evalueras, evaluerar det alla satser i sitt `Nested_Stmt_List` objekt upprepade gånger, tills dess att `Binary_condition` objektet evaluerar till falskt.

6 Erfarenheter och reflektion

Under projektets gång har svårigheten att fortsätta utvecklingen varierat grovt men aldrig känts omöjlig. Vissa delar av språket har kommit naturligt och fungerat nästan direkt, medan andra delar visat sig vara stora utmaningar utan någon klar lösning. I slutändan är språket inte färdigt till den grad som utvecklarna hade önskat men de anser ändå att arbetet som utförts på tiden allokerad till projektet har varit imponerande och välgjort.

Att börja utvecklingen var ett relativt stort steg att ta men igenom att läsa igenom tidigare projekts källkod kunde projektet börja. Efter själva starten flöt det på bra medan utvecklarna löste grundläggande implementation och det tog inte lång tid innan ensamma rader aritmetik gick att exekvera i interpretatorn. Att läsa in kod från filer gick smärtfritt men då uppstod en av projektets största farthinder, raderna smälte gärna ihop och oväntade saker hände. Utvecklarna observerade att radbrytningen som de ville skulle vara slutmarkören för varje linje inte fungerade korrekt och behövde på något sätt skapa ett token som bröt inläsningen per rad. Detta ledde till en lång process av att testa olika regex-uttryck, läsa igenom `rdparser.rb` och generellt ifrågasätta mycket av koden. I slutändan löstes problemet på ett godtyckligt sätt igenom att skapa nya token på radbrytningar som sedan tolkas som ett stopp så att inget token kan paras ihop med något från raden innan eller efter. Tyvärr så är även denna lösningen inte perfekt eftersom att det hindrar användaren från att skriva kommentarer på vissa platser utan att programmet kraschar.

Problem som radbrytningen har bidragit till att Bean Storm inte riktigt nådde potentialen utvecklarna beskrev i sitt BNF när kursen börja, men samtidigt så har BNF-dokumentet behövt utvecklas i andra riktningar när flera funktioner lagts till. Funktioner som 'each-loopar' och 'hash-listor' har tyvärr fått stanna i utvecklingsstadiet på grund av brist på tid då utvecklarna är säkra på att de hade lyckats implementera

funktionerna med nog tid. Å andra sidan har BNF-dokumentet som sagt blivit utökat med kursens gång och funktioner som `print` och `input` hade helt glömts av i planeringsstadiet och tillkom därför senare. Därtill var utvecklarna osäkra på hur långt de skulle hinna så från början var inte heller klasser planerade, något som är en stor funktionalitet nu när projektet når sitt slut.

Överlag så är utvecklarna nöjda med sin egen prestation och riktningen språket har utvecklats i under projektets gång. Trots att det finns buggar och vissa delar som kanske skulle behöva skrivas om i sin helhet för framtida utveckling så har språket en tydlig och lättföljd riktning framåt. Hade detta varit ett projekt utan tidsbegränsning så hade språket kunnat utvecklas i flera veckor till innan kreativiteten för nya och relevanta funktioner hade börjat sina. Idéerna och arbetet har varit sunt och lärdomarna som utvecklarna tar med sig från projektet är många, inte minst så har programspråk avmystifierats och känns plötsligt väldigt åtkomliga.

Ett par funktioner är utvecklarna mindre nöjda med som t.ex felhanteringen som varken är väldigt tydlig eller ger uttömmande feedback. Därtill har språket i slutändan fått något av en konstig syntax där parenteser är nödvändiga i vissa fall och valbara i andra. Detta är som konsekvens av att utvecklarna gärna ville implementera den lösa syntax som Ruby erbjuder men märkte att det blev komplicerat och krockade med funktionalitet från C++. Denna insikt kom lite sent och det gjordes en bedömning att den hårda C++ syntaxen var att föredra då den gav övertag som var svåra att lösa med Ruby liknande syntax.

7 Programkoden

7.1 bean_storm.rb

```
1
2  require './parser.rb'
3
4  class BeanStorm
5    def initialize
6      @parser = BeanStormParser.new.parser
7    end
8
9    def program
10     for arg in ARGV
11       puts "Loading #{arg}..."
12       begin
13         @parser.parse "load #{arg}"
14       rescue Exception => error
15         puts error
16       end
17     end
18
19     puts if ARGV.length > 0
20
21     puts "Welcome to Bean Storm!\nHow have you bean?"
22     while true
23       print ">> "
24       str = $stdin.gets
25       if str.match(/^\s*$/)
26         next
27       elsif str.match(/(^exit$|^quit$)/)
28         break
29       elsif str.match(/^reset$/)
30         $scopes = [Hash.new]
31         $classes = Hash.new
32         puts "The Beanstorm Interpreter Has Bean Reset."
33       else
34         begin
35             @parser.parse str
36           rescue Exception => error
37             puts error
38           end
39       end
40     end
41   end
42 end
43
44 BeanStorm.new.program
```

7.2 program.rb

```
1
2  require './types_variables_expressions.rb'
3  require './conditions_ifs_loops.rb'
4  require './functions.rb'
5  require './classes.rb'
6
7  # List of hashes of scope layers
8  # Global scope is at index 0,
9  # and the current scope is at back of the list (index -1)
10 $scopes = [Hash.new]
11
12 # Hash of Hashes with class definitions
13 $classes = Hash.new
14
15
16 ########################################
17 #              PROGRAM                 #
18 ########################################
19
20 # recursive class for storing all statements in a .bean file
21 # @stmt is the top statement
22 # @stmt_list is the rest of the statements
23 class Stmt_List
24
25   @@stmt_array = []
26
27   def initialize(stmt, stmt_list)
28     @stmt, @stmt_list = stmt, stmt_list
29
30   end
31
32   def eval
33     # evalute function- and class definitions before all other statements
34     if @stmt.is_a?(Func_Def) || @stmt.is_a?(Class_Def)
35       @stmt.eval
36     else
37       @@stmt_array << @stmt
38     end
39
40     # evaluate the rest of the statements
41     if @stmt_list.is_a?(Stmt_List)
42       return @stmt_list.eval
43     else
44       @@stmt_array << @stmt_list
45       for i in @@stmt_array
46         evaluated_stmt = i.eval
47       end
48     end
49     @@stmt_array = []
50     return evaluated_stmt # for testing
51   end
52 end
53
54 # outputs a string representation of an object
55 class Print
56   def initialize(obj)
57     @obj = obj
58   end
59
60   def eval
61     str = @obj.to_string
62     puts str
63     return str # for testing
```

```
  end
end

# take input from the user, and save it in a Data_Obj
class Input
  def get_obj(type = nil)
    input = $stdin.gets

    if type == "string"
      obj = Data_Obj.new(input, type)
    elsif input =~ /^-?\d+$/
      obj = Data_Obj.new(input.to_i, "int")
    elsif input =~  /^-?\d+\.\d+$/
      obj = Data_Obj.new(input.to_f, "float")
    elsif input =~ /^True$/
      obj = Data_Obj.new(true, "bool")
    elsif input =~ /^False$/
      obj = Data_Obj.new(false, "bool")
    else
      obj = Data_Obj.new(input, "string")
    end

    return obj
  end

  def eval(type = nil)
    return get_obj(type).eval
  end
end

# parses and runs a .bean file
class Load
  def initialize(parser,file_name)
    @parser, @file_name = parser, file_name
  end

  def eval
    file = File.new(@file_name)
    return @parser.parse file.read
  end
end



########################################
#         HELPER FUNCTIONS             #
########################################

# print error message when an error occurs
def raise_error(text)
  raise Exception.new("An Error Has Bean Found :(\n#{text}")
end

# checks if a variable's value is valid for it's type
def type_check(object, type)
  if object.class == Class_Obj
    if object.type != type
      raise_error("Error Invalid Type: #{object.name} is not a(n) #{type}")
    else
      return
    end
  end

  value = object.eval
  if value.class == Array
```

```
129     text = object.to_string
130   else
131     text = value
132   end
133
134   if type == "int" && value.class != Integer
135     raise_error("Error Invalid Type: #{text} is not an integer.")
136
137   elsif type == "float" && value.class != Integer && value.class != Float
138     raise_error("Error Invalid Type: #{text} is not a float.")
139
140   elsif type == "bool" && value != true and value != false
141     raise_error("Error Invalid Type: #{text} is not a bool.")
142
143   elsif type == "string" && value.class != String
144     raise_error("Error Invalid Type: #{text} is not a string.")
145
146   elsif type == "list" && value.class != Array
147     raise_error("Error Invalid Type: #{text} is not a list.")
148
149   elsif type == "void"
150     raise_error("Error Invalid Type: #{text} is set to void")
151   end
152
153 end
154
155 # finds the closest scope that contains [name] and returns a variable object/function definition
156 # using static scoping
157 def get_var(name, param_key = nil)
158   ($scopes.length-1).downto(0) do |i|
159     # function definitions
160     if param_key
161       if $scopes[i][name] != nil && $scopes[i][name][param_key] != nil
162         return $scopes[i][name][param_key]
163       # check class scope after class function scope
164       elsif $scopes[i]["@scope_type"] == :func
165         (i-1).downto(0) do |j|
166           if $scopes[j]["@scope_type"] == :class
167             if $scopes[j][name] != nil && $scopes[j][name][param_key] != nil
168               return $scopes[j][name][param_key]
169             else
170               break
171             end
172           end
173         end
174       end
175
176       # check global scope after class/function scope
177       if $scopes[i]["@scope_type"] == :class || $scopes[i]["@scope_type"] == :func
178         if $scopes[0][name] != nil && $scopes[0][name][param_key] != nil
179           return $scopes[0][name][param_key]
180         end
181
182         return nil
183       end
184
185     # variables
186     else
187       if $scopes[i][name] != nil
188         cls = $scopes[i][name].class
189         if cls == Data_Obj || cls == Class_Obj
190           return $scopes[i][name]
191         end
192
193       # check class scope after class function scope
```

```
194       elsif $scopes[i]["@scope_type"] == :func
195         (i-1).downto(0) do |j|
196           if $scopes[j]["@scope_type"] == :class
197             if $scopes[j][name] != nil
198               cls = $scopes[j][name].class
199               if cls == Data_Obj || cls == Class_Obj
200                 return $scopes[j][name]
201               end
202             else
203               break
204             end
205           end
206         end
207       end
208       # check global scope after class/function scope
209       if $scopes[i]["@scope_type"] == :class || $scopes[i]["@scope_type"] == :func
210         if $scopes[0][name] != nil
211           cls = $scopes[0][name].class
212           if cls == Data_Obj || cls == Class_Obj
213             return $scopes[0][name]
214           end
215         end
216         raise_error("Error: unable to evaluate #{name}.")
217       end
218     end
219   end
220
221   if param_key
222     return nil
223   else
224     raise_error("Error: unable to evaluate #{name}.")
225   end
226 end
227
228 # finds the closest scope that contains variable [name] and set it to [obj]
229 # using static scoping
230 def set_var(name, obj)
231   ($scopes.length-1).downto(0) do |i|
232     if $scopes[i][name] != nil
233       if $scopes[i][name].class == obj.class
234         $scopes[i][name] = obj
235         return
236       else
237         raise_error("Error: a #{obj.type} can't be assigned to a(n) #{$scopes[i][name].type}.")
238       end
239     # check class scope after class function scope
240     elsif $scopes[i]["@scope_type"] == :func
241       (i-1).downto(0) do |j|
242         if $scopes[j]["@scope_type"] == :class
243           if $scopes[j][name] != nil
244             if $scopes[j][name].class == obj.class
245               $scopes[j][name] = obj
246               return
247             else
248               raise_error("Error: a #{obj.type} can't be assigned to a(n) #{$scopes[i-1][name].
    type}.")
249             end
250           else
251             break
252           end
253         end
254       end
255     end
256     # check global scope after class/function scope
257     if $scopes[i]["@scope_type"] == :func || $scopes[i]["@scope_type"] == :class
```

```
258       if $scopes[0][name] != nil
259         if $scopes[0][name].class == obj.class
260           $scopes[0][name] = obj
261           return
262         else
263           raise_error("Error: a #{obj.type} can't be assigned to a(n) #{$scopes[0][name].type}.")
264         end
265       end
266       raise_error("Error: variable #{name} is not declared.")
267     end
268   end
269   raise_error("Error: #{name} is not declared.")
270 end
```

7.3 types_variables_expressions.rb

```ruby
########################################
#              TYPES                   #
########################################


class Data_Obj
  attr_accessor :value, :type

  def initialize(value, type)
    @value, @type = value, type
  end

  def to_string
    if @type == "list"
      str = "["
      for element in eval
        str += ", " if str != "["
        str += element.to_string
      end
      return str + "]"
    elsif @type == "bool"
      return "True" if @value
      return "False"
    else
      return @value.to_s
    end
  end

  def truth_value
    return false if (@type == "int" || @type == "float") && @value == 0
    return false if @type == "string" && @value == ''
    return eval if @type == "bool"
    return false if @type == "list" && @value == nil || @value == []
    return true
  end

  def eval
    if @type == "list"
      if @value == nil
        @value = []
      elsif @value.class == Nested_Objects
        @value = @value.eval
      end
    end

    return @value
  end

end

class Nested_Objects
  attr_reader :value, :next_value

  def initialize(value, next_value, pass_by_ref = false, allow_class_obj = true)
    @value, @next_value = value, next_value
    @pass_by_ref, @allow_class_obj = pass_by_ref, allow_class_obj
  end

  def to_param_key
    key = ""
    value = @value
    next_value = @next_value
    while(true)
```

```
64        key += " " if key != ""
65
66        value = value.get_obj if value.class != Data_Obj
67        key += value.type
68
69        break if !next_value
70
71        value = next_value.value
72        next_value = next_value.next_value
73      end
74
75      return key
76    end
77
78    def eval
79      list = []
80      value = @value
81      next_value = @next_value
82
83      while (true)
84        if value.class == Data_Obj
85          list << value
86        elsif value.class == Unary_Operator
87          list << Data_Obj.new(value.eval, "float")
88        else
89          obj = value.get_obj
90          if @pass_by_ref
91            list << obj
92          else
93            if obj.class == Class_Obj
94              list << Marshal.load(Marshal.dump(obj)) # deep copy Class_Obj
95            else
96              list << obj.clone
97            end
98          end
99          if list[-1].class == Class_Obj && !@allow_class_obj
100           raise_error("Error: lists only support built in types")
101         end
102       end
103
104       break if !next_value
105
106       value = next_value.value
107       next_value = next_value.next_value
108     end
109
110     return list
111   end
112 end
113
114 class List_Index
115   attr_reader :name, :index
116
117   def initialize(name, index)
118     @name, @index = name, index
119   end
120
121   def to_string
122     return get_obj.to_s
123   end
124
125   def get_obj
126     index_array = @index.eval
127     obj = get_var(@name)
128
```

```
129     for ind in index_array
130       obj = obj.eval.at(ind.eval)
131     end
132
133     return obj
134   end
135
136   def eval
137     return get_obj.eval
138   end
139 end
140
141 ########################################
142 #              VARIABLES               #
143 ########################################
144
145 # create a variable of a built-in type
146 class Declare_Variable
147
148   attr_accessor :type, :name, :value
149
150   def initialize(type, name, value = nil, value_by_ref = false)
151     @type, @name, @value, @value_by_ref = type, name, value, value_by_ref
152   end
153
154   def eval(add_to_scope = true)
155
156     # create a Data_Obj that holds the value of the variable
157     cls = @value.class
158     if @value == nil
159       # assign a default value if none was given
160       case @type
161       when "int"
162         obj = Data_Obj.new(0, @type)
163       when "float"
164         obj = Data_Obj.new(0.0, @type)
165       when "bool"
166         obj = Data_Obj.new(false, @type)
167       when "string"
168         obj = Data_Obj.new("", @type)
169       when "list"
170         obj = Data_Obj.new(nil, @type)
171       when "auto"
172         raise_error("Error: variable #{@name} is not assigned a value.")
173       end
174     elsif cls == Retrieve_Variable || cls == List_Index || cls == Member_Access
175       if @value_by_ref
176         obj = @value.get_obj
177       else
178         obj = @value.get_obj.clone
179       end
180     elsif cls == Input
181       obj = @value.get_obj(@type)
182     else
183       if @type == "auto"
184         if cls == Unary_Operator
185           obj = Data_Obj.new(@value.eval, @value.get_obj.type)
186         elsif cls == Expression
187           obj = @value.get_obj
188           obj = Data_Obj.new(obj.eval, obj.type)
189         else
190           obj = Data_Obj.new(@value.eval, @value.type)
191         end
192       else
193         obj = Data_Obj.new(@value.eval, @type)
```

```
      end
    end

    @type = obj.type if @type == "auto"
    type_check(obj, @type)

    if add_to_scope
      # check that @name isn't declared already
      if $scopes[-1][@name]
        raise_error("Error: variable #{@name} is already declared.")
      else
        $scopes[-1][@name] = obj
      end
    end

    return obj
  end
end

# assign a value/object to a variable
class Assign_Variable
  attr_reader :name, :value

  def initialize(name, value, value_by_ref = false)
    @name, @value, @value_by_ref = name, value, value_by_ref
  end

  def eval
    # assign to list index
    if @name.class == List_Index
      var_obj = @name.get_obj

      if @value.class != Data_Obj
        val_obj = @value.get_obj
      else
        val_obj = @value
      end

      if val_obj.class == Class_Obj
        raise_error("Error: lists only support built in types")
      end

      var_obj.type = val_obj.type
      if @value.class == Unary_Operator
        var_obj.value = @value.eval
      else
        var_obj.value = val_obj.value
      end
      return var_obj.value # for testing
    end

    var_obj = get_var(@name)
    # assign to variable of build in type
    if var_obj.class == Data_Obj
      if @value_by_ref
        val_obj = @value.get_obj
        set_var(@name, val_obj)
        return val_obj # for testing
      else
        var_obj.value = @value.eval
        type_check(var_obj, var_obj.type)
        return var_obj.value # for testing
      end
    # assign to variable of class type
    elsif var_obj.class == Class_Obj
```

```
259       if @value_by_ref
260         val_obj = @value.get_obj
261       else
262         val_obj = Marshal.load(Marshal.dump(@value.get_obj)) # deep copy class obj
263       end
264
265       type_check(var_obj, val_obj.type)
266       var_obj.scope = val_obj.scope
267       return val_obj # for testing
268     end
269   end
270 end
271
272 # retrieve a variable from $scopes
273 class Retrieve_Variable
274
275   attr_reader :name
276
277   def initialize(name)
278     @name = name
279   end
280
281   def to_string
282     return get_var(@name).to_string
283   end
284
285   def truth_value
286     return get_var(@name).truth_value
287   end
288
289   def get_obj
290     return get_var(@name)
291   end
292
293   def eval
294     return get_var(@name).eval
295   end
296 end
297
298 ########################################
299 #              EXPRESSIONS             #
300 ########################################
301
302 class Expression
303   def initialize(lh, op, rh)
304     @lh, @op, @rh = lh, op, rh
305   end
306
307   def to_string
308     return eval.to_s
309   end
310
311   def get_obj
312
313     # evaluate @lh into a Data_obj
314     if @lh.class == Data_Obj
315       lh = @lh
316     elsif @lh.class == Unary_Operator
317       lh = Data_Obj.new(@lh.eval, "float")
318     else
319       lh = @lh.get_obj
320     end
321
322     # evaluate @rh into a Data_Obj
323     if @rh.class == Data_Obj
```

```
324       rh = @rh
325     elsif @rh.class == Unary_Operator
326       rh = Data_Obj.new(@rh.eval, "float")
327     else
328       rh = @rh.get_obj
329     end
330
331     # evaluate the expression and return a Data_Obj
332     if (lh.type == "int" || lh.type == "float") && (rh.type == "int" || rh.type == "float")
333       if @op == '//'
334         # evaluate to a float, then truncate any decimals
335         value = instance_eval("#{lh.eval.to_f}/#{rh.eval}").truncate
336         return Data_Obj.new(value, "int")
337       else
338         # evaluate to a float, round to 9 decimals, then return as an integer if the result is a
    whole number
339         billion = 1000000000.0
340         value = (instance_eval("#{lh.eval.to_f}#{@op}#{rh.eval}")*billion).round / billion
341         if value == value.to_i
342           return Data_Obj.new(value.to_i, "int")
343         else
344           return Data_Obj.new(value, "float")
345         end
346       end
347     elsif lh.type == "string" && rh.type == "string" && @op == '+'
348       value = lh.eval + rh.eval
349       return Data_Obj.new(value,"string")
350     elsif lh.type == "list" && rh.type == "list" && @op == '+'
351       value = lh.eval + rh.eval
352       return Data_Obj.new(value,"list")
353     else
354       raise_error("Error: #{lh.type} #{@op} #{rh.type} is an invalid expression.")
355     end
356   end
357
358   def eval
359     return get_obj.eval
360   end
361
362 end
363
364 class Unary_Operator
365
366   def initialize(obj, op)
367     @obj, @op = obj, op
368   end
369
370   def to_string
371     return eval.to_s
372   end
373
374   def get_obj
375     # get/create a Data_Obj
376     cls = @obj.class
377     if cls == Retrieve_Variable || cls == List_Index || cls == Member_Access
378       @change_value = true
379       obj = @obj.get_obj
380     elsif cls == Expression
381       obj = @obj.get_obj
382     elsif cls == Func_Call
383       obj = Data_Obj.new(@obj.eval, @obj.type)
384     elsif cls == Unary_Operator
385       obj = Data_Obj.new(@obj.eval, "float")
386     else
387       obj = @obj
```

```
388     end
389
390     return obj
391   end
392
393   def eval
394     @change_value = false
395
396     obj = get_obj
397
398     # typechecking
399     if obj.type != "int" && obj.type != "float"
400       if @op == 'pre_--' || @op == 'post_--'
401         op = "--"
402       elsif @op == 'pre_++' || @op == 'post_++'
403         op = "++"
404       else
405         op = @op
406       end
407       raise_error("Error: invalid operator #{op} on #{@obj.type}.")
408     end
409
410     # evaluate
411     billion = 1000000000.0
412     if @op == 'pre_--'
413       value = obj.eval - 1
414       value = (value*billion).round / billion if value != value.to_i
415       obj.value = value if @change_value
416       return value
417     elsif @op == 'post_--'
418       value = obj.eval
419       obj.value = value - 1 if @change_value
420       obj.value = (obj.value*billion).round / billion if obj.value != obj.value.to_i
421       return value
422     elsif @op == 'pre_++'
423       value = obj.eval + 1
424       value = (value*billion).round / billion if value != value.to_i
425       obj.value = value if @change_value
426       return value
427     elsif @op == 'post_++'
428       value = obj.eval
429       obj.value = value + 1 if @change_value
430       obj.value = (obj.value*billion).round / billion if obj.value != obj.value.to_i
431       return value
432     elsif @op == '-'
433       return -obj.eval
434     end
435   end
436 end
```

7.4 conditions_ifs_loops.rb

```ruby

########################################
#             CONDITION                #
########################################

class Binary_Condition
  def initialize(lh, op, rh)
    @lh, @op, @rh = lh, op, rh
  end

  def truth_value
    begin
      if @op == "&&"
        return @lh.truth_value && @rh.truth_value
      elsif @op == "||"
        return @lh.truth_value || @rh.truth_value
      else
        lh, rh = @lh.eval, @rh.eval

        if lh.class == Array && rh.class == Array && lh.length > 0 && rh.length > 0
          return false if lh.length != rh.length

          # compare each element in two list types
          0.upto(lh.length-1) do |ind|
            return false if !Binary_Condition.new(lh[ind], @op, rh[ind]).truth_value
          end
          return true
        else
          return lh == rh if @op == "=="
          return lh != rh if @op == "!="
          return lh <= rh if @op == "<="
          return lh >= rh if @op == ">="
          return lh < rh if @op == "<"
          return lh > rh if @op == ">"
        end
      end
    rescue
      @lh = @lh.get_obj if @lh.class != Data_Obj
      @rh = @rh.get_obj if @rh.class != Data_Obj
      raise_error("Error: #{@lh.type} #{@op} #{@rh.type} is an invalid condition.")
    end
  end
end

class Not_Condition
  def initialize(condition)
    @condition = condition
  end

  def truth_value
    return !@condition.truth_value
  end
end


########################################
#              IF STATEMENT            #
########################################

class If_Stmt
  def initialize(condition, stmt_list, else_if = nil)
    @condition, @stmt_list, @else_if = condition, stmt_list, else_if
  end
```

```
64
65    def eval
66      if @condition.truth_value
67        $scopes << {"@scope_type" => :if}
68
69        # create array with new copies of each statement each iteration
70        if @stmt_list.is_a?(Nested_Stmt_List)
71          stmt_array = @stmt_list.eval
72        else
73          stmt_array = [@stmt_list.clone]
74        end
75
76        # eval each statement
77        for stmt in stmt_array
78          result = stmt.eval
79          if result == "break" or result == "continue" or result.class == Return_Stmt
80            $scopes.delete_at(-1)
81            return result
82          end
83        end
84
85        $scopes.delete_at(-1)
86      elsif @else_if
87        @else_if.eval
88      end
89    end
90  end
91
92  ########################################
93  #              LOOPS                   #
94  ########################################
95
96
97  class While_Loop
98    def initialize(condition, stmt_list)
99      @condition, @stmt_list = condition, stmt_list
100   end
101
102   def eval
103     break_loop = false
104     while @condition.truth_value
105       $scopes << {"@scope_type" => :loop}
106
107       # create array with new copies of each statement each iteration
108       if @stmt_list.is_a?(Nested_Stmt_List)
109         stmt_array = @stmt_list.eval
110       else
111         stmt_array = [@stmt_list.clone]
112       end
113
114       # eval each statement
115       for stmt in stmt_array
116         result = stmt.eval
117         if result.class == Return_Stmt
118           $scopes.delete_at(-1)
119           return result
120         elsif result == "break"
121           break_loop = true
122           break
123         elsif result == "continue"
124           break
125         end
126       end
127
128       $scopes.delete_at(-1)
```

```
129
130       break if break_loop
131     end
132   end
133 end
134
135 class For_Loop
136
137   def initialize(var,cond,expr,stmts)
138     @var, @cond, @expr, @stmts = var, cond, expr, stmts
139   end
140
141   def eval
142     break_loop = false
143
144     var = @var.eval(false) if @var.class == Declare_Variable
145
146     while true
147       if @var.class == Declare_Variable
148         $scopes << {"@scope_type" => :loop, @var.name => var}
149       else
150         $scopes << {"@scope_type" => :loop}
151       end
152
153       break if !@cond.truth_value
154
155       # create array with new copies of each statement each iteration
156       if @stmts.is_a?(Nested_Stmt_List)
157         stmt_array = @stmts.eval
158       else
159         stmt_array = [@stmts.clone]
160       end
161
162       # eval each statement
163       for stmt in stmt_array
164
165         result = stmt.eval
166         if result.is_a?(Return_Stmt)
167           $scopes.delete_at(-1)
168           return result
169         elsif result == "break"
170           break_loop = true
171           break
172         elsif result == "continue"
173           break
174         end
175       end
176
177       break if break_loop
178
179       @expr.eval
180
181       $scopes.delete_at(-1)
182     end
183     $scopes.delete_at(-1)
184   end
185 end
186
187 # recursive class for storing all statements in loops, if-statements, and functions
188 # @stmt is the top statement
189 # @stmt_list is the rest of the statements
190 class Nested_Stmt_List
191
192   attr_accessor :stmt, :stmt_list
193
```

```
194   def initialize(stmt, stmt_list)
195     @stmt, @stmt_list = stmt, stmt_list
196   end
197
198   def eval
199     stmt_array = []
200     next_stmt = @stmt
201     rem_stmts = @stmt_list
202
203     while true
204       stmt_array << next_stmt.clone
205
206       if rem_stmts.class != Nested_Stmt_List
207         stmt_array << rem_stmts.clone
208         break
209       end
210
211       next_stmt = rem_stmts.stmt
212       rem_stmts = rem_stmts.stmt_list
213     end
214
215     return stmt_array
216   end
217 end
218
219 # shell around a string used to store break and continue statements
220 class Loop_Manipulator
221   def initialize(name)
222     @name = name
223   end
224
225   def eval
226     return @name
227   end
228 end
```

7.5 functions.rb

```
########################################
#              FUNCTIONS               #
########################################


class Func_Def
  attr_reader :name

  def initialize(type, name, param, stmt_list)
    @type, @name, @param, @stmt_list = type, name, param, stmt_list
  end

  def eval(save_to_scope = true)

    func_def = [@type, @param, @stmt_list]

    # create parameter key
    if @param
      param_key = @param.to_param_key
    else
      param_key = ""
    end

    if save_to_scope
      $scopes[0][@name] = Hash.new if $scopes[0][@name] == nil

      if $scopes[0][@name][param_key] == nil
        $scopes[0][@name][param_key] = func_def
      else
        raise_error("Error: global function #{@name} with parameters #{param_key} has multiple
    definitions.")
      end

      return self # for testing
    else
      return func_def << @name << param_key  # return to Class_Def
    end

  end
end

class Func_Call

  attr_reader :name, :type

  def initialize(name, args = nil)
    @name, @args = name, args
  end

  def to_string
    return get_obj.to_string
  end

  def truth_value
    if @args
      param_key = @args.to_param_key
    else
      param_key = ""
    end

    func_def = get_func_def(param_key)

    @type = func_def[0]
```

```
63
64    return false if @type == "int" && eval == 0
65    return false if @type == "float" && eval == 0.0
66    return eval if @type == "bool"
67    return false if @type == "string" && eval == ""
68    return false if @type == "list" && eval == []
69    return false if @type == "void"
70
71    return true
72  end
73
74  # finds the closest viable function definition for the given parameters
75  # used to handle int type arguments being passed to a function taking float(s)
76  def get_func_def(param_key)
77    args_array = param_key.split
78    param_keys = []
79    param_keys << "" if param_key == ""
80
81    # generate param keys
82    for arg in args_array
83      if param_keys == []
84        param_keys << arg
85        param_keys << "float" if arg == "int"
86      else
87        nr_keys = param_keys.length
88        0.upto(nr_keys-1) do |ind|
89          param_keys[ind] += " " + arg
90          param_keys << param_keys[ind] + " " + "float" if arg == "int"
91        end
92      end
93    end
94
95    # find function definition
96    for param_key in param_keys
97      func_def = get_var(@name, param_key)
98      return func_def if func_def
99    end
100
101   raise_error("Error: function #{@name}(#{param_key}) is not defined.")
102 end
103
104 def get_obj
105   if @args
106     param_key = @args.to_param_key
107   else
108     param_key = ""
109   end
110
111   # get function definition from global function hash
112   func_def = get_func_def(param_key)
113   @type = func_def[0]
114
115   # create new scope layer
116   scope = {"@scope_type" => :func}
117
118   # temporarily remove class scope to access variables passed as arguments
119   cls_scope = $scopes.pop if $scopes[-1]["@scope_type"] == :class
120
121   # add function arguments to $scopes
122   if @args
123     args = @args
124     param_list = func_def[1].eval
125     (0..param_list.length-1).each do |i|
126       var_type = param_list[i][0]
127       name = param_list[i][1]
```

```
128
129         # get/create Data_Obj
130         cls = args.value.class
131         if cls == Retrieve_Variable || cls == List_Index || cls == Expression || cls ==
    Member_Access
132           var_obj = args.value.get_obj
133         elsif cls == Unary_Operator || cls == Func_Call
134           var_obj = Data_Obj.new(args.value.eval, var_type)
135         else
136           var_obj = args.value
137         end
138
139         args = args.next_value
140
141         # check that variable type is correct
142         type_check(var_obj, var_type)
143
144         # add argument to scope
145         if scope[name] != nil
146           raise_error("Error: function #{@name} has two parameters with the same name.")
147         else
148           scope[name] = var_obj
149         end
150
151       end
152
153     end
154
155     if cls_scope
156       $scopes << cls_scope
157     end
158
159     $scopes << scope
160
161     # get recursive list with function statements
162     stmt_list = func_def[2]
163
164     # create array with each function statement
165     if stmt_list.is_a?(Nested_Stmt_List)
166       stmt_array = stmt_list.eval
167     else
168       stmt_array = [stmt_list.clone]
169     end
170
171     # eval each function statement
172     for stmt in stmt_array
173       result = stmt.eval
174       if result.class == Return_Stmt
175         if result.value != nil
176
177           # type check return statement
178           type_check(result.value, @type)
179
180           # remove function scope layer
181           $scopes.delete_at(-1)
182
183           return result.value
184         else
185           break
186         end
187       end
188     end
189
190     # remove function scope layer
191     $scopes.delete_at(-1)
```

```
192
193     # return default value of function return type
194     return Data_Obj.new(0, @type) if @type == "int"
195     return Data_Obj.new(0.0, @type) if @type == "float"
196     return Data_Obj.new(false, @type) if @type == "bool"
197     return Data_Obj.new("", @type) if @type == "string"
198     return Data_Obj.new([], @type) if @type == "list"
199     return Data_Obj.new(nil, @type) if @type == "void"
200     return Class_Obj.new(@type, @name, nil)
201   end
202
203
204   def eval
205     get_obj.eval
206   end
207
208 end
209
210 # recursive class that holds (type, name) of function parameters
211 class Param_List
212   attr_reader :type, :name, :next_param
213
214   def initialize(type, name, next_param = nil)
215     @type, @name, @next_param = type, name, next_param
216   end
217
218   # create string with types,
219   # used for function overloading
220   def to_param_key
221     out = @type.to_s
222     next_param = @next_param.clone
223     while next_param
224       out += " " + next_param.type.to_s
225       next_param = next_param.next_param.clone
226     end
227     return out
228   end
229
230   # returns list of lists with parameter data, [[type, name], ...]
231   def eval
232     out = [[@type, @name]]
233     next_param = @next_param
234     while next_param
235       out << [next_param.type, next_param.name]
236       next_param = next_param.next_param
237     end
238     return out
239   end
240 end
241
242 # shell around an object to be returned from a Func_Call
243 class Return_Stmt
244   attr_reader :value
245
246   def initialize(value = nil)
247     @value = value
248   end
249
250   def eval
251     if @value
252       cls = @value.class
253       if cls == Func_Call
254         @value = Data_Obj.new(@value.eval, @value.type)
255       elsif cls == Unary_Operator
256         @value = Data_Obj.new(@value.eval, "float")
```

```
257       elsif cls != Data_Obj
258         @value = @value.get_obj
259       end
260     end
261
262     return self
263   end
264
265 end
```

7.6 classes.rb

```ruby
require 'set'

########################################
#              CLASSES                 #
########################################

class Class_Def
  attr_reader :variables

  def initialize(name, stmts, parent = nil)
    @name, @stmts, @parent = name, stmts, parent
    @access_mod = :public
  end

  def eval

    scope = {:public => {"@scope_type" => :class},
             :protected => {"@scope_type" => :class},
             :private => {"@scope_type" => :class}}

    if @stmts.is_a?(Nested_Stmt_List)
      stmt_array = @stmts.eval
    else
      stmt_array = [@stmts.clone]
    end

    # evaluate constructors & function definitions,
    # and save every other stmt for when a class obj is created
    class_body_stmts = []
    constructors = []
    declared_vars = Set.new
    for stmt in stmt_array
      cls = stmt.class
      if cls == Declare_Variable || cls == Decl_Class_Var
        declared_vars << stmt.name
        class_body_stmts << stmt
      elsif cls == Static_Variable
        obj = stmt.eval
        scope[:private][stmt.name] = obj
        if @access_mod == :protected || @access_mod == :public
          scope[:protected][stmt.name] = obj
        end
        scope[:public][stmt.name] = obj if @access_mod == :public
      elsif cls == Func_Def || cls == Class_Constructor
        if cls == Class_Constructor && stmt.name != @name
          error_txt = "Error: constructor #{stmt.name} \
                       doesn't match class name #{@name}."
          raise_error(error_txt)
        elsif cls == Func_Def && stmt.name == @name
          raise_error("Error: constructor #{stmt.name} has a return type.")
        end

        func_def = stmt.eval(false)
        func_name = func_def[3]
        param_key = func_def[4]

        if cls == Func_Def
          if scope[:private][func_name] == nil
            scope[:private][func_name] = Hash.new
            if @access_mod == :protected || @access_mod == :public
              scope[:protected][func_name] = Hash.new
            end
            scope[:public][func_name] = Hash.new if @access_mod == :public
```

```
64          elsif scope[:private][func_name][param_key] != nil
65            error_txt = "Error: function #{func_name}(#{param_key})\
66                         in class #{@name} is defined twice."
67            raise_error(error_txt)
68          end
69
70          func_def = func_def[0..-3]
71          scope[:private][func_name][param_key] = func_def
72          if @access_mod == :protected || @access_mod == :public
73            scope[:protected][func_name][param_key] = func_def
74          end
75          scope[:public][func_name][param_key] = func_def if @access_mod == :public
76        elsif cls == Class_Constructor
77          constructors << func_def
78        end
79      elsif cls == String
80        @access_mod = :public if stmt == "public:"
81        @access_mod = :protected if stmt == "protected:"
82        @access_mod = :private if stmt == "private:"
83        class_body_stmts << stmt
84      end
85    end
86
87    # add parent's functions, static variable, and member variables
88    if @parent
89      parent = $classes[@parent]
90      if parent == nil
91        raise_error("Error: parent class #{@parent} is not defined.")
92      else
93        parent_funcs = parent[0][:protected]
94        parent_funcs.each do |key,value|
95          next if key == @name || key == "@scope_type"
96
97          # add static variables
98          if value.class != Hash
99            if scope[:private][key] == nil
100             scope[:private][key] = value
101             if @access_mod == :protected || @access_mod == :public
102               scope[:protected][key] = value
103             end
104             scope[:public][key] = value if @access_mod == :public
105           end
106         # add functions
107         else
108           if scope[:private][key] == nil
109             scope[:private][key] = Hash.new
110             if @access_mod == :protected || @access_mod == :public
111               scope[:protected][key] = Hash.new
112             end
113             scope[:public][key] = Hash.new if @access_mod == :public
114           end
115
116           value.each do |param_list,func_def|
117             if scope[:private][key][param_list] == nil
118               scope[:private][key][param_list] = func_def
119               if @access_mod == :protected || @access_mod == :public
120                 scope[:protected][key][param_list] = func_def
121               end
122               scope[:public][key][param_list] = func_def if @access_mod == :public
123             end
124           end
125         end
126       end
127
128       # add variables
```

```
129         @access_mod = :public
130         parent_stmts = parent[1]
131         for stmt in parent_stmts
132           if stmt.class == String
133             @access_mod = :public if stmt == "public:"
134             @access_mod = :protected if stmt == "protected:"
135             @access_mod = :private if stmt == "private:"
136           else
137             next if @access_mod == :private || declared_vars.include?(stmt.name)
138           end
139
140           class_body_stmts << stmt
141         end
142
143         # add constructors
144         constructors = constructors + parent[2]
145       end
146     end
147
148     $classes[@name] = [scope, class_body_stmts, constructors]
149
150   end
151 end
152
153 # shell around a Func_Def used to verify that a constructor have the same name as the class
154 class Class_Constructor
155   attr_reader :name
156
157   def initialize(name, param_list, stmt_list)
158     @name = name
159     @func_def = Func_Def.new("void", name, param_list, stmt_list)
160   end
161
162   def eval(add_to_scope)
163     @func_def.eval(add_to_scope)
164   end
165 end
166
167 # shell around a Declare_Variable used to separate static class variables from other class
      variables
168 class Static_Variable
169   attr_reader :name
170
171   def initialize(declare_stmt)
172     @declare_stmt = declare_stmt
173     @name = declare_stmt.name
174   end
175
176   def eval
177     return @declare_stmt.eval
178   end
179 end
180
181 class Decl_Class_Var
182   attr_reader :name
183
184   def initialize(type, name, constructor_args = nil, value = nil, value_by_ref = false)
185     @type, @name, @constructor_args, @value = type, name, constructor_args, value
186   end
187
188   def eval(add_to_scope = true)
189     if $classes[@type] == nil
190       raise_error("Error: type #{@type} is not defined.")
191     end
192
```

```
193     if @value
194       if @value_by_ref
195         value = @value.get_obj
196       else
197         value = Marshal.load(Marshal.dump(@value.get_obj))
198       end
199     else
200       value = Class_Obj.new(@type, @name, @constructor_args)
201     end
202
203     if add_to_scope
204       if $scopes[-1][@name]
205         raise_error("Error: variable #{@name} is already declared.")
206       else
207         $scopes[-1][@name] = value
208       end
209     else
210       return value
211     end
212   end
213 end
214
215 class Class_Obj
216   attr_reader :type, :name
217   attr_accessor :scope
218
219   def initialize(type, name, constructor_args)
220     @type, @name = type, name
221
222
223     # create @scope for Class_Obj variables w/o a @scope_type key,
224     # so that Class_Obj @scope and class functions + static variables,
225     # can be added to $scopes seperately
226     @scope = {:public => Hash.new, :protected => Hash.new, :private => Hash.new}
227
228     # add class functions and static variables to global $scopes
229     $scopes << $classes[type][0][:private]
230     # add Class_Obj @scope to global $scopes
231     $scopes << @scope[:private]
232
233     # add variables to Class_Obj @scope
234     access_mod = :public
235     for stmt in $classes[type][1]
236       cls = stmt.class
237       if cls == Declare_Variable || cls == Decl_Class_Var
238         if @scope[:private][stmt.name] == nil
239           obj = stmt.eval(false)
240           @scope[:private][stmt.name] = obj
241           if access_mod == :protected || access_mod == :public
242             @scope[:protected][stmt.name] = obj
243           end
244           @scope[:public][stmt.name] = obj if access_mod == :public
245         else
246           error_txt = "Error: member variable #{stmt.name} \
247                        in class #{type} is defined twice."
248           raise_error(error_txt)
249         end
250       else
251         access_mod = :public if stmt == "public:"
252         access_mod = :protected if stmt == "protected:"
253         access_mod = :private if stmt == "private:"
254       end
255     end
256
257
```

```
258     if $classes[type][2] != []
259       # merge @scope and class functions + static variables,
260       # and add it to global $scopes
261       $scopes << $scopes.pop.merge($scopes.pop)
262
263       # add constructors to global $scopes
264       for constructor in $classes[type][2]
265         if $scopes[-1][constructor[3]] == nil
266           $scopes[-1][constructor[3]] = Hash.new
267         end
268         $scopes[-1][constructor[3]][constructor[4]] = constructor[0..2]
269       end
270
271       # run constructor
272       if constructor_args == ""
273         Func_Call.new(@type, nil).eval
274       elsif constructor_args
275         Func_Call.new(@type, constructor_args).eval
276       end
277
278     else
279       $scopes.delete_at(-1)
280     end
281
282     # set correct @scope_type for @scope
283     @scope[:private]["@scope_type"] = :class
284     @scope[:protected]["@scope_type"] = :class
285     @scope[:public]["@scope_type"] = :class
286
287
288     $scopes.delete_at(-1)
289   end
290
291
292 end
293
294
295 class Member_Access
296   attr_reader :name, :member
297
298   def initialize(name, member)
299     @name, @member = name, member
300   end
301
302   def to_string
303     return get_obj.to_string
304   end
305
306   def access_member(member_func)
307     member = @member.clone
308     name = @name.clone
309
310     obj = get_var(name)
311     type = obj.type
312     while (member.class == Member_Access)
313       name = member.name
314       obj = obj.scope[:public][name]
315       member = member.member
316     end
317
318     if member.class == Func_Call && $classes[type][0][:public][obj.name] != nil
319       $scopes << obj.scope[:private].merge($classes[type][0][:private])
320     elsif member.class == Assign_Variable
321       scope = obj.scope[:public].merge($classes[type][0][:public])
322       scope["@scope_type"] = :assign_member_access
```

```
323       $scopes << scope
324     else
325       $scopes << obj.scope[:public].merge($classes[type][0][:public])
326     end
327
328     if member_func == :eval
329       value = member.eval
330     elsif member_func == :get_obj
331       value = member.get_obj
332     elsif member_func == :truth_value
333       value = member.truth_value
334     end
335
336     $scopes.delete_at(-1)
337
338
339     return value
340   end
341
342   def truth_value
343     return access_member(:truth_value)
344   end
345
346   def get_obj
347     return access_member(:get_obj)
348   end
349
350   def eval
351     return access_member(:eval)
352   end
353 end
```

7.7 parser.rb

```
1
2  require 'logger'
3  require './program.rb'
4  require './rdparser.rb'
5
6
7  class BeanStormParser
8
9    attr_reader :parser
10
11   def initialize
12     @parser = Parser.new("Bean Storm") do |parser|
13
14       token(/\s*{\s*/) {|_| '{'} # { scope start
15       token(/\s*}/) {|_| '}'}    # } scope end
16
17       token(/^\#.*\s*/)   # comment
18
19       token(/\s*(\#.*)?\n+\s*/) {|_| "@eol"}  # end of line
20
21       token(/\s+/)   # whitespace
22
23       token(/\d+\.\d+/) {|t| t}  # float literal
24
25       token(/"[^\"]*"/)  {|t| t}   # "string" literal
26       token(/'[^\']*'/)  {|t| t}   # 'string' literal
27
28       token(/(^True$|^False$)/) {|t| t}   # bool literal
29
30       token(/(^public:$|^protected:$|^private:$)/)  {|t| t}  # access modifiers
31
32       token(/(^load$|^print$|^input$|^if$|^elif$|^else$)/) {|t| t} # keywords
33       token(/(^while$|^for$|^break$|^continue$)/)          {|t| t} # keywords
34       token(/(^return$|^class$|^static$)/)                 {|t| t} # keywords
35
36       token(/(^auto$|^int$|^float$|^bool$)/)   {|t| t} # types
37       token(/(^string$|^list$|^hash$|^void$)/) {|t| t} # types
38
39       token(/[^(load)].*\.bean/) {|t| t}  # file name
40
41       token(/\w+/) {|t| t} # types, keywords, names
42
43       token(/(\+\+|--|\/\/|&&|\|\|)/){|t| t}  # ++ -- // && || operators
44       token(/(==|!=|>=|<=|>>|<<)/)  {|t| t}  # == != > < >= <= >> << operators
45
46       # + - * / % ( ) { } [ ] = , . ! < > : & single symbol tokens
47       token(/(\+|-|\*|\/|%|\(|\)|\{|\})/) {|t| t}
48       token(/(\[|\]|=|,|\.|!|<|>|:|&)/) {|t| t}
49
50       # invalid syntax error
51       token(/.*/) {|t| raise_error("Error: Invalid syntax #{t}")}
52
53
54       ########################################
55       #              PROGRAM                 #
56       ########################################
57
58       start :program do
59         match(:eol, :stmt_list, :eol) {|_,sl,_| sl.eval}
60         match(:eol, :stmt_list) {|_,sl| sl.eval}
61         match(:stmt_list, :eol) {|sl,_| sl.eval}
62         match(:stmt_list) {|sl| sl.eval}
63       end
```

```
64
65      rule :stmt_list do
66        match(:stmt, :eol, :stmt_list) {|stmt,_,sl| Stmt_List.new(stmt,sl)}
67        match(:stmt)
68      end
69
70      rule :eol do
71        match(/^@eol$/)
72      end
73
74      rule :stmt do
75        match(/^load$/, /\S*.bean/) {|_,fn| Load.new(parser,fn)}
76        match(/^print$/, :value) {|_,val| Print.new(val)}
77        match(:input)
78        match(:class_def)
79        match(:func_def)
80        match(:loop_stmt)
81        match(:if_stmt)
82        match(:declare_stmt)
83        match(:assign_stmt)
84        match(:value)
85      end
86
87      rule :input do
88        match(/^input$/) {|_| Input.new}
89      end
90
91      ########################################
92      #              VARIABLES               #
93      ########################################
94
95      rule :declare_stmt do
96        # built in type variables
97        match(:built_in_type, :name, '=', :input) do
98          |type,name,_,inp| Declare_Variable.new(type,name,inp)
99        end
100       match(:built_in_type, :name, '=', :value) do
101         |type,name,_,val| Declare_Variable.new(type,name,val)
102       end
103       match(:built_in_type, :name, '=', '&', :var) do
104         |type,name,_,_,var| Declare_Variable.new(type,name,var,true)
105       end
106       match(:built_in_type, :name) {|type,name| Declare_Variable.new(type,name)}
107
108       # class type variables
109       match(:class_name, :name, '(', :func_args, ')') do
110         |type,name,_,args,_| Decl_Class_Var.new(type,name,args)
111       end
112       match(:class_name, :name, '(', ')') do
113         |type,name,_,_| Decl_Class_Var.new(type,name,"")
114       end
115       match(:class_name, :name, '=', '&', :var) do
116         |type,name,_,_,var| Decl_Class_Var.new(type,name,nil,var,true)
117       end
118       match(:class_name, :name, '=', :func_call) do
119         |type,name,_,func| Decl_Class_Var.new(type,name,nil,func)
120       end
121       match(:class_name, :name, '=', :var) do
122         |type,name,_,var| Decl_Class_Var.new(type,name,nil,var)
123       end
124       match(:class_name, :name) {|type,name| Decl_Class_Var.new(type,name)}
125     end
126
127     rule :built_in_type do
128       match(/^auto$/)
```

```
129         match(/^bool$/)
130         match(/^int$/)
131         match(/^float$/)
132         match(/^string$/)
133         match(/^list$/)
134       end
135
136       rule :name do
137         match(/\b(?!auto\b|bool\b|int\b|float\b|string\b|list\b|void\b|
138                    if\b|elif\b|else\b|print\b|input\b|while\b|for\b|break\b|
139                    continue\b|return\b|class\b|static\b)^[a-z]\w*\b/x)
140       end
141
142       rule :assign_stmt do
143         # assign to list index
144         match(:list_index, '=', :input) {|li,_,inp| Assign_Variable.new(li,inp)}
145         match(:list_index, '=', :value) {|li,_,val| Assign_Variable.new(li,val)}
146
147         # assign to variable
148         match(:name, '=', '&', :var) do
149           |name,_,_,var| Assign_Variable.new(name,var,true)
150         end
151         match(:name, '=', :input) {|name,_,inp| Assign_Variable.new(name,inp)}
152         match(:name, '=', :value) {|name,_,val| Assign_Variable.new(name,val)}
153       end
154
155       rule :value do
156         match(:bool)
157         match(:expr)
158       end
159
160
161       ########################################
162       #              EXPRESSION              #
163       ########################################
164
165       rule :expr do
166         match(:expr, '+', :term) {|expr,op,term| Expression.new(expr,op,term)}
167         match(:expr, '-', :term) {|expr,op,term| Expression.new(expr,op,term)}
168         match(:term)
169       end
170
171       rule :term do
172         match(:term, '//', :un_opr) {|term,op,uo| Expression.new(term,op,uo)}
173         match(:term, '*', :un_opr)  {|term,op,uo| Expression.new(term,op,uo)}
174         match(:term, '/', :un_opr)  {|term,op,uo| Expression.new(term,op,uo)}
175         match(:term, '%', :un_opr)  {|term,op,uo| Expression.new(term,op,uo)}
176         match(:un_opr)
177       end
178
179       rule :un_opr do
180         match('--', :atom) {|_,atom| Unary_Operator.new(atom,"pre_--")}
181         match('++', :atom) {|_,atom| Unary_Operator.new(atom,"pre_++")}
182         match(:atom, '--') {|atom,_| Unary_Operator.new(atom,"post_--")}
183         match(:atom, '++') {|atom,_| Unary_Operator.new(atom,"post_++")}
184         match('-', :atom)  {|op,atom| Unary_Operator.new(atom,op)}
185         match(:atom)
186       end
187
188       rule :atom do
189         match(:func_call)
190         match(:list_index)
191         match(:var)
192         match(:string)
193         match(:list)
```

```
194         match(:float)
195         match(:int)
196         match('(', :expr, ')') {|_,expr,_| expr}
197       end
198
199       rule :var do
200         match(:class_member_access)
201         match(:name) {|name| Retrieve_Variable.new(name)}
202       end
203
204
205       ########################################
206       #               TYPES                  #
207       ########################################
208
209       # round floats to 9 decimals
210       rule :float do
211         match(/\d+\.\d+/) do |num|
212           billion = 1000000000.0
213           Data_Obj.new((num.to_f*billion).round / billion, "float")
214         end
215       end
216
217       rule :int do
218         match(/^\d+$/) {|num| Data_Obj.new(num.to_i, "int")}
219       end
220
221       rule :string do
222         match(/"[^\"]*"/) {|str| Data_Obj.new(str.to_s[1..-2], "string")}
223         match(/'[^\']*'/) {|str| Data_Obj.new(str.to_s[1..-2], "string")}
224       end
225
226       rule :bool do
227         match(/^True$/)  {|_| Data_Obj.new(true, "bool")}
228         match(/^False$/) {|_| Data_Obj.new(false, "bool")}
229       end
230
231       rule :list do
232         match("[", :list_nested_objects, "]") {|_,lno,_| Data_Obj.new(lno, "list")}
233         match("[", "]") {|_,_| Data_Obj.new(nil, "list")}
234       end
235
236       rule :list_nested_objects do
237         match(:value, ",", :nested_objects) do
238           |val,_,no| Nested_Objects.new(val,no,false,false)
239         end
240         match(:value) {|val| Nested_Objects.new(val,nil,false,false)}
241       end
242
243       rule :nested_objects do
244         match(:value, ",", :nested_objects) {|val,_,no| Nested_Objects.new(val,no)}
245         match(:value) {|val| Nested_Objects.new(val, nil)}
246       end
247
248       rule :list_index do
249         match(:name, :index) {|name,ind| List_Index.new(name,ind)}
250       end
251
252       rule :index do
253         match( "[", :expr, "]", :index) {|_,expr,_,ind| Nested_Objects.new(expr,ind)}
254         match("[", :expr, "]") {|_,expr,_| Nested_Objects.new(expr,nil)}
255       end
256
257
258       ########################################
```

```
259       #                 CONDITION              #
260       ##########################################
261
262       rule :or_cond do
263         match(:or_cond, '||', :and_cond)  do
264           |oc,op,ac| Binary_Condition.new(oc,op,ac)
265         end
266         match(:and_cond)
267       end
268
269       rule :and_cond do
270         match(:and_cond, '&&', :comparison) do
271           |ac,op,c| Binary_Condition.new(ac,op,c)
272         end
273         match(:comparison)
274       end
275
276       rule :comparison do
277         match(:value, :rel_opr, :value) {|lh,op,rh| Binary_Condition.new(lh,op,rh)}
278         match('!', '(', :or_cond, ')') {|_,_,oc,_| Not_Condition.new(oc)}
279         match('(', :or_cond, ')') {|_,oc,_| oc}
280         match('!', :condition_value) {|_,cv| Not_Condition.new(cv)}
281         match(:condition_value)
282       end
283
284       rule :rel_opr do
285         match('==')
286         match('!=')
287         match('>=')
288         match('<=')
289         match('>')
290         match('<')
291       end
292
293       rule :condition_value do
294         match(:literal)
295         match(:func_call)
296         match(:list_index)
297         match(:var)
298       end
299
300       rule :literal do
301         match(:int)
302         match(:float)
303         match(:bool)
304         match(:string)
305         match(:list)
306       end
307
308       ##########################################
309       #               IF STATEMENT             #
310       ##########################################
311
312       rule :if_stmt do
313         match(/^if$/, :or_cond, '{', :nested_stmt_list, '}', :if_rule) do
314           |_,oc,_,nsl,_,ir| If_Stmt.new(oc,nsl,ir)
315         end
316
317         match(/^if$/, :or_cond, '{', :nested_stmt_list, '}', :eol, :if_rule) do
318           |_,oc,_,nsl,_,_,ir| If_Stmt.new(oc,nsl,ir)
319         end
320
321         match(/^if$/, :or_cond, '{', :nested_stmt_list, '}') do
322           |_,oc,_,nsl,_| If_Stmt.new(oc,nsl)
323         end
```

```
324       end
325
326       rule :if_rule do
327         match(/^elif$/, :or_cond, '{', :nested_stmt_list, '}', :if_rule) do
328           |_,oc,_,nsl,_,ir| If_Stmt.new(oc,nsl,ir)
329         end
330
331         match(/^elif$/, :or_cond, '{', :nested_stmt_list, '}', :eol, :if_rule) do
332           |_,oc,_,nsl,_,_,ir| If_Stmt.new(oc,nsl,ir)
333         end
334
335         match(/^elif$/, :or_cond, '{', :nested_stmt_list, '}') do
336           |_,oc,_,nsl,_| If_Stmt.new(oc,nsl)
337         end
338
339         match(/^else$/, '{', :nested_stmt_list, '}') do
340           |_,_,nsl,_| If_Stmt.new(Data_Obj.new(true, "bool"),nsl)
341         end
342       end
343
344
345       ########################################
346       #               LOOPS                  #
347       ########################################
348
349       rule :loop_stmt do
350         match(/^while$/, :or_cond, '{', :nested_stmt_list, '}') do
351           |_,oc,_,nsl,_| While_Loop.new(oc,nsl)
352         end
353
354         match(/^for$/, '(', :declare_stmt, ':', :or_cond, ':', :expr, ')',
355               '{', :nested_stmt_list, '}') do |_,_,var,_,cond,_,expr,_,_,nsl,_|
356           For_Loop.new(var,cond,expr,nsl)
357         end
358
359         match(/^for$/, '(', :declare_stmt, ':', :or_cond, ':', :assign_stmt, ')',
360               '{', :nested_stmt_list, '}') do |_,_,var,_,cond,_,expr,_,_,nsl,_|
361           For_Loop.new(var,cond,expr,nsl)
362         end
363
364         match(/^for$/, '(', :var, ':', :or_cond, ':', :expr, ')',
365               '{', :nested_stmt_list, '}') do |_,_,var,_,cond,_,expr,_,_,nsl,_|
366           For_Loop.new(var,cond,expr,nsl)
367         end
368
369         match(/^for$/, '(', :var, ':', :or_cond, ':', :assign_stmt, ')',
370               '{', :nested_stmt_list, '}') do |_,_,var,_,cond,_,expr,_,_,nsl,_|
371           For_Loop.new(var,cond,expr,nsl)
372         end
373       end
374
375       rule :nested_stmt_list do
376         match(:nested_stmt, :eol, :nested_stmt_list) do
377           |ns,_,nsl| Nested_Stmt_List.new(ns,nsl)
378         end
379         match(:nested_stmt, :eol) {|ns,_| ns}
380         match(:nested_stmt)
381       end
382
383       rule :nested_stmt do
384         match(/^break$/)    {|str| Loop_Manipulator.new(str)}
385         match(/^continue$/) {|str| Loop_Manipulator.new(str)}
386         match(:return_stmt)
387         match(:stmt)
388       end
```

```
389
390
391       ########################################
392       #               FUNCTION               #
393       ########################################
394
395       rule :func_def do
396         match(:func_type, :name, "(", :param_list, ")", "{", :func_stmt_list, "}") do
397           |type,name,_,pl,_,_,fsl,_| Func_Def.new(type,name,pl,fsl)
398         end
399         match(:func_type, :name, "(", ")", "{", :func_stmt_list, "}") do
400           |type,name,_,_,_,fsl,_| Func_Def.new(type,name,nil,fsl)
401         end
402       end
403
404       rule :func_type do
405         match(/^void$/)
406         match(:type)
407       end
408
409       rule :type do
410         match(:built_in_type)
411         match(:class_name)
412       end
413
414       rule :param_list do
415         match(:type, :name, ',', :param_list) do
416           |type,name,_,pl| Param_List.new(type,name,pl)
417         end
418         match(:type, :name) {|type,name| Param_List.new(type,name)}
419       end
420
421       rule :func_stmt_list do
422         match(:func_stmt, :eol, :func_stmt_list) do
423           |stmt,_,sl| Nested_Stmt_List.new(stmt,sl)
424         end
425         match(:func_stmt, :eol)
426         match(:func_stmt)
427       end
428
429       rule :func_stmt do
430         match(:return_stmt)
431         match(:stmt)
432       end
433
434       rule :return_stmt do
435         match(/^return$/, :value) {|_,val| Return_Stmt.new(val)}
436         match(/^return$/) {|_| Return_Stmt.new()}
437       end
438
439       rule :func_call do
440         # global/class functions
441         match(:name, '(', :func_args, ')') do
442           |name,_,args,_| Func_Call.new(name,args)
443         end
444         match(:name, '(', ')')   {|name,_,_| Func_Call.new(name)}
445
446         # constructors
447         match(:class_name, '(', :func_args, ')') do
448           |name,_,args,_| Func_Call.new(name,args)
449         end
450         match(:class_name, '(', ')') {|name,_,_| Func_Call.new(name)}
451       end
452
453       rule :func_args do
```

```
454         match('&', :var, ',', :func_args) do
455           |_,var,_,args| Nested_Objects.new(var,args,true)
456         end
457         match(:value, ',', :func_args) {|val,_,args| Nested_Objects.new(val,args)}
458         match('&', :var) {|_,var| Nested_Objects.new(var,nil,true)}
459         match(:value) {|val| Nested_Objects.new(val,nil)}
460       end
461
462
463       ########################################
464       #              CLASSES                 #
465       ########################################
466
467       rule :class_def do
468         match(/^class$/, :class_name, '<', :class_name, '{', :class_def_body, '}') do
469           |_,name,_,par,_,body,_| Class_Def.new(name,body,par)
470         end
471         match(/^class$/, :class_name, '{', :class_def_body, '}') do
472           |_,name,_,body,_| Class_Def.new(name,body)
473         end
474       end
475
476       rule :class_name do
477         match(/\b(?!True\b|False\b)^[A-Z]\w*\b/)
478       end
479
480       rule :class_def_body do
481         match(:class_member_def, :eol, :class_def_body) do
482           |memb,_,body| Nested_Stmt_List.new(memb,body)
483         end
484         match(:class_member_def, :eol) {|memb,_| memb}
485         match(:class_member_def)
486       end
487
488       rule :class_member_def do
489         match(:func_def)
490         match(:class_constructor)
491         match(:class_variable)
492         match(:access_mod)
493       end
494
495       rule :class_constructor do
496         match(:class_name, "(", :param_list, ")", "{", :func_stmt_list, "}") do
497           |name,_,pl,_,_,fsl,_| Class_Constructor.new(name,pl,fsl)
498       end
499         match(:class_name, "(", ")", "{", :func_stmt_list, "}") do
500           |name,_,_,_,fsl,_| Class_Constructor.new(name,nil,fsl)
501         end
502       end
503
504       rule :class_variable do
505         match(/^static$/, :declare_stmt) {|_,ds| Static_Variable.new(ds)}
506         match(:declare_stmt)
507       end
508
509       rule :access_mod do
510         match(/^public:$/)
511         match(/^private:$/)
512         match(/^protected:$/)
513       end
514
515       rule :class_member_access do
516         match(:name, '.', :class_member_access) do
517           |name,_,memb| Member_Access.new(name,memb)
518         end
```

```
519         match(:name, '.', :member) {|name,_,memb| Member_Access.new(name,memb)}
520       end
521
522       rule :member do
523         match(:func_call)
524         match(:assign_stmt)
525         match(:list_index)
526         match(:name) {|name| Retrieve_Variable.new(name)}
527       end
528     end
529   end
530 end
```

7.8 rdparse.rb

```
1
2  require './program.rb'
3
4  class Rule
5
6    Match = Struct.new :pattern, :block
7
8    def initialize(name, parser)
9      # The name of the expressions this rule matches
10     @logger=Logger.new(STDOUT)
11     @name = name
12     # We need the parser to recursively parse sub-expressions
13     # occurring within the pattern of the match objects associated with this rule
14     @parser = parser
15     @matches = []
16     # Left-recursive matches, which in the first two cases
17     @lrmatches = []
18   end
19
20   def match(*pattern, &block)
21     match = Match.new(pattern, block)
22     # If the pattern is left-recursive, then add it to the left-recursive set
23     if pattern[0] == @name
24       pattern.shift
25       @lrmatches << match
26     else
27       @matches << match
28     end
29   end
30
31   def parse
32     # Try non-left-recursive matches first, to avoid infinite recursion
33     match_result = try_matches(@matches)
34     return nil unless match_result
35     loop do
36       result = try_matches(@lrmatches, match_result)
37       return match_result unless result
38       match_result = result
39     end
40   end
41
42   private
43
44   # Try out all matching patterns of this rule
45   def try_matches(matches, pre_result = nil)
46     match_result = nil
47     # Begin at the current position in the input string of the parser
48     start = @parser.pos
49     matches.each do |match|
50       # pre_result is a previously available result from evaluating expressions
51       result = pre_result ? [pre_result] : []
52
53       # We iterate through the parts of the pattern, which may be
54       # [:expr,'*',:term]
55       match.pattern.each_with_index do |token,index|
56
57         # If this "token" is a compound term, add the result of
58         # parsing it to the "result" array
59
60         if @parser.rules[token]
61           result << @parser.rules[token].parse
62           unless result.last
63             result = nil
```

```
64            break
65          end
66        else
67          # Otherwise, we consume the token as part of applying this rule
68          nt = @parser.expect(token)
69          if nt
70            result << nt
71            if @lrmatches.include?(match.pattern) then
72              pattern=[@name]+match.pattern
73            else
74              pattern=match.pattern
75            end
76          else
77            result = nil
78            break
79          end
80        end
81      end
82      if result
83        if match.block
84          match_result = match.block.call(*result)
85        else
86          match_result = result[0]
87        end
88        break
89      else
90        # If this rule did not match the current token list, move
91        # back to the scan position of the last match
92        @parser.pos = start
93      end
94    end
95
96    return match_result
97  end
98end
99
100
101class Parser
102
103  attr_accessor :pos
104  attr_reader :rules,:string
105  class ParseError < RuntimeError; end
106  def initialize(language_name, &block)
107    @logger=Logger.new(STDOUT)
108    @lex_tokens = []
109    @rules = {}
110    @start = nil
111    @language_name=language_name
112    instance_eval(&block)
113  end
114
115  # Tokenize the string into small pieces
116  def tokenize(string)
117    @tokens = []
118    @string=string.clone
119    until string.empty?
120      # Unless any of the valid tokens of our language are the prefix of
121      # 'string', we fail with an exception
122      raise ParseError, "unable to lex '#{string}" unless @lex_tokens.any? do |tok|
123        match = tok.pattern.match(string)
124        # The regular expression of a token has matched the beginning of 'string'
125        if match
126          #@logger.debug("Token #{match[0]} consumed")
127          # Also, evaluate this expression by using the block
128          # associated with the token
```

```
129           @tokens << tok.block.call(match.to_s) if tok.block
130           # consume the match and proceed with the rest of the string
131           string = match.post_match
132           true
133         else
134           # this token pattern did not match, try the next
135           false
136         end # if
137       end # raise
138     end # until
139   end
140
141   def parse(string)
142     # First, split the string according to the "token" instructions given to Parser
143     tokenize(string)
144     # Now, @tokens contains all tokens that are to be parsed.
145
146     # These variables are used to match if the total number of tokens
147     # are consumed by the parser
148     @pos = 0
149     @max_pos = 0
150     @expected = []
151     # Parse (and evaluate) the tokens received
152     result = @start.parse
153     # If there are unparsed extra tokens, signal error
154     if @pos != @tokens.size
155       raise_error("Error: unable to parse statement.")
156     end
157     return result
158   end
159
160   def next_token
161     @pos += 1
162     return @tokens[@pos - 1]
163   end
164
165   # Return the next token in the queue
166   def expect(tok)
167     t = next_token
168     if @pos - 1 > @max_pos
169       @max_pos = @pos - 1
170       @expected = []
171     end
172     return t if tok === t
173     @expected << tok if @max_pos == @pos - 1 && !@expected.include?(tok)
174     return nil
175   end
176
177   def to_s
178     "Parser for #{@language_name}"
179   end
180
181   private
182
183   LexToken = Struct.new(:pattern, :block)
184
185   def token(pattern, &block)
186     @lex_tokens << LexToken.new(Regexp.new('\\A' + pattern.source), block)
187   end
188
189   def start(name, &block)
190     rule(name, &block)
191     @start = @rules[name]
192   end
193
```

```
194   def rule(name,&block)
195     @current_rule = Rule.new(name, self)
196     @rules[name] = @current_rule
197     instance_eval &block
198     @current_rule = nil
199   end
200
201   def match(*pattern, &block)
202     @current_rule.send(:match,*pattern,&block)
203   end
204
205 end
```