SNL Final Report

by
James Lin
Alex Liu
Andre Paiva
Daniel Maxson

December 17 2014

Contents

1 Introduction **1**

1.1 Introduction 1

1.2 Motivation 1

1.3 Why call it Stage (null) language? 1

2 Language Tutorial **2**

2.1 Compilation Walkthrough 2

2.1.1 Compiler Setup 2

2.1.2 Compile SNL to Java 2

2.1.3 Compile SNL to Java Executable 2

2.2 Stages 3

2.3 Recipes 3

2.4 Variables 3

2.5 Lists 3

2.6 Control Flow 4

2.7 Input/Output 4

3 Language Reference Manual **5**

3.1 Lexical Elements 5

3.1.1 Comments 5

3.1.2 Identifiers 5

3.1.3 Keywords 5

3.1.4 Literals 5

3.1.5 Operators 6

3.1.6 Seperators 6

3.1.7 White Space 6

3.2 Data Types 7

3.2.1 Typing 7

3.2.2 Built-in Data Types 7

3.3 Expressions and Operators 7

3.4 Recipes 8

3.4.1 Recipe Definitions . . . 8
3.4.2 Calling Recipes . . . 8
3.5 Program Structure and Scope . . . 9
3.5.1 Program Structure . . . 9
3.5.2 Stages . . . 9
3.5.3 Scope . . . 9

4 Project Plan **10**
4.1 Project Processes . . . 10
4.1.1 Planning . . . 10
4.1.2 Specification . . . 10
4.1.3 Development . . . 10
4.1.4 Testing . . . 10
4.2 Style Guide . . . 11
4.3 Team Responsibilities . . . 11
4.4 Project Timeline . . . 12
4.5 Development Environment . . . 12
4.6 Project Log . . . 12

5 Architectural Design **13**
5.1 Overview . . . 13
5.2 Scanning . . . 13
5.3 Parsing . . . 14
5.4 Semantic Analysis . . . 14
5.4.1 Java Generation . . . 14

6 Testing **15**
6.1 Overview . . . 15
6.2 Test Suite File Listings . . . 16
6.2.1 Expression Tests . . . 16
6.2.2 Java Tests . . . 16
6.2.3 Program Tests . . . 17
6.2.4 Statement Tests . . . 17
6.2.5 Failing Tests . . . 17
6.3 Example Tests . . . 17
6.3.1 Factorial . . . 17
6.3.2 Fibonacci . . . 18

7 Lessons Learned **20**
7.1 James Lin . . . 20
7.2 Alex Liu . . . 20

7.3 Daniel Maxson 21
7.4 Andre Paiva 22

A Project Log **23**

B Full Source Code **27**
B.1 Parser 27
B.2 Scanner 30
B.3 AST 32
B.4 SAST 34
B.5 Semantic Analyzer 35
B.6 Code Generator 41
B.7 Compiler 45
B.8 SNL Script 46
B.9 Java Backend 46
B.10 Test Script 54
B.11 Makefile 57
B.12 Tests 58

Chapter 1

Introduction

1.1 Introduction

SNL is a language designed to model role-playing game scenarios based on the structure of state machines. It allows users to programmatically re-create sequential scenarios as "stories." Its syntax aims to be simple so that adolescents can easily understand how to use it. Where possible, SNL uses intuitive keywords in place of symbols, since symbols commonly known in the CS community can be confusing to those who are unfamiliar with programming. This will encourage children to learn how to write code in a controlled, fun environment while allowing them to exercise their creativity.

1.2 Motivation

Since our program will essentially model state diagrams, a "story" can be expressed in terms of a series of stages with transitions in between them in SNL. Recipes, the SNL version of functions, help with modularization. Despite its simplification of syntax and features, SNL is still robust enough to write programs at the same level as other common programming languages. These programs can range from algorithms like calculating GCD or factorial to more creative outlets like multi-ending chapter books or RPGs. A further application of SNL could be the generation of computerized surveys and studies within the social sciences.

1.3 Why call it Stage (null) language?

The core structure of the language of movement through a sequence of stages, which pretty obviously explains the first word. This is a programming language, which pretty obviously explains the third word. So why null? Well, the word "null" gets hated on so much in the computer science community that we decided we wanted to give it a positive spin. Plus, it allows us to have a funny acronym that reminds people of a funny TV show.

Chapter 2

Language Tutorial

2.1 Compilation Walkthrough

2.1.1 Compiler Setup

To compile the compiler, utilize the “make” command. This will create the SNL compiler “snlc”.

```
$ make
```

2.1.2 Compile SNL to Java

To compile SNL code to Java code, run the SNL compiler with the “-j” flag on the SNL file and the Java files will appear in the same level directory. Use the flag “–output_path <path>” to designate a specific path for the Java files. This can then be manually compiled with “javac” given that the file SNLObject.java is in the same level as “snlc”.

```
$ ./snlc -j program_name.snl
```

2.1.3 Compile SNL to Java Executable

To compile SNL code to a Java executable, run the SNL compilation script on the SNL file. You can then run this executable using Java.

```
$ ./snl program_name.snl
```

To run program, enter: java program_name

```
$ java program_name
```

2.2 Stages

The starting stage is the point at which an SNL program begins running. Subsequent stages are accessed by the call “next <stage name>”. Stages can be called any number of times and are terminated with the keyword “done”.

```
start stage1:
    next stage2
done
stage2:
done
```

2.3 Recipes

Recipes are essentially functions in our language. They can be defined and called by the user, and have the same stage flow as the top level of the program. Recipes are called with the format “do <recipe> to <arg1>, <arg2>, ... <argn>”. Recipes can also return a value using the “return” keyword. Recipes can be called any number of times and are terminated with the keyword “done”.

```
recipe inc to number:
    start recipe_stage:
         return number + 1
     done
done
```

2.4 Variables

Variables do not have an explicitly assigned type in SNL, which utilizes dynamic typing similar to Python. Below is an example of a variable assignment using the “is” assignment keyword:

```
number is 24
str is ‘‘this is a string variable’’
```

2.5 Lists

A list can also be stored as a variable. Lists are not restricted to a single type and are mutable. Lists have several additional operations and features outlined in the reference manual. These are all performed in the same way that recipes are called. Below is an example of a list being assigned to a variable:

```
lst is [1,2,4.5,‘‘element’’, 5>=2]
```

2.6 Control Flow

In order to keep the language intuitive for non-programmers, control flow has been reduced to if- and if-else statements. The conditionals can be modified using logical operators “=”, “and”, “or”, and “not”. All subsequent statements in the if- or if-else statement must be placed inside the parentheses.

```
if num < 0 or str = ‘‘hello’’
(return 42)
```

2.7 Input/Output

Receiving input in the form of strings is done using the keyword “input”. Printing to standard output is done through the library function “show”.

```
do show to ‘‘hello world!’’
person_to_greet is input
do show to ‘‘hello ’’ + person_to_greet
```

Chapter 3

Language Reference Manual

3.1 Lexical Elements

3.1.1 Comments

All comments are single-line and denoted by the # character. Any content to the right of the # will be ignored.

3.1.2 Identifiers

Identifiers are sequences of characters used for naming variables, functions, and stages. All characters must be alphanumeric or the underscore character. The first character must be an alphabetic character.

3.1.3 Keywords

if	else	not
and	or	do
to	start	next
is	local	true
false	return	recipe
done	input	of

3.1.4 Literals

There are several literals in SNL. These include integer literals, float literals, boolean literals, string literals, and list literals.

Integer Literals
An integer literal is a sequence of digits. All digits are taken to be decimal. 12 is an example of an integer constant.

Float Literals
A float literal consists of a decimal point, and either an integer part or a fraction part or both. 5.0

and 5. and .5 are all valid floating constants.

Boolean Literals
A boolean literals is either ‘true’ or ‘false’.

String Literals
A string literal is a sequence of chars. These are sequences of characters surrounded by double quotes. Two examples of string literals are “hello” and “world”.

List Literals
A list literal is a sequence of literals that have been placed between square brackets ‘[ ]’ and separated by commas ‘,’. Lists can contain one or more types and are mutable. [1,2,3,4] and [1,2,true,“peggy”] are both examples of lists.

3.1.5 Operators

An operator is a special token that performs an operation on two operands. More information about these are provided in the Expressions and Operations section (4).

3.1.6 Seperators

A separator separates tokens. These are not seen as tokens themselves, but rather break our language into discrete pieces.

White Space
White space is the general purpose separator in SNL. More information is provided in the White Space section (2.7).

Comma
The comma is a separator, specifically in the context of creating lists (and their elements) and also for parameters passed to a function which is being called.

Colon
The colon is a separator in the context of starting a new stage or recipe. The separator will be placed right after the name of the stage or after the recipe declaration.

3.1.7 White Space

Spaces
Spaces are used to separate tokens within a single line outside of the creation of list and the first line of a stage.

Newline

Newlines are used to separate statements from one another. There is only one

3.2 Data Types

3.2.1 Typing

Variables in SNL are dynamically typed, similar to those in Python or Perl. Variables are implicitly assigned a type depending on the value assigned to it. You can find more information about these constants in the section about Literals (2.4).

3.2.2 Built-in Data Types

Type	Description
int	A series of digits
float	A series of digits with a single '.'
bool	Boolean values of true or false
string	A sequence of characters within " "
list	A sequence of items enclosed by [ ]

3.3 Expressions and Operators

Operator	Use
+	Addition, string concatenation
-	Subtraction
	Multipication
/	Division
=	Equals
!=	Not equals
and	Conjunction
or	Disjunction
not	Negation
<	Less than
<=	Less than or equals
>	Greater than
=>	Greater than or equals
( )	Grouping expressions/statements
is	Assignment
of	Access element from list

3.4 Recipes

3.4.1 Recipe Definitions

A recipe is set of stages with a separate global scope. If there are any arguments passed into the recipe, the 'to' keyword must come before the comma-separated list of arguments. The 'return' keyword will return at most one item back to the stage from which it was initially called.
An example of a recipe built using multiple stages:

```
1  start example_program:
2      lst is [3, 4, 5, 6]
3      do inc_list to lst
4      show lst
5  done
6
7  recipe inc_list to my_list:  # declaration of recipe
8
9      start start_inc_list:
10         length is do get_len to my_list  # calling a recipe
11         index is 0
12         next loop_start
13     done
14
15     loop_start:
16         if index < length
17         (next s_list_modifier)
18         else (return my_list)  # returning out of our recipe
19     done
20
21     s_list_modifier:
22         index of my_list is index of my_list + 1
23         index is index + 1
24         next loop_start
25     done
26
27 done
```

Listing 3.1: Recipe Example

3.4.2 Calling Recipes

The keywords 'do' and 'to' mark recipe calls, and the comma is used to separate function arguments. For example:

```
do foo to bar, baz
```

When there are no arguments to a recipe, 'to' must be omitted such as:

```
do foo
```

3.5 Program Structure and Scope

3.5.1 Program Structure

Each program must be written within one source file and are a combination of a single Universe along with Stages and Recipes. These can each be defined anywhere within the file.

3.5.2 Stages

A Stage will consist of a series of statements. The starting Stage for each recipe or program will be specified by the the 'start' keyword. Next will come the name of the Stage followed by a colon. For all Stages outside of the starting Stage of a recipe or program, only the name of the Stage and the colon should be used.

Within a Stage, the 'next' keyword will designate the following Stage to jump to. These will control the movement of the Character between different Stages, particularly by utilizing conditional statements to vary between different next Stages.

3.5.3 Scope

Global Scope
All variables defined either in the Universe or a Stage are by default part of the global scope and can be accessed and modified from any of the other stages within the program.

Scope within a Stage
To declare a variable at a Stage scope you will use the reserved keyword 'local' followed by the variable name. For example:

```
local colour_of_ball is "blue"
```

Scope within a Recipe
A recipe does not have any access to the Universe scope but will only have access to any items passed in or declared within this recipe. Users must be careful to remember which recipe they are declaring variables in at each stage.

Chapter 4

Project Plan

4.1 Project Processes

4.1.1 Planning

We decided to meet every Tuesday night at 11 pm in order to plan and work on the SNL compiler. This was a good time for all of us, but also proved to be one of the few times during the week our team could set aside to work on the compiler.

4.1.2 Specification

Our proposal and LRM were key for identifying the required tools and features of SNL. These were particularly helpful because we had a strong vision for our language from the beginning. The LRM was particularly helpful and provided the baseline for all of the different components that ended up comprising our language. There were obviously on-the-fly additions to the language as we saw fit in the late nights making our compiler.

4.1.3 Development

Our general process for implementing features was to plan out the feature, then write a test in SNL for the feature (using a .snl file and a corresponding .out file), and finally implement the feature. This was incredibly useful and a process that we largely picked up from our awesome Language Guru.

4.1.4 Testing

At the early stages of development, we tested our AST for the correctness of its expressions and statements as well as overall program structure. These were done by printing out the AST and comparing it with expected output. When code generation and compilation were implemented, we wrote many tests to match output of executables to ensure correctness. Through this process, we were able to catch several major bugs and and apply fixes. We implemented a script to perform

large scale testing and facilitate easy addition of new tests to be performed each time changes were made to ensure changes would not break existing code.

4.2 Style Guide

We followed rules for style:

- No more than 80 characters per line of code
- Automatic Tuareg and Omlet style formatting in Emacs and Vim, respectively
- SNL is written with standard Python formatting, using lowercase letters and underscores for names when possible
- Java syntax was used for backend Java whenever possible
- Modularization was prioritized through intensive re-factoring of source code

4.3 Team Responsibilities

Our team chemistry took a bit of time to fully take form, but came together nicely in the end. The parts that each team member worked on are listed below. We used Github for version control and sharing the same source code. The project log will be included in the appendix.

Language Guru: Alex Liu
scanner, parser, AST, analyzer, testing suite (OCaml, Python)
System Architect: Daniel Maxson
SAST, analyzer, codegen (OCaml)
Testing and Validation: Andre Paiva
tests (SNL, Python)
Project Manager: James Lin
Java backend, tests, codegen (Java, SNL, OCaml)

4.4 Project Timeline

Date	Task Finished
9/24	Project Proposal Done
10/27	LRM Done
11/1	Scanner and Parser Done
11/17	Testing Suite Constructed
12/10	SAST / Analyzer Done
12/12	Java Object Abstraction Done
12/14	Core Compiler and Code Generation Done
12/15	Library Functions Completed
12/16	Presentation and Final Report

4.5 Development Environment

Our entire team was effectively using Mac OS X due to the Windows 7 teammate's computer not functioning correctly. Our OCaml was version 4.02.1; our Python was version 2.7.6; our Java was Java 6. We used git on Github for version control. For text editing, we used a combination of Vim, Emacs, and Sublime Text. We used Bash and Python scripts along with Makefiles for automation.

4.6 Project Log

Located in Appendix A

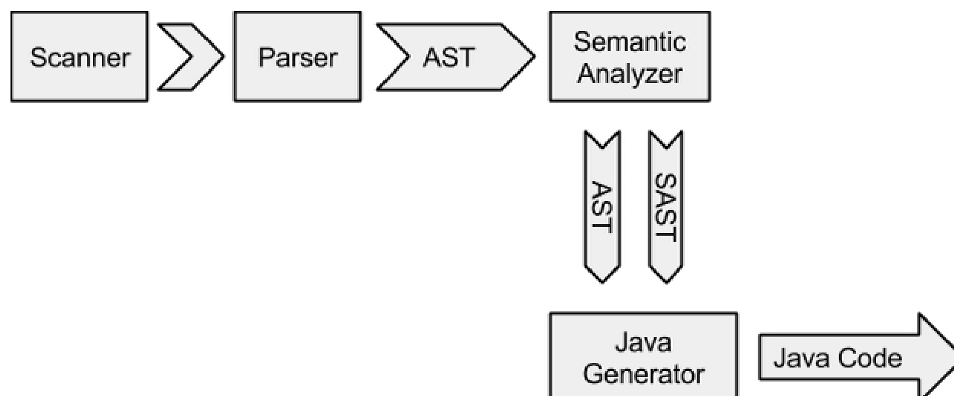
Chapter 5

Architectural Design

5.1 Overview

The SNL compiler consists of the major components seen in most compiler designs: scanner, parser, semantic analyzer and code generator. From the diagram below you can see the order of these processes.

Figure 5.1: Process Flow of SNL Compiler



The scanning and parsing was primarily done by Alex, the semantic analysis done by Alex and Daniel, the Java code generation done by Daniel and James while the Java implementation was done by James. Andre was responsible for testing all components of the compiler.

5.2 Scanning

The SNL scanner, written using ocamllex. takes all the input from the a supplied .snl file and tokenizes it. In this stage we remove all whitespace (except for newlines) and comments and throw exceptions for any illegal characters. The scanner also removes certain newlines to produce a token stream that can be described by an unambiguous grammar in the parsing phase.

5.3 Parsing

The parser. written using ocamlyacc, takes in the tokens generated during the scanning stage and produces an abstract syntax tree (AST) from an unambiguous grammar. This process catches any syntax errors and populates the tree with stages, recipes, and their respective components.

5.4 Semantic Analysis

After the AST is produced the semantic analyzer walks through the tree annotating each stage, recipe, expression and statement creating the SAST. Errors in variable scope are checked but there is minimal type checking since our language is dynamically typed.

After building the SAST, the analyzer runs through recipe and stage names to determine if any of them are duplicated. The analyzer then checks to make sure there is exactly one starting stage. It also builds and traverses a directed graph of the stages, as linked by 'next' statements, so it can warn if there are unreachable stages or return errors if 'next' calls a stage that does not exist. Recipe calls are checked to ensure that the recipe is either a valid library recipe or defined in the program and that the appropriate number of arguments is passed in.

5.4.1 Java Generation

The final part of our compiler runs through the checked SAST and prints out the appropriate Java code. There is one Java file for the main set of stages and an additional Java file for each of the recipes. The library functions are built directly into the code generation. This Java code is then compiled using the snl compile script.

Chapter 6

Testing

6.1 Overview

Each test was stored as a combination of two files in the tests directory: the SNL program and the expected result. Our team wrote wrote a Python script that would attempt to reproduce the expected output based on the .snl file fed in to it and compare this output with the .out file.

We separated these tests into five different folders. For the files in expr, program, and stmt, our script would produce the AST output. The failing folder contained some tests with simple errors to make sure that our compiler failed properly on certain syntax errors. Finally, the java directory was reserved for testing the correctness of our compiled Java code. Here, we sought to validate every feature of our language (operators, library functions, etc.) while also trying our best to catch subtler loose ends (processing comments interspersed with newlines, naming stages and recipes after Java keywords).

The AST testing was done by Alex, and the Java tests were written mainly by Andre.

6.2 Test Suite File Listings

6.2.1 Expression Tests

add.snl
and.snl
assign1.snl
assign2.snl
assign3.snl
bool1.snl
bool2.snl
call1.snl
call2.snl
call3.snl
call4.snl
div.snl
equal.snl
float1.snl
float2.snl
float3.snl
geq.snl
gt.snl
id1.snl
id2.snl
id3.snl
id4.snl
input.snl
int.snl
leq.snl
lt.snl
math1.snl
math2.snl
math3.snl
math4.snl
mult.snl
negate.snl
neq.snl
next.snl
not.snl
or.snl
paren-int.snl
return.snl
seq1.snl
seq2.snl
seq3.snl
seq4.snl
string1.snl
string2.snl
string3.snl
string4.snl
sub.snl

6.2.2 Java Tests

add1.snl
add2.snl
add3.snl
add4.snl
add5.snl
add6.snl
add7.snl
append2.snl
append.snl
bool1.snl
bool2.snl
comment1.snl
factorial2.snl
factorial.snl
fib2.snl
fib.snl
float1.snl
func1.snl
func2.snl
func3.snl
gcd.snl
global1.snl
global2.snl
hello_world.snl
id1.snl
if1.snl
if2.snl
if3.snl
if4.snl
if5.snl
insert.snl
int1.snl
length.snl
list1.snl
list2.snl
list3.snl
list4.snl
name.snl
neg1.snl
number_to_word2.snl
number_to_word.snl
ops2.snl
ops.snl
perform1.snl
remove.snl
stmt1.snl
string1.snl
sub1.snl
sub2.snl
sub3.snl
sub4.snl
var2.snl
var3.snl
var4.snl
while1.snl
whitespace.snl

6.2.3 Program Tests

program1.snl
recipe1.snl
stage1.snl
stage2.snl

6.2.4 Statement Tests

if1.snl
if2.snl
newline2.snl
if3.snl
newline1.snl

6.2.5 Failing Tests

invalid_next.snl
missing_done.snl
missing_endquote.snl

6.3 Example Tests

6.3.1 Factorial

```
start init:
    num is 5
    total is 1
    next stage1
done

stage1:
    if num = 0
    (next last)

    total is total*num
    num is num-1
    next stage1
done

last:
    do show to total
done
```

Listing 6.1: Factorial SNL Source Code: factorial.snl

```

import java.util.Scanner;
public class factorial2{
  private static Scanner input = new Scanner(System.in);

  public static void main(String args[]){
    s_init();
  }
  private static void s_init(){
```

```
    num= new SNLObject(5, "int");

    total= new SNLObject(1, "int");

    s_stage1();
    return;
  }

  private static void s_stage1(){
    if(num.eq( new SNLObject(0, "int")).getBool())
      {s_last();
        return;
      }
      else{}

        total= total.mult(num);

      num= num.sub( new SNLObject(1, "int"));

      s_stage1();
      return;
    }
    private static void s_last(){
      System.out.println(total);
    }

    private static SNLObject total;
    private static SNLObject num;
  }
```

Listing 6.2: Factorial Compiled Java Code: factorial.java

6.3.2 Fibonacci

```
recipe fib to x:
  start fib:
    if x < 2
      (return 1)
    else
      (return (do fib to x -1) + (do fib to x -2))
  done
done

start main:
  do show to do fib to 0
  do show to ""
  do show to do fib to 1
  do show to ""
  do show to do fib to 2
  do show to ""
  do show to do fib to 3
  do show to ""
  do show to do fib to 4
  do show to ""
  do show to do fib to 5
  do show to ""
  do show to do fib to 10
done
```

Listing 6.3: Fibonacci SNL Source Code: fib.snl

```java
import java.util.Scanner;

public class fib{
  private static Scanner input = new Scanner(System.in);

  public static void main(String args[]){
    s_main();
  }
  private static void s_main(){
    System.out.println(new Recipe_fib().perform( new SNLObject(0)));
    System.out.println( new SNLObject(""));
    System.out.println(new Recipe_fib().perform( new SNLObject(1)));
    System.out.println( new SNLObject(""));
    System.out.println(new Recipe_fib().perform( new SNLObject(2)));
    System.out.println( new SNLObject(""));
    System.out.println(new Recipe_fib().perform( new SNLObject(3)));
    System.out.println( new SNLObject(""));
    System.out.println(new Recipe_fib().perform( new SNLObject(4)));
    System.out.println( new SNLObject(""));
    System.out.println(new Recipe_fib().perform( new SNLObject(5)));
    System.out.println( new SNLObject(""));
    System.out.println(new Recipe_fib().perform( new SNLObject(10)));
  }
}
```

Listing 6.4: Fibonacci Compiled Java Code: fib.java

```java
import java.util.Scanner;
public class Recipe_fib{
  private SNLObject ret;
  private Scanner input = new Scanner(System.in);
  public Recipe_fib(){}

  public SNLObject perform(SNLObject x_arg){
    x = new SNLObject(x_arg);
    s_fib();
    return ret;
  }

  private void s_fib(){
    if(x.lt( new SNLObject(2)).getBool())
      {ret =  new SNLObject(1);
        return;
      }
      else{ret = new Recipe_fib().perform(x.sub( new SNLObject(1))).add(new Recipe_fib().
    perform(x.sub( new SNLObject(2))));
        return;
      }
    }
    private SNLObject x;
  }
```

Listing 6.5: Recipe For Fibonacci Compiled Java Code: Recipe_fib.java

Chapter 7

Lessons Learned

7.1 James Lin

Undertaking this project was incredibly fruitful for me on many levels. Though I have experience with project management, I have never done this with a programming project of this scale. I was very humbled by this experience, because I realized how familiarity with the project and its components is key to proper management. At the beginning of the course and the project, I did not really understand the technical aspects of the project. As a result, my leadership ability was severely crippled and I was unable to make wise decisions. I am determined not to be this kind of manager in the future and do my best to invest in understanding the ins and outs of my future projects.

Two other important areas of growth were the utilization of git and a testing-first mentality. Again, I only had minimal exposure to both of these in my software engineering career thus far. These projects pushed me and challenged me to use these tools to their maximum potential. I was also able to pick up handy additions to my shell and other development environments.

My suggestion for future groups would be to pair program as early as possible in the development process, especially if there are disparities in technical understanding. Our productivity increased greatly when we had two sets of eyes and two brains tackling a single problem at a time. It also made merge conflicts much easier to deal with. I will be sure to use this in my future career. Thank you to Professor Edwards for this excellent class!

7.2 Alex Liu

If you're the language guru, debugging the scanner and parser is a lot easier when you understand how shift-reduce parsing works. As the internet helpfully suggests, use the parser output (run verb|ocamlyacc -v filename.mly) and set the `OCAMLRUNPARAM` environment variable to p (run export OCAMLRUNPARAM='p' in bash) so you can see all the stages the parser is passing through on what input and figure out why the dreaded exception Parsing.Parse_error pops up.

Also, the language your group envisions may actually be ambiguous if you can only look one character ahead, and there may be no way to easily and crisply describe it for your parser. One way to address this problem is to come up with a standardized form of the language that is more easily described with an unambiguous context-free grammar. Then, use the scanner to strip away all the characters or sequences that are nice to allow in the language but difficult to parse, and run the parser on this transformed language.

OCaml is actually a nice language and the compiler is very helpful in figuring out what's wrong with your code. What's even better is if you use an editor with automatic indentation and syntax highlighting, like tuareg-mode in Emacs.

Having lots of regression tests is extremely important because if you're constantly refactoring code, especially if it's code mainly authored by your teammates, you need to know when you've broken something. Knowing that you can check if you've safely refactored code should encourage you to do so more often, which is great because it promotes clarity.
Pair programming is really fun! It also produces fewer bugs and makes quashing them faster, especially if no member in your group is very familiar with OCaml. Chances are that no member remembers all of the OCaml syntax and also that each remembers different things. When programming together, you get a greater coverage of the language.

7.3 Daniel Maxson

This was probably one of the most complex systems I have been a part of building. My role as the software architect placed me in a position where understanding the big picture and how all the components tie together was crucial. Through this project I learned the importance of building and using a regression test suite from the earliest stages of development – it allowed us to be confident that the changes we had made didn't break other components of the whole system. I also became a lot more comfortable with git, particularly tricky git merges which I had always tried to avoid before this project.

Although I never thought I would be saying this I actually came to enjoy programming in OCaml, there are a lot of things which initially don't seem intuitive but after practice it slowly starts to become more familiar. In general it allows you to write a pretty complex system in a very concise manner.

Probably in terms of advice for future teams I would suggest you to find a time to meet to allow you to do most of your programming in the same room as your other team-mates. This allows you to quickly bounce questions off of each other without wasting time in an email exchange. Also I can not stress enough the advantage of pair programming. Having an extra mind and set of eyes

looking at the code helps with catching bugs and makes solving problems a lot faster – it's also more fun than coding by yourself. Also become comfortable with your coding environment – you will be spending endless hours in front of your terminal or text editor so set it up in a way that allows you to focus and be the most efficient. If you are a Vim user download and install Omlet for automatic OCaml styling. Also do your eyes a favor and install flux.

All the advice we got told us to start early, this probably would have made sense but if you find yourself like we did with almost nothing to show and one or two weeks left in the semester don't lose heart. Just realize that you'll be seeing an awful lot of your teammates in the days leading up to the deadline.

7.4 Andre Paiva

Testing was perhaps both the least and most stressful part of the project - it's fun to do your best to break things, and unsettling to realize that you've effectively broken a program you've been obsessing over for the past two or three weeks (the night before said program is due). Do your best to write the most outlandish scenarios possible (pay attention to one lecture where Prof. Edwards will point out strange undefined behaviors in C)!

Weekly meetings are a must, to make sure everyone reasons through broad decisions about the language and to simply force the group to work on something for the week (thanks to this we never discovered the limitations of email for project collaboration). This is much easier in September - when crunch time comes, your group might want to allow for some degree of flexibility in the precise schedule. Other than that, make sure to learn git, don't set up grand expectations for your language, don't fret if it doesn't meet your lower expectations either.

This project is quite a departure from anything else undergraduates have seen in an academic setting. Don't be daunted: as long as you follow the advice given by Prof. Edwards and read about the projects of those who came before, you should be more than well prepared to cook up something respectable. (though if you're reading this, you either don't need this advice or are five minutes from giving our group a grade [comment this part out?]).

Appendix A

Project Log

Note: Andre Paiva has minimal commits because his computer had problems during the time of development. As such, he was mainly using James Lin's and Daniel Maxson's computers in order to commit.

```
[2014—12—16, A. Liu] Refactor SNLObject.java, checked style formatting
[2014—12—16, A. Liu] Merge branch 'master' of github.com:khuumi/SNL
[2014—12—16, A. Liu] scanner lint
[2014—12—16, James Lin] Merge branch 'master' of https://github.com/khuumi/SNL merge
[2014—12—16, James Lin] presentation examples
[2014—12—16, A. Liu] Hopefully made scanner strip comments better
[2014—12—16, tinyvm] Probation
[2014—12—16, tinyvm] Couple of more cases
[2014—12—16, A. Liu] codegen working
[2014—12—16, A. Liu] Continuing edits to fix block generation
[2014—12—16, A. Liu] Merge branch 'master' of github.com:khuumi/SNL
[2014—12—16, A. Liu] Changes to language to support stmts inside stmts
[2014—12—16, James Lin] fixed tests
[2014—12—16, tinyvm] work test work
[2014—12—16, tinyvm] Merge branch 'master' of https://github.com/khuumi/SNL
[2014—12—16, Daniel Maxson] added number to string functionality
[2014—12—16, tinyvm] Merge branch 'master' of https://github.com/khuumi/SNL
[2014—12—16, tinyvm] fixed append test
[2014—12—16, James Lin] Merge branch 'master' of https://github.com/khuumi/SNL merge
[2014—12—16, James Lin] fixed snlobject error
[2014—12—16, tinyvm] Merge branch 'master' of https://github.com/khuumi/SNL
[2014—12—16, James Lin] Merge branch 'master' of https://github.com/khuumi/SNL merge
[2014—12—16, James Lin] adding number_to_word feature
[2014—12—16, tinyvm] append test
[2014—12—16, A. Liu] More changes to parser concerning newlines, fix string test out
[2014—12—16, James Lin] Merge branch 'master' of https://github.com/khuumi/SNL merge
[2014—12—16, James Lin] fixed string bug
[2014—12—16, tinyvm] Ok, no input tests then.
[2014—12—16, A. Liu] Merge branch 'master' of github.com:khuumi/SNL
[2014—12—16, Andre Paiva] Merge branch 'master' of https://github.com/khuumi/SNL
[2014—12—16, A. Liu] Allow for actual multiple newlines before program
[2014—12—16, Andre Paiva] Input tests anyone?
[2014—12—16, A. Liu] add diagnostics for recipe internals
[2014—12—16, A. Liu] Add diagnostics checking to recipe internals
[2014—12—16, Daniel Maxson] merging
[2014—12—16, Daniel Maxson] added word_to_number library function
[2014—12—16, James Lin] fixed word_to_number library function
[2014—12—16, James Lin] added word_to_number library function
[2014—12—16, James Lin] factorial tests
```

```
40 [2014-12-16, Daniel Maxson] fixed a bug with the Next statement
41 [2014-12-16, James Lin] Merge branch 'master' of https://github.com/khuumi/SNL merge
42 [2014-12-16, James Lin] fixed bug in snlobject
43 [2014-12-16, A. Liu] Merging
44 [2014-12-16, A. Liu] Working on recipe call checking
45 [2014-12-15, Daniel Maxson] added list length
46 [2014-12-15, Daniel Maxson] merge'
47 [2014-12-15, Daniel Maxson] added list library functions
48 [2014-12-15, James Lin] added remove_back functionality
49 [2014-12-15, James Lin] adding list feature tests
50 [2014-12-15, James Lin] housekeeping
51 [2014-12-15, James Lin] Merge branch 'master' of https://github.com/khuumi/SNL merge
52 [2014-12-15, James Lin] re-organizing executables and scripts
53 [2014-12-15, Daniel M.] Update TODO
54 [2014-12-15, A. Liu] Merge branch 'analyzer'
55 [2014-12-15, A. Liu] finish analysis error generation, updating tests
56 [2014-12-15, A. Liu] Recoding analyzer's error finding
57 [2014-12-15, Daniel Maxson] remerging
58 [2014-12-15, Daniel Maxson] merging
59 [2014-12-15, James Lin] fixed while result
60 [2014-12-15, Daniel Maxson] add s_ to all stage names
61 [2014-12-15, James Lin] fib results fixed
62 [2014-12-15, Daniel Maxson] merge
63 [2014-12-15, Daniel Maxson] fixed problem with static variables messing up recursion
64 [2014-12-15, James Lin] revised tests
65 [2014-12-15, James Lin] merging
66 [2014-12-15, James Lin] test updates
67 [2014-12-15, A. Liu] Merge branch 'master' of github.com:khuumi/SNL
68 [2014-12-15, A. Liu] Fix a test, allow for NEWLINE before program starts
69 [2014-12-15, Daniel Maxson] merge
70 [2014-12-15, Daniel Maxson] re-added code to clear the global hashtbl for each recipe and
       fixed formal argument bug
71 [2014-12-15, A. Liu] Merge branch 'master' of github.com:khuumi/SNL
72 [2014-12-15, A. Liu] Fixing syntax errors in tests
73 [2014-12-15, Daniel Maxson] mergin
74 [2014-12-15, Daniel Maxson] got rid of compile warnings
75 [2014-12-15, James Lin] adding and cleaning tests
76 [2014-12-15, James Lin] adding tests
77 [2014-12-15, A. Liu] Working on making the tests pass
78 [2014-12-15, A. Liu] Continuing to refactor codegen
79 [2014-12-15, A. Liu] Starting to refactor codegen
80 [2014-12-15, A. Liu] Merge branch 'master' of github.com:khuumi/SNL
81 [2014-12-15, A. Liu] Adding checking for stage flow errors
82 [2014-12-14, James Lin] housekeeping
83 [2014-12-14, James Lin] Merge branch 'master' of https://github.com/khuumi/SNL merge
84 [2014-12-14, James Lin] house cleaning and added list features
85 [2014-12-14, Daniel Maxson] merging
86 [2014-12-14, Daniel Maxson] fixed a bug with block statements
87 [2014-12-14, A. Liu] Merge branch 'master' of github.com:khuumi/SNL
88 [2014-12-14, A. Liu] modifications to analyzer to allow global var access
89 [2014-12-14, Andre Paiva] Merge branch 'master' of https://github.com/khuumi/SNL
90 [2014-12-14, Andre Paiva] More tests
91 [2014-12-14, James Lin] fixed append
92 [2014-12-14, James Lin] Merge branch 'master' of https://github.com/khuumi/SNL merge
93 [2014-12-14, Daniel Maxson] added local -- Analyzer isn't working though.. see testing.snl
94 [2014-12-14, James Lin] Now 5
95 [2014-12-14, James Lin] Added five tests
96 [2014-12-14, Daniel Maxson] fixed some parser bugs -- only local needs to work
97 [2014-12-14, James Lin] revised list example
```

```
98 [2014-12-14, James Lin] Merge branch 'master' of https://github.com/khuumi/SNL resolved
     conflict
99 [2014-12-14, James Lin] switched to array implementation
100 [2014-12-14, Daniel Maxson] removed extra checks from analyzer and finished recipes
101 [2014-12-14, Daniel Maxson] working on recipes
102 [2014-12-14, A. Liu] Add formals to recipe global scope
103 [2014-12-14, A. Liu] Remove unnecessary array of types from TList
104 [2014-12-14, Daniel Maxson] almost working recipes
105 [2014-12-14, A. Liu] lint run_tests.py
106 [2014-12-14, A. Liu] Change list access from int * expr to expr * expr
107 [2014-12-14, Daniel Maxson] started work on recipes
108 [2014-12-14, Daniel Maxson] started work on recipes
109 [2014-12-14, Daniel Maxson] merge conflict resolved
110 [2014-12-14, Daniel Maxson] working ID's and assignments
111 [2014-12-14, James Lin] Andre doing that testing business
112 [2014-12-14, James Lin] fixed file path name stuff
113 [2014-12-14, James Lin] Merge branch 'master' of https://github.com/khuumi/SNL merge
114 [2014-12-14, Daniel Maxson] just in case
115 [2014-12-13, James Lin] Added some tests
116 [2014-12-13, Daniel Maxson] adding support for lists
117 [2014-12-13, Daniel Maxson] fixing a merge conflict
118 [2014-12-13, Daniel Maxson] forced git commit
119 [2014-12-13, James Lin] got --ouput_path fully functional
120 [2014-12-13, Daniel Maxson] refactored code
121 [2014-12-13, James Lin] merge conflict
122 [2014-12-13, James Lin] making chagnes
123 [2014-12-13, A. Liu] editing run_tests script and snl.ml
124 [2014-12-13, James Lin] tests for java gen
125 [2014-12-13, James Lin] changes to list structure, codegen, naming
126 [2014-12-13, James Lin] resolved merge conflict
127 [2014-12-13, James Lin] partial work for merge
128 [2014-12-13, Daniel Maxson] added return
129 [2014-12-13, James Lin] next stage and remove java files before codegen
130 [2014-12-13, James Lin] working on adding tests
131 [2014-12-13, James Lin] rm
132 [2014-12-13, Daniel Maxson] added input and printing
133 [2014-12-13, Daniel Maxson] working print function
134 [2014-12-13, Daniel Maxson] merge
135 [2014-12-13, Daniel Maxson] added unary and binary operations
136 [2014-12-12, A. Liu] Merge branch 'master' of github.com:khuumi/SNL
137 [2014-12-12, A. Liu] Modified run_tests.py to run failing tests
138 [2014-12-12, Daniel Maxson] merge conflict
139 [2014-12-12, James Lin] fixed merge conflict
140 [2014-12-12, Daniel Maxson] working constants
141 [2014-12-12, James Lin] snl can read from file
142 [2014-12-12, A. Liu] Merge branch 'master' of github.com:khuumi/SNL
143 [2014-12-12, A. Liu] Adding tests to make sure syntax errors cause failure and for compiler
      system
144 [2014-12-12, James Lin] got list example working, fixed list operations in snlobject
145 [2014-12-12, Daniel Maxson] merging
146 [2014-12-12, Daniel Maxson] starting to work on code generation
147 [2014-12-12, James Lin] changed source copy as well
148 [2014-12-12, James Lin] product java complete and edits to snl
149 [2014-12-12, Daniel Maxson] merging
150 [2014-12-12, James Lin] small test file
151 [2014-12-12, Daniel Maxson] merging
152 [2014-12-12, Daniel Maxson] started to work on code gen
153 [2014-12-12, James Lin] Merge branch 'master' of https://github.com/khuumi/SNL merge
154 [2014-12-12, James Lin] snlobject for generating java
155 [2014-12-12, A. Liu] Add more newlines in the parser and add tests
```

```
156 [2014—12—11, A. Liu] fix recipe parsing: add NEWLINE
157 [2014—12—11, James Lin] fixed automated test break
158 [2014—12—10, James Lin] work on code gen
159 [2014—12—10, James Lin] remove class files accidentally committed
160 [2014—12—10, A. Liu] space
161 [2014—12—10, A. Liu] Finished with sast creation
162 [2014—12—10, A. Liu] JK sast not done but annotate_expr should be
163 [2014—12—09, A. Liu] Merge branch 'master' of github.com:khuumi/SNL
164 [2014—12—09, A. Liu] Sast is built, working on finishing type checking
165 [2014—12—09, James Lin] updated compilation examples
166 [2014—12—03, Daniel Maxson] Merging
167 [2014—12—03, Daniel Maxson] Working on analyzer, got through compiler with TODOs left
168 [2014—12—03, James Lin] answered some questions
169 [2014—12—03, James Lin] humanly generated java with commentary
170 [2014—11—26, Daniel Maxson] added a todo list
171 [2014—11—26, Daniel Maxson] fixed a merge conflict
172 [2014—11—26, Daniel Maxson] started working on SAST and semantic analyzer
173 [2014—11—25, James Lin] starting code generation
174 [2014—11—17, A. Liu] Start adding program tests, fix a bug in parser
175 [2014—11—13, A. Liu] Add more tests, fix some parser stuff, lint files
176 [2014—11—03, James Lin] python script for outputting AST
177 [2014—11—03, James Lin] input working. tests added.
178 [2014—11—03, A. Liu] Add more expr tests, refactor test script
179 [2014—11—03, A. Liu] Writing tests for the language
180 [2014—11—03, A. Liu] Merge branch 'master' of github.com:khuumi/SNL
181 [2014—11—03, A. Liu] Starting to test everything by printing out ASTs
182 [2014—11—01, James Lin] added input in scanner and parser
183 [2014—10—27, A. Liu] Actually, of course there would be something missing. Added LOCAL in
        parser.
184 [2014—10—27, A. Liu] Merge branch 'master' of github.com:khuumi/SNL
185 [2014—10—27, A. Liu] Parser and scanner theoretically in line with current LRM but untested
186 [2014—10—25, Daniel M.] Update README.md
187 [2014—10—25, Daniel M.] Update README.md
188 [2014—10—25, Daniel M.] Update README.md
189 [2014—10—25, Daniel M.] Update README.md
190 [2014—10—25, Daniel M.] Update README.md
191 [2014—10—25, Daniel M.] Update README.md
192 [2014—10—25, Daniel M.] Update README.md
193 [2014—10—25, Daniel M.] Update README.md
194 [2014—10—25, Daniel M.] Update README.md
195 [2014—10—24, A. Liu] Add statements to parser, move some files around
196 [2014—10—23, A. Liu] Various work on parser, scanner, and ast.
197 [2014—10—22, A. Liu] fix list construction
198 [2014—10—22, A. Liu] add lists to parser, rename some vars
199 [2014—10—22, A. Liu] Begin to work on parser, only very basics down
200 [2014—10—15, James Lin] Merge branch 'master' of https://github.com/khuumi/SNL
201 [2014—10—15, James Lin] edwards code micro c as testing example
202 [2014—10—15, A. Liu] change double to single quotes for \n
203 [2014—10—15, A. Liu] add tokens for SNL
204 [2014—10—15, James Lin] removed temp
205 [2014—10—15, James Lin] adding edward's source code
206 [2014—10—15, Daniel Sadik] making folder structrure
207 [2014—10—15, Daniel Sadik] renamed
208 [2014—10—14, A. Liu] add binary search example
209 [2014—09—24, James Lin] sample code for proposal
210 [2014—09—24, James Lin] first commit
```

Appendix B

Full Source Code

B.1 Parser

```
%{ open Ast %}

%token COMMENT COLON LPAREN RPAREN LBRACKET RBRACKET COMMA
%token PLUS MINUS TIMES DIVIDE ASSIGN
%token EQ NEQ LT LEQ GT GEQ
%token IF ELSE AND OR NOT TRUE FALSE
%token RECIPE DONE START NEXT RETURN DO TO OF LOCAL INPUT
%token <int> INT
%token <float> FLOAT
%token <string> ID STRING
%token NEWLINE EOF

%nonassoc NOCOMMA
%nonassoc COMMA
%nonassoc NOELSE
%nonassoc ELSE
%nonassoc RETURN
%nonassoc DO TO
%right ASSIGN
%left AND OR
%right NOT
%left EQ NEQ LT GT LEQ GEQ
%right OF
%left PLUS MINUS
%left TIMES DIVIDE
%nonassoc UMINUS
%nonassoc LPAREN RPAREN

%start expr
%type <Ast.expr> expr

%start stmt
%type <Ast.stmt> stmt

%start program
%type <Ast.program> program

%%


/* Matches NEWLINE* */
```

```
42 opt_nl:
43     /* nothing */ %prec NOCOMMA { }
44   | multi_nl      %prec NOCOMMA { }
45
46
47 /* Matches NEWLINE+ */
48 multi_nl:
49     NEWLINE          %prec COMMA { }
50   | multi_nl NEWLINE %prec COMMA { }
51
52
53 /* int, float, bool, string literals. */
54 constant:
55     INT    { Int($1) }
56   | FLOAT  { Float($1) }
57   | TRUE   { Bool(true) }
58   | FALSE  { Bool(false) }
59   | STRING { String($1) }
60
61
62 /* Ids may be local or global. */
63 ids:
64     ID       { Id($1, Global) }
65   | LOCAL ID { Id($2, Local) }
66
67
68 /* exprs are the basic building blocks of programs.
69  * No newlines are allowed inside.*/
70 expr:
71     constant                   { Constant($1) }
72   | LPAREN expr RPAREN         { $2 }
73   | ids                        { $1 }
74   | LBRACKET expr_seq RBRACKET { List($2) }
75   | expr ASSIGN expr           { Assign($1, $3) }
76   | math                       { $1 }
77   | logic                      { $1 }
78   | recipe_app                 { $1 }
79   | RETURN expr                { Return($2) }
80   | NEXT ID                    { Next($2) }
81   | INPUT                      { Input }
82   | expr OF ids                { Access($1, $3) }
83
84
85 /* Mathematical expressions. */
86 math:
87     expr PLUS   expr        { Binop($1, Add, $3) }
88   | expr MINUS  expr        { Binop($1, Sub, $3) }
89   | expr TIMES  expr        { Binop($1, Mult, $3) }
90   | expr DIVIDE expr        { Binop($1, Div, $3) }
91   | MINUS expr %prec UMINUS { Unop(Negate, $2) }
92
93
94 /* Boolean expressions. */
95 logic:
96     expr EQ  expr { Binop($1, Equal, $3) }
97   | expr NEQ expr { Binop($1, Neq, $3) }
98   | expr LT  expr { Binop($1, Lt, $3) }
99   | expr LEQ expr { Binop($1, Leq, $3) }
100  | expr GT  expr { Binop($1, Gt, $3) }
101  | expr GEQ expr { Binop($1, Geq, $3) }
```

```
102   | expr AND expr { Binop($1, And, $3) }
103   | expr OR  expr { Binop($1, Or, $3) }
104   | NOT expr      { Unop(Not, $2) }
105 
106 
107 /* A sequence is a comma-separated succession of exprs. It can be used inside
108  * brackets to define a list or when defining or applying recipes. */
109 expr_seq:
110     /* nothing */    %prec NOCOMMA { [] }
111   | expr_seq_builder %prec NOCOMMA { List.rev $1 }
112 
113 
114 expr_seq_builder:
115     expr %prec NOCOMMA          { [$1] }
116   | expr_seq_builder COMMA expr { $3 :: $1 }
117 
118 
119 /* Applying recipes. */
120 recipe_app:
121     DO ID TO expr_seq_builder %prec NOCOMMA { Call($2, List.rev $4) }
122   | DO ID                                   { Call($2, []) }
123 
124 
125 /* A statement is either an expression or an if-else construct. */
126 stmt:
127     expr %prec NOELSE { Expr($1) }
128   | IF expr multi_nl LPAREN block_builder RPAREN
129     ELSE opt_nl LPAREN block_builder RPAREN
130       { If($2, Block(List.rev $5), Block(List.rev $10)) }
131   | IF expr multi_nl LPAREN block_builder RPAREN %prec NOELSE
132       { If($2, Block(List.rev $5), Block([])) }
133 
134 
135 /* A block is a sequence of expr separated by newlines that appears in an
136    if statement. */
137 block_builder:
138     stmt                        { [$1] }
139   | block_builder multi_nl stmt { $3 :: $1 }
140 
141 
142 stage_body:
143     stmt                     { [$1] }
144   | stage_body opt_nl stmt { $3 :: $1 }
145 
146 
147 stage:
148     ID COLON multi_nl stage_body opt_nl DONE       { { sname = $1;
149                                                        body = List.rev $4;
150                                                        is_start = false } }
151   | START ID COLON multi_nl stage_body opt_nl DONE { { sname = $2;
152                                                        body = List.rev $5;
153                                                        is_start = true } }
154 
155 
156 formal_list:
157     ID                    { [$1] }
158   | formal_list COMMA ID { $3 :: $1 }
159 
160 
161 stage_seq:
```

```
    stage                  { [$1] }
  | stage_seq opt_nl stage { $3 :: $1 }


recipe:
    RECIPE ID COLON multi_nl
    stage_seq opt_nl DONE
      { { rname = $2;
          formals = [];
          body = List.rev $5; } }
  | RECIPE ID TO formal_list COLON multi_nl
    stage_seq opt_nl DONE
      { { rname = $2;
          formals = List.rev $4;
          body = List.rev $7; } }


program_body:
    stage opt_nl               { { recipes = [];
                                   stages = [$1]; } }
  | recipe opt_nl              { { recipes = [$1];
                                   stages = []; } }
  | stage opt_nl program_body  { { recipes = $3.recipes;
                                   stages = $1 :: $3.stages; } }
  | recipe opt_nl program_body { { recipes = $1 :: $3.recipes;
                                   stages = $3.stages; } }


program:
    /* nothing */         { { recipes = [];
                              stages = []; } }
  | program_body          { $1 }
  | multi_nl program_body { $2 }
```

Listing B.1: parser.mly

B.2 Scanner

```
{ open Parser }

let digit = ['0'-'9']
let whitespace = [' ' '\t' '\r']
let comment = "#" [^ '\n']* "\n"
let ws_strip = (whitespace|comment|'\n')*


rule tokenize = parse
  (* Whitespace we split on. *)
    whitespace  { tokenize lexbuf }

  (* Comments. *)
  | comment  { tokenize lexbuf }

  (* Binary operators: math, comparison, and logic. *)
  | "+"     { PLUS }
  | "-"     { MINUS }
  | "*"     { TIMES }
  | "/"     { DIVIDE }
  | "="     { EQ }
  | "!="    { NEQ }
```

```
23    | "<"     { LT }
24    | "<="    { LEQ }
25    | ">"     { GT }
26    | ">="    { GEQ }
27    | "and"   { AND }
28    | "or"    { OR }
29    | "not"   { NOT }
30
31    (* Control flow. *)
32    | "if"              { IF }
33    | ws_strip "else"   { ELSE }
34
35    (* Function calls. *)
36    | "do"  { DO }
37    | "to"  { TO }
38
39    (* Used for grouping things and creating lists. *)
40    | "(" ws_strip  { LPAREN }
41    | ws_strip ")"  { RPAREN }
42    | "["           { LBRACKET }
43    | "]"           { RBRACKET }
44    | ","           { COMMA }
45
46    (* Recipe- and stage-related terms. *)
47    | ":"        { COLON }
48    | "recipe"   { RECIPE }
49    | "done"     { DONE }
50    | "start"    { START }
51    | "next"     { NEXT }
52    | "return"   { RETURN }
53
54    (* Other operators used with variables. *)
55    | "is"      { ASSIGN }
56    | "of"      { OF }
57    | "local"   { LOCAL }
58
59    (* I/O *)
60    | "input"   { INPUT }
61
62    (* Identifiers and literals (int, float, bool, string). *)
63    | "true"    { TRUE }
64    | "false"   { FALSE }
65    | digit+ as lxm { INT(int_of_string lxm) }
66    | ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
67    | (digit+'.'digit*)|(digit*'.'digit+) as lxm { FLOAT(float_of_string lxm) }
68    | '"'       { read_string (Buffer.create 17) lexbuf }
69
70    (* Special characters we use to mark end of programs/statements. *)
71    | eof             { EOF }
72    | '\n'+ ws_strip  { NEWLINE }  (* Empty lines are collapsed. *)
73
74    (* Anything else is an illegal character. *)
75    | _ as char  { raise (Failure("illegal character " ^ Char.escaped char)) }
76
77
78  (* Read in string literals. The code is from
79  https://realworldocaml.org/v1/en/html/parsing-with-ocamllex-and-menhir.html *)
80  and read_string buf = parse
81      '"'            { STRING(Buffer.contents buf) }
82    | '\\' '/'       { Buffer.add_char buf '/'; read_string buf lexbuf }
```

```
  | '\\' '\\'       { Buffer.add_char buf '\\'; read_string buf lexbuf }
  | '\\' 'b'        { Buffer.add_char buf '\b'; read_string buf lexbuf }
  | '\\' 'f'        { Buffer.add_char buf '\012'; read_string buf lexbuf }
  | '\\' 'n'        { Buffer.add_char buf '\n'; read_string buf lexbuf }
  | '\\' 'r'        { Buffer.add_char buf '\r'; read_string buf lexbuf }
  | '\\' 't'        { Buffer.add_char buf '\t'; read_string buf lexbuf }
  | '\\' '"'        { Buffer.add_char buf '\\';
                      Buffer.add_char buf '"'; read_string buf lexbuf }
  | [^ '"' '\\']+   { Buffer.add_string buf (Lexing.lexeme lexbuf);
                      read_string buf lexbuf }
  | _               { raise (Failure("Illegal string character: " ^
                                      Lexing.lexeme lexbuf)) }
  | eof             { raise (Failure("String is not terminated")) }
```

Listing B.2: scanner.mll

B.3 AST

```
type op =
  Add | Sub | Mult | Div | Negate |
  Equal | Neq | Lt | Leq | Gt | Geq |
  And | Or | Not

type scope = Local | Global

type constant =
    Int of int
  | Float of float
  | Bool of bool
  | String of string

type expr =
    Constant of constant
  | Id of string * scope
  | Unop of op * expr
  | Binop of expr * op * expr
  | Assign of expr * expr
  | Call of string * expr list
  | List of expr list
  | Return of expr
  | Next of string
  | Input
  | Access of expr * expr

type stmt =
    Expr of expr
  | Block of stmt list
  | If of expr * stmt * stmt

type stage = {
    sname: string;          (* Name of the stage. *)
    body: stmt list;        (* The statements that comprise the stage. *)
    is_start: bool;         (* Whether the stage is a start stage. *)
  }

type recipe = {
    rname: string;           (* Name of the recipe. *)
    formals: string list;    (* Formal argument names. *)
    body: stage list;        (* Stages in the recipe's scope. *)
  }
```

```
type program = {
    recipes: recipe list;
    stages: stage list;
  }


(* Low-level AST printing, to help debug the structure. *)

let op_s = function
    Add -> "Add"
  | Sub -> "Sub"
  | Mult -> "Mult"
  | Div -> "Div"
  | Negate -> "Negate"
  | Equal -> "Equal"
  | Neq -> "Neq"
  | Lt -> "Lt"
  | Leq -> "Leq"
  | Gt -> "Gt"
  | Geq -> "Geq"
  | And -> "And"
  | Or -> "Or"
  | Not -> "Not"

let constant_s = function
    Int(i) -> "Int " ^ string_of_int i
  | Float(f) -> "Float " ^ string_of_float f
  | Bool(b) -> "Bool " ^ string_of_bool b
  | String(s) -> "String " ^ s

let rec expr_s = function
    Constant(c) -> constant_s c
  | Id(str, scope) -> "Id " ^
                        (match scope with Local -> "Local "
                                        | Global -> "Global ") ^
                          str
  | Unop(o, e) -> "Unop " ^ (op_s o) ^ " (" ^ expr_s e ^ ")"
  | Binop(e1, o, e2) -> "Binop (" ^ expr_s e1 ^ ") " ^
                          (op_s o) ^
                            " (" ^ expr_s e2 ^ ")"
  | Assign(v, e) -> "Assign (" ^ expr_s v ^ ") (" ^ expr_s e ^ ")"
  | Call(f, es) -> "Call " ^ f ^ " [" ^
                     String.concat ", " (List.map
                                           (fun e -> "(" ^ expr_s e ^ ")")
                                           es) ^
                       "]"
  | List(es) -> "List [" ^
                  String.concat ", " (List.map
                                        (fun e -> "(" ^ expr_s e ^ ")")
                                        es) ^
                    "]"
  | Return(e) -> "Return (" ^ expr_s e ^ ")"
  | Next(s) -> "Next " ^ s
  | Input -> "input"
  | Access(i, l) -> "Access " ^ (expr_s i) ^ " of " ^ (expr_s l)

let rec stmt_s = function
    Expr(e) -> "Expr (" ^ expr_s e ^ ")"
  | Block(ss) -> "Block [" ^
```

```
                   String.concat ",\n"
                                 (List.map (fun s -> "(" ^ stmt_s s ^ ")") ss) ^
                     "]"
  | If(e, s1, s2) -> "If (" ^ expr_s e ^ ") (" ^ stmt_s s1 ^ ") (" ^
                       stmt_s s2 ^ ")"

let stage_s s =
  "{ sname = \"" ^ s.sname ^ "\"\n" ^
    "   is_start = " ^ string_of_bool s.is_start ^ "\n" ^
      "   body = [" ^ String.concat ",\n" (List.map stmt_s s.body) ^
        "]}\n"

let recipe_s r =
  "{ rname = \"" ^ r.rname ^ "\"\n" ^
    "   formals = [" ^ String.concat ", " r.formals ^ "]\n" ^
      "   body = [" ^ String.concat ",\n" (List.map stage_s r.body) ^
        "]}\n"

let program_s prog =
  "recipes = [" ^ String.concat ",\n" (List.map recipe_s prog.recipes) ^
    "],\n" ^
      "stages = [" ^  String.concat ",\n" (List.map stage_s prog.stages) ^ "]"
```

Listing B.3: ast.ml

B.4 SAST

```

(* The basic types used in annotation *)
type t =
    TInt
  | TFloat
  | TBool
  | TString
  | TList
  | TOCamlString
  | TUnknown

type a_constant =
    AInt of int * t
  | AFloat of float * t
  | ABool of bool * t
  | AString of string * t

type a_expr =
    AConstant of a_constant
  | AId of string * Ast.scope * t
  | AUnop of Ast.op * a_expr * t
  | ABinop of a_expr * Ast.op * a_expr * t
  | AAssign of a_expr * a_expr
  | ANext of string * t
  | AReturn of a_expr * t
  | AList of a_expr list * t
  | AInput of t
  | ACall of string * a_expr list * t
  | AAccess of a_expr * a_expr * t

type a_stmt =
    AExpr of a_expr
  | ABlock of a_stmt list
```

```
  | AIf of a_expr * a_stmt * a_stmt

type a_stage = {
    sname: string;          (* Name of the stage. *)
    body: a_stmt list;      (* The annotated statements in the stage. *)
    is_start: bool;         (* Whether the stage is a start stage. *)
  }

type a_recipe = {
    rname: string;          (* Name of the recipe. *)
    formals: string list;   (* Formal argument names. *)
    body: a_stage list;       (* Stages in the recipe's scope. *)
  }

type a_program = {
    recipes: a_recipe list;
    stages: a_stage list;
  }
```

Listing B.4: sast.ml

B.5 Semantic Analyzer

```
open Ast
open Sast


module StringMap = Map.Make(String);;
module StringSet = Set.Make(String);;


let lib_funcs = [("show", 1); ("remove", 2); ("insert", 3);
                 ("append", 2); ("length", 1);
                 ("word_to_number", 1); ("number_to_word", 1)];;


(* A symbol table wich includes a parent symbol table
   and variables which are tuples of stings and Sast types *)
type symbol_table = {
    mutable variables : (string * Sast.t) list;
  }


type environment = {
    global_scope : symbol_table;
    local_scope : symbol_table;
  }


let type_of_const (ac : Sast.a_constant) : Sast.t =
  match ac with
    AInt(_, t) -> t
  | AFloat(_, t) -> t
  | ABool(_, t) -> t
  | AString(_, t) -> t


let rec type_of (ae : Sast.a_expr) : Sast.t =
  match ae with
    AConstant(const) -> type_of_const const
```

```
38    | AId(_, _, t) -> t
39    | AUnop(_, _, t) -> t
40    | ABinop(_, _, _, t) -> t
41    | AAssign(e1, e2) -> type_of e2
42    | ANext(_, t) -> t
43    | AReturn(_, t) -> t
44    | AList(_, t) -> t
45    | AInput(t) -> t
46    | ACall(_, _, t) -> t
47    | AAccess(_, _, t) -> t
48
49
50 let find_variable_type (env : environment) (id : Ast.expr) :
51       Sast.t option =
52   try
53     let (_, typ) = match id with
54         Id(name, Local) -> List.find
55                               (fun (s, _) -> s = name)
56                               env.local_scope.variables
57       | Id(name, Global) -> List.find
58                               (fun (s, _) -> s = name)
59                               env.global_scope.variables
60       | _ -> failwith "Error using find_variable_type"
61     in
62     Some(typ)
63   with Not_found -> match id with
64                        Id(_, Global) -> Some(TUnknown)
65                      | _ -> None
66
67
68 (* Check to see if param is important or not *)
69 let mutate_or_add (env : environment) (id : Ast.expr) (new_type : Sast.t) =
70   let typ = find_variable_type env id in
71   let name, scope = match id with
72       Id(i, Local) -> i, env.local_scope
73     | Id(i, Global) -> i, env.global_scope
74     | _ -> failwith "Error using mutate_or_add"
75   in
76   match typ with
77     Some(t) ->
78     (* filter name, t out of symbol_table.variables *)
79     scope.variables <-
80       (name, new_type) :: (List.filter (fun (s, _) -> s <> name)
81                                        scope.variables)
82   | None ->
83      scope.variables <- (name, new_type) :: scope.variables
84
85
86 let annotate_const (c : Ast.constant) : Sast.a_expr =
87   match c with
88     Int(n) -> AConstant(AInt(n, TInt))
89   | Float(f) -> AConstant(AFloat(f, TFloat))
90   | Bool(b) -> AConstant(ABool(b, TBool))
91   | String(s) -> AConstant(AString(s, TString))
92
93
94 let rec annotate_expr (e : Ast.expr) (env : environment) : Sast.a_expr =
95   match e with
96     Constant(c) -> annotate_const c
97   | Id(i, s) ->
```

```
98        (match find_variable_type env e with
99         | Some(x) -> AId(i, s, x)
100        | None -> failwith ("unrecognized identifier " ^ i ^ "."))
101   | Unop(op, e1) ->
102       let ae1 = annotate_expr e1 env in
103       AUnop(op, ae1, type_of ae1)
104   | Binop(e1, op, e2) ->
105       let ae1 = annotate_expr e1 env
106       and ae2 = annotate_expr e2 env in
107       ABinop(ae1, op, ae2, TUnknown)
108   | Assign(e1, e2) ->
109       (match e1 with
110        | Id(str, scope) -> let ae2 = annotate_expr e2 env in
111                            mutate_or_add env e1 (type_of ae2);
112                            let ae1 = annotate_expr e1 env in
113                            AAssign(ae1, ae2)
114        | Access(e, id) -> let ae2 = annotate_expr e2 env in
115                           let ae1 = annotate_expr e1 env in
116                           (match find_variable_type env id with
117                            | Some(TList) -> AAssign(ae1, ae2)
118                            | _ -> failwith "Variable not found")
119        | _ -> failwith "Invalid assignment operation")
120   | Next(s) -> ANext(s, TOCamlString)
121   | Return(e) -> let ae = annotate_expr e env in
122                  AReturn(ae, type_of ae)
123   | List(e_list) -> let ae_list = List.map
124                                     (fun e -> annotate_expr e env)
125                                     e_list in
126                     AList(ae_list, TList)
127   | Input -> AInput(TString)
128   | Call(s, e_list) -> let ae_list = List.map
129                                        (fun e-> annotate_expr e env)
130                                        e_list in
131                        ACall(s, ae_list, TUnknown)
132   | Access(e, id) -> let l = find_variable_type env id in
133                      let ind_expr = annotate_expr e env in
134                      match l with
135                      | Some(TList) ->
136                         AAccess(ind_expr,
137                                 (annotate_expr id env),
138                                 TUnknown)
139                      | _ -> failwith "Bad list access"
140
141
142 let rec annotate_stmt (s : Ast.stmt) (env : environment) : Sast.a_stmt =
143   match s with
144     Expr(e) -> AExpr(annotate_expr e env)
145   | Block(s_list) -> ABlock(List.map (fun s -> annotate_stmt s env) s_list)
146   | If(e, s1, s2) -> let ae = annotate_expr e env in
147                      AIf(ae,
148                          annotate_stmt s1 env,
149                          annotate_stmt s2 env)
150
151
152 let annotate_stage (s : Ast.stage) (env : environment) : Sast.a_stage =
153   let new_env = { global_scope = env.global_scope;
154                   local_scope = { variables = []; }; } in
155   { sname = s.sname;
156     body = List.map (fun stmt -> annotate_stmt stmt new_env) s.body;
157     is_start = s.is_start; }
```

```
158
159
160 let annotate_recipe (r : Ast.recipe) : Sast.a_recipe =
161   let new_env = { global_scope = {
162                     variables = List.map (fun s -> (s, TUnknown)) r.formals;
163                   };
164                   local_scope = { variables = []; }; } in
165   { rname = r.rname;
166     formals = r.formals;
167     body = List.map (fun stage -> annotate_stage stage new_env) r.body; }
168
169
170 let annotate_program (p : Ast.program) : Sast.a_program =
171   let new_env = { global_scope = { variables = []; };
172                   local_scope = { variables = []; }; } in
173   { recipes = List.map annotate_recipe p.recipes;
174     stages = List.map (fun stage -> annotate_stage stage new_env) p.stages; }
175
176
177 let rec collect_outs (s : Sast.a_stage) : string list =
178   List.fold_left collect_nexts_stmt [] s.body
179 and collect_nexts_stmt (l : string list) (s : Sast.a_stmt) : string list =
180   match s with
181     AExpr(ae) -> collect_nexts_expr l ae
182   | ABlock(s_l) -> List.fold_left collect_nexts_stmt l s_l
183   | AIf(_, s1, s2) -> collect_nexts_stmt (collect_nexts_stmt l s1) s2
184 and collect_nexts_expr (l : string list) (e : Sast.a_expr) : string list =
185   match e with
186     ANext(s, _) -> if List.exists (fun name -> name = s) l
187                    then l
188                    else s :: l
189   | _ -> l
190
191
192 (* Returns a set of the names of reachable stages and a list of errors with the
193    names of invalid stages attempted to visit. *)
194 let rec visit_stages (queue : string list)
195                      (visited : StringSet.t)
196                      (stages)
197                      (errors : string list) : StringSet.t * string list =
198   if List.length queue = 0
199   then visited, errors
200   else let current = List.hd queue in
201        let nexts = StringMap.find current stages in
202        visit_stages
203          ((List.tl queue) @
204             (List.filter
205                (fun name -> not(List.mem name queue) &&
206                             not(StringSet.mem name visited) &&
207                               StringMap.mem name stages)
208                nexts))
209          (StringSet.add current visited)
210          stages
211          (errors @
212             List.map
213               (fun inval -> "Error in stage " ^ current ^ ": next " ^
214                             inval ^ " calls an invalid stage.")
215               (List.filter
216                  (fun name -> not(StringMap.mem name stages))
217                  nexts))
```

```
218
219
220 (* Returns a list of warnings and a list of errors for unreachable and
221    invalid stages. *)
222 let generate_stage_flow_diagnostics (stages : Sast.a_stage list) :
223       string list * string list =
224   let start = List.find (fun s -> s.is_start) stages in
225   let visited, errors =
226     visit_stages
227       [start.sname]
228       StringSet.empty
229       (List.fold_left (fun map stage ->
230                          StringMap.add stage.sname (collect_outs stage) map)
231                        StringMap.empty
232                        stages)
233       []
234   and stage_set = StringSet.of_list (List.map (fun s -> s.sname) stages) in
235   let unreachable = StringSet.diff stage_set visited in
236   (StringSet.fold
237      (fun name warnings ->
238       ("Warning: stage " ^ name ^ " is unreachable.") :: warnings)
239      unreachable
240      []), errors
241
242
243 (* Checks for duplicate strings in a list and returns the duplicates. *)
244 let dup_string_check (names : string list) : string list =
245   StringMap.fold
246     (fun name count dups ->
247      if count > 1
248      then name :: dups
249      else dups)
250     (List.fold_left
251        (fun map name ->
252         if StringMap.mem name map
253         then StringMap.add name ((StringMap.find name map) + 1) map
254         else StringMap.add name 1 map)
255        StringMap.empty
256        names)
257     []
258
259
260 (* Returns a list of warnings and a list of errors.
261    Warnings: if any stages are unreachable in the program.
262    Errors: if multiple stages have the same name,
263            if the number of stages marked start is not exactly one,
264            if any stages try to call 'next' to a stage that was not defined.
265  *)
266 let generate_stage_diagnostics (stages : Sast.a_stage list) :
267       string list * string list =
268   let snames = List.map (fun s -> s.sname) stages in
269   let dup_name_errors =
270     List.map
271       (fun name -> "Error: multiple stages named " ^ name ^ ".")
272       (dup_string_check snames)
273   and num_starts = List.length (List.filter (fun s -> s.is_start) stages) in
274   let errors =
275     if num_starts > 1
276     then ["Error: more than one stage is marked start."] @ dup_name_errors
277     else if num_starts < 1
```

```
278      then ["Error: no stages marked start."] @ dup_name_errors
279      else dup_name_errors in
280    if List.length errors > 0
281    then [], errors
282    else generate_stage_flow_diagnostics stages
283
284
285  (* Check if multiple recipes have the same name. Returns a list of errors. *)
286  let generate_recipe_diagnostics (recipes : Sast.a_recipe list) =
287    let rnames = List.map (fun r -> r.rname) recipes in
288    List.map
289      (fun name -> "Error: multiple recipes named " ^ name ^ ".")
290      (dup_string_check rnames)
291
292
293  let rec collect_calls (s : Sast.a_stage) : (string * int) list =
294    List.fold_left collect_calls_stmt [] s.body
295  and collect_calls_stmt (l : (string * int) list) (s : Sast.a_stmt) :
296        (string * int) list =
297    match s with
298      AExpr(ae) -> collect_calls_expr l ae
299    | ABlock(s_l) -> List.fold_left collect_calls_stmt l s_l
300    | AIf(_, s1, s2) -> collect_calls_stmt (collect_calls_stmt l s1) s2
301  and collect_calls_expr (l : (string * int) list) (e : Sast.a_expr) :
302        (string * int) list =
303    match e with
304      ACall(name, formals, _) -> (name, List.length formals) :: l
305    | _ -> l
306
307
308  (* Check if all recipe calls are calls to library functions or user-defined
309     functions. Also checks if the number of arguments is correct.
310     Args:
311       recipes: a list of recipes, assumed to have unique names
312       stages: a list of stages
313   *)
314  let generate_call_diagnostics (recipes : Sast.a_recipe list)
315                                (stages : Sast.a_stage list) : string list =
316    let rformals = List.fold_left
317                     (fun l r -> (r.rname, List.length r.formals) :: l)
318                     []
319                     recipes @ lib_funcs in
320    List.fold_left
321      (fun list stage ->
322       (List.fold_left
323          (fun l name_formals ->
324           let name = fst name_formals in
325           let count = snd name_formals in
326           if not(List.mem_assoc name rformals)
327           then ("Error in stage " ^ stage.sname ^ ": call to " ^ name ^
328                 " does not refer to a defined recipe.") :: l
329           else let ecount = List.assoc name rformals in
330                if ecount != count
331                then ("Error in stage " ^ stage.sname ^ ": call to " ^ name ^
332                      " expects " ^ (string_of_int ecount) ^ " arguments but " ^
333                        (string_of_int count) ^ " provided.") :: l
334                else l)
335          []
336          (collect_calls stage)) @ list)
337      []
```

```
    stages


(* Returns a list of diagnostics (warnings and errors) and whether any of the
   diagnostics are fatal errors. *)
let generate_diagnostics (p : Sast.a_program) : string list * bool =
  let r_format name str = "In recipe " ^ name ^ ": " ^ str in
  let r_internal_call_errors =
    List.concat (List.map
                   (fun r -> List.map
                               (fun str -> r_format r.rname str)
                               (generate_call_diagnostics p.recipes r.body))
                   p.recipes)
  and r_internal_diagnostics, has_r_internal_errors =
    List.fold_left
      (fun pair r -> let r_internal_s_warnings, r_internal_s_errors =
                       generate_stage_diagnostics r.body in
                     ((fst pair) @
                        (List.map
                           (fun str -> r_format r.rname str)
                           r_internal_s_warnings) @
                          (List.map
                             (fun str -> r_format r.rname str)
                             r_internal_s_errors)),
                     snd pair || List.length r_internal_s_errors > 0)
      ([], false)
      p.recipes
  and r_errors = generate_recipe_diagnostics p.recipes
  and s_warnings, s_errors = generate_stage_diagnostics p.stages
  and c_errors = generate_call_diagnostics p.recipes p.stages in
  let all_diagnostics = (r_errors @ s_warnings @ s_errors @ c_errors @
                           r_internal_call_errors @ r_internal_diagnostics) in
  all_diagnostics, (has_r_internal_errors ||
                      List.length all_diagnostics - List.length s_warnings > 0)
```

Listing B.5: analyzer.ml

B.6 Code Generator

```
open Ast
open Printf
open Sast


let global_scope = Hashtbl.create 1000;;
let local_scope = Hashtbl.create 1000;;


let get_initial_stage_header (start_stage_name : string)
                             (is_recipe : bool)
                             (formals : string list) =
  if is_recipe
  then let initial = "\n \tpublic SNLObject perform(" in
       let list_of_args = List.map
                            (fun name -> "SNLObject " ^ name ^ "_arg")
                            formals in
       let perform_args =  (String.concat ", " list_of_args) ^ "){\n" in
       let args_in_body = List.map
                            (fun name -> name ^ " = new SNLObject(" ^
                                           name ^ "_arg);\n") formals in
```

```
22        let constructs = (String.concat "" args_in_body) in
23        initial ^ perform_args ^ constructs ^"s_" ^ start_stage_name ^
24          "();\nreturn ret;\n}\n"
25    else "\n public static void main(String args[]){\ns_" ^ start_stage_name ^
26          "();\n}\n"
27
28
29  let to_string_const (const : a_constant) : string =
30    match const with
31      AInt(num, _) -> " new SNLObject(" ^ (string_of_int num) ^ ")"
32    | AFloat(fl, _) ->" new SNLObject(" ^ (string_of_float fl) ^ ")"
33    | ABool(b, _) -> " new SNLObject(" ^  (string_of_bool b) ^ ")"
34    | AString(s, _) -> " new SNLObject(\"" ^ s ^ "\")"
35
36
37  let to_string_id (name : string) (scope : Ast.scope) : string =
38    match scope with
39      Local -> (match Hashtbl.mem local_scope name with
40                  true -> name
41                | false ->  Hashtbl.add local_scope name name;
42                            "SNLObject " ^ name)
43    | Global -> (match Hashtbl.mem global_scope name with
44                   true -> name
45                 | false -> Hashtbl.add global_scope name name; name)
46
47
48  let rec to_string_expr (expr : a_expr) : string =
49    match expr with
50      AConstant(const) -> to_string_const const
51    | AId(name, scope, _) -> to_string_id name scope
52    | AUnop(op, e, _) ->  to_string_unop e op
53    | ABinop(e1, op, e2, _) -> to_string_binop e1 e2 op
54    | AAssign(e1, e2) -> to_string_expr e1 ^  "= " ^ to_string_expr e2
55    | ANext(s, _) -> "s_" ^ s ^ "();\nreturn"
56    | AReturn(e, _) -> "ret = " ^ (to_string_expr e) ^ ";\n" ^ "return"
57    | AList(e_list, _) -> to_string_list e_list
58    | AInput(t) -> "new SNLObject(input.nextLine())"
59    | ACall(s, e_list, _) -> to_string_call s e_list
60    | AAccess(index_e, e, _) -> (to_string_expr e) ^
61                                  ".getArr()[" ^
62                                    (to_string_expr index_e) ^
63                                      ".getInt()]"
64
65
66  and to_string_unop (e : a_expr) (op : Ast.op) : string =
67    let string_op =
68      match op with
69        Negate -> "neg"
70      | Not -> "not"
71      | _ -> "Error" in
72    (to_string_expr e) ^ "." ^ string_op ^ "()"
73
74
75  and to_string_binop (e1 : a_expr) (e2 : a_expr) (op : Ast.op) =
76    let string_op =
77      match op with
78        Add -> "add"
79      | Sub -> "sub"
80      | Mult -> "mult"
81      | Div -> "div"
```

```
82      | Equal -> "eq"
83      | Neq -> "neq"
84      | Gt -> "gt"
85      | Geq -> "geq"
86      | Lt -> "lt"
87      | Leq -> "leq"
88      | And -> "and"
89      | Or -> "or"
90      | _ -> "ERROR" in
91    (to_string_expr e1) ^ "." ^ string_op ^ "(" ^ (to_string_expr e2) ^ ")"
92
93
94  and to_string_call (name : string) (e_list : a_expr list) : string =
95    match name with
96      "show" -> let list_e_strings = List.rev (List.fold_left
97                                                  (fun list e ->
98                                                   (to_string_expr e) :: list)
99                                                  []
100                                                 e_list) in
101               "System.out.println(" ^ (String.concat " + " list_e_strings) ^")"
102   | "remove" -> let lst = to_string_expr (List.nth e_list 0) in
103                 let index = to_string_expr (List.nth e_list 1) in
104                 lst ^ ".remove(" ^ index ^ ")"
105   | "insert" -> let lst = to_string_expr (List.nth e_list 0) in
106                 let item_to_add = (to_string_expr (List.nth e_list 1)) in
107                 let index = to_string_expr (List.nth e_list 2) in
108                 lst ^ ".insert(" ^ index ^ ", " ^ item_to_add ^ ")"
109   | "append" -> let lst = to_string_expr (List.nth e_list 0) in
110                 let item_to_add = to_string_expr (List.nth e_list 1) in
111                 lst ^ ".app(" ^ item_to_add ^ ")"
112   | "length" -> let lst = to_string_expr (List.nth e_list 0) in
113                 lst ^ ".length()"
114   | "word_to_number" -> let word = to_string_expr (List.nth e_list 0) in
115                         word ^ ".word_to_number()"
116   | "number_to_word" -> let word = to_string_expr (List.nth e_list 0) in
117                         word ^ ".number_to_word()"
118   | _ -> let list_e_strings = List.rev (List.fold_left
119                                            (fun list e ->
120                                             (to_string_expr e) :: list)
121                                            []
122                                            e_list) in
123          "new Recipe_" ^ name ^ "().perform(" ^
124            (String.concat ", " list_e_strings) ^ ")"
125
126
127 and to_string_list (e_list : a_expr list) : string =
128   let list_e_strings = List.rev (List.fold_left
129                                    (fun list e ->
130                                     (to_string_expr e) :: list)
131                                    []
132                                    e_list) in
133   "new SNLObject(" ^ (String.concat ", " list_e_strings) ^ ")"
134
135
136 let rec to_string_stmt (statement : a_stmt) =
137   match statement with
138     AExpr(e) -> (to_string_expr e) ^ ";\n"
139   | ABlock(s_list) ->
140      let list_of_strings = List.rev (List.fold_left
141                                         (fun list e ->
```

```
142                                          (to_string_stmt e) :: list)
143                                          []
144                                          s_list) in
145      String.concat "" list_of_strings
146   | AIf(e, first, second) -> let expr_str = (to_string_expr e) in
147                              let first_str = to_string_stmt first in
148                              let second_str = to_string_stmt second in
149                              "if(" ^  expr_str ^ ".getBool())\n{" ^  first_str ^
150                                "}\n" ^ "else{" ^second_str ^ "}\n"
151
152
153 let to_string_stage (stage : a_stage)
154                     (is_recipe : bool)
155                     (formals : string list) : string =
156   Hashtbl.clear local_scope;
157   let header =
158     if is_recipe then "private void s_" ^ stage.sname ^ "(){\n"
159     else "private static void s_" ^ stage.sname ^ "(){\n"
160   in
161   let initial_header =
162     if stage.is_start
163     then get_initial_stage_header stage.sname is_recipe formals
164     else ""
165   in let list_of_strings = List.rev (List.fold_left
166                                          (fun list s ->
167                                           (to_string_stmt s) :: list)
168                                          []
169                                          stage.body) in
170      initial_header ^ header ^ (String.concat "\n" list_of_strings) ^ "}"
171
172
173 let to_string_stages (stages : a_stage list)
174                      (is_recipe : bool)
175                      (formals : string list) =
176   let list_of_strings = List.rev
177                           (List.fold_left
178                              (fun list s ->
179                               (to_string_stage s is_recipe formals) :: list)
180                              []
181                              stages) in
182   let global_vars =
183     if is_recipe then Hashtbl.fold (fun k v acc ->
184                                     "private SNLObject "
185                                     ^ k ^ ";\n" ^ acc) global_scope ""
186     else Hashtbl.fold (fun k v acc ->
187                        "private static SNLObject "
188                        ^ k ^ ";\n" ^ acc) global_scope ""
189   in
190   (String.concat "" list_of_strings) ^ global_vars ^ "}"
191
192
193 (* name should be the file name of the snl file or recipe
194    without any extensions. *)
195 let make_header (name : string) (is_recipe : bool) : string =
196   let scanner = "import java.util.Scanner;\n" in
197   if is_recipe
198   then let scanner2 = "\tprivate Scanner input = new Scanner(System.in);" in
199        scanner ^ "public class " ^ "Recipe_" ^ name ^ "{\n" ^
200          "\tprivate SNLObject ret;\n" ^ scanner2 ^ "\npublic Recipe_" ^ name ^
201            "(){}\n"
```

```
202    else let scanner2 = "\tprivate static Scanner input = " ^
203                          "new Scanner(System.in);" in
204         scanner ^ "public class " ^ name ^ "{\n" ^ scanner2
205 
206 
207 let gen_main (stages : a_stage list) (name : string) : string =
208   make_header name false ^ to_string_stages stages false []
209 
210 
211 let gen_recipe (recipe : a_recipe) : string =
212   Hashtbl.clear global_scope;
213   List.iter (fun formal -> Hashtbl.add global_scope formal formal)
214             recipe.formals;
215   make_header recipe.rname true ^
216     to_string_stages recipe.body true recipe.formals
```

Listing B.6: codegen.ml

B.7 Compiler

```
1  (* Usage: ./snlc [-e | -s | -p | -j] file [-o output_file] *)
2  
3  open Analyzer
4  open Printf
5  open Sast
6  
7  
8  type action = Expr | Stmt | Program | Java
9  
10 
11 let write_out (filename : string) (buffer : string) =
12   if Sys.file_exists filename then Sys.remove(filename);
13   let file = (open_out_gen
14                 [Open_creat; Open_wronly; Open_text]
15                 0o666
16                 filename) in
17   fprintf file "%s" buffer;
18   close_out file
19 
20 
21 let _ =
22   let action = List.assoc Sys.argv.(1) [("-e", Expr);
23                                         ("-s", Stmt);
24                                         ("-p", Program);
25                                         ("-j", Java);] in
26   let lexbuf = Lexing.from_channel (open_in Sys.argv.(2)) in
27   match action with
28     (* expr, stmt, and program are for testing the AST, java is code gen *)
29     Expr -> print_string (Ast.expr_s (Parser.expr Scanner.tokenize lexbuf))
30   | Stmt -> print_string (Ast.stmt_s (Parser.stmt Scanner.tokenize lexbuf))
31   | Program -> print_string (Ast.program_s
32                               (Parser.program Scanner.tokenize lexbuf))
33 
34   | Java ->
35      (* see if file exists and remove if it is already there *)
36      let strlst = Str.split (Str.regexp "/") Sys.argv.(2) in
37      let tail = List.hd (List.rev strlst) in
38      let name = String.sub tail 0 ((String.length tail) - 4)
39      and path = if Array.length Sys.argv > 3 && Sys.argv.(3) = "--output_path"
40                 then Sys.argv.(4) ^ "/"
```

```
41                  else "./" in
42      let ast = Parser.program Scanner.tokenize lexbuf in
43      let sast = Analyzer.annotate_program ast in
44      let diagnostics, any_error = Analyzer.generate_diagnostics sast in
45      List.iter print_endline diagnostics;
46      if any_error
47      then failwith "Errors in program."
48      else write_out (path ^ name ^ ".java") (Codegen.gen_main sast.stages name);
49      ignore (List.map
50                 (fun recipe -> write_out
51                                  (path ^ "Recipe_" ^ recipe.rname ^ ".java")
52                                  (Codegen.gen_recipe recipe))
53                 sast.recipes)
```

Listing B.7: snlc.ml

B.8 SNL Script

```
1 #!/bin/bash
2 
3 ./snlc -j $1 > /dev/null
4 x=$1
5 y=${x%.snl}
6 javac "$y.java"
7 echo "To run program, enter: java $y"
```

Listing B.8: snl

B.9 Java Backend

```
1  public class SNLObject {
2      // used for comparison in typeCheck
3      private enum Type {
4          INT, FLOAT, BOOL, STRING, LIST;
5      }
6  
7      // all of the different meta data available for Object wrapper
8      private Type type;
9      private int valueInt;
10     private double valueFloat;
11     private boolean valueBool;
12     private String valueString;
13     private SNLObject[] valueList;
14 
15     // constructor for int object
16     public SNLObject(int vInt) {
17         type = Type.INT;
18         valueInt = vInt;
19     }
20 
21     // constructor for float object
22     public SNLObject(double vFloat) {
23         type = Type.FLOAT;
24         valueFloat = vFloat;
25     }
26 
27     // constructor for bool object
28     public SNLObject(boolean vBool) {
29         type = Type.BOOL;
```

```
30          valueBool = vBool;
31      }
32  
33      // constructor for string object
34      public SNLObject(String vString) {
35          type = Type.STRING;
36          valueString = vString;
37      }
38  
39      // constructor for list object
40      // t is moved because of Java requirements
41      public SNLObject(SNLObject ... objects) {
42          type = Type.LIST;
43          valueList = new SNLObject[objects.length];
44          for (int i = 0; i < objects.length; i++)
45              valueList[i] = objects[i];
46      }
47  
48      // copy constructor
49      public SNLObject(SNLObject old) {
50          type = old.type;
51          switch (type) {
52          case INT:
53              valueInt = old.getInt();
54              break;
55          case FLOAT:
56              valueFloat = old.getFloat();
57              break;
58          case BOOL:
59              valueBool = old.getBool();
60              break;
61          case STRING:
62              valueString = old.getString();
63              break;
64          case LIST:
65              valueList = new SNLObject[old.getArr().length];
66              for (int i = 0; i < old.getArr().length; i++)
67                  valueList[i] = old.getArr()[i];
68              break;
69          }
70      }
71  
72      // Getter methods for private data.
73      private double getFloat() {
74          return valueFloat;
75      }
76  
77      private String getString() {
78          return valueString;
79      }
80  
81      // These three are the only public ones
82      // because of if statements and access.
83      public boolean getBool() {
84          return valueBool;
85      }
86  
87      public SNLObject[] getArr() {
88          return valueList;
89      }
```

```
90
91      public int getInt() {
92          return valueInt;
93      }
94
95      // goes from a string to a number
96      public SNLObject word_to_number() {
97          return new SNLObject(Integer.parseInt(getString()));
98      }
99
100     // goes from a number to a string
101     public SNLObject number_to_word() {
102         SNLObject ret = null;
103         if (type == Type.INT)
104             ret = new SNLObject(String.valueOf(getInt()));
105         if (type == Type.FLOAT)
106             ret = new SNLObject(String.valueOf(getFloat()));
107         return ret;
108     }
109
110     // helper method to check types
111     private static boolean typeMatch(SNLObject subject, SNLObject desired) {
112         return subject.type == desired.type;
113     }
114
115     // this is the '+' operator
116     public SNLObject add(SNLObject right) {
117         SNLObject snlo = null;
118         // if types match
119         if (typeMatch(this, right)) {
120             // add two ints
121             if (type == Type.INT)
122                 snlo = new SNLObject(this.getInt() + right.getInt());
123             // add two floats
124             else if (type == Type.FLOAT)
125                 snlo = new SNLObject(this.getFloat() + right.getFloat());
126             // add two strings
127             else if (type == Type.STRING)
128                 snlo = new SNLObject(this.getString() + right.getString());
129         }
130         // can also add float and int
131         else if (type == Type.FLOAT && right.type == Type.INT)
132             snlo = new SNLObject(this.getFloat() + right.getInt());
133         // can also add int and float
134         else if (type == Type.INT && right.type == Type.FLOAT)
135             snlo = new SNLObject(this.getInt() + right.getFloat());
136         // return is null if something went wrong at runtime
137         return snlo;
138     }
139
140     // this is the '-' binary operator
141     public SNLObject sub(SNLObject right) {
142         SNLObject snlo = null;
143         // if types match
144         if (typeMatch(this, right)) {
145             // sub two ints
146             if (type == Type.INT)
147                 snlo = new SNLObject(this.getInt() - right.getInt());
148             // sub two floats
149             if (type == Type.FLOAT)
```

```
150                 snlo = new SNLObject(this.getFloat() - right.getFloat());
151         }
152         // can also sub float and int
153         else if (type == Type.FLOAT && right.type == Type.INT)
154             snlo = new SNLObject(this.getFloat() - right.getInt());
155         // can also sub int and float
156         else if (type == Type.INT && right.type == Type.FLOAT)
157             snlo = new SNLObject(this.getInt() - right.getFloat());
158         // return is null if something went wrong at runtime
159         return snlo;
160     }
161 
162     // this is the '*' operator
163     public SNLObject mult(SNLObject right) {
164         SNLObject snlo = null;
165         // if types match
166         if (typeMatch(this, right)) {
167             // mult two ints
168             if (type == Type.INT)
169                 snlo = new SNLObject(this.getInt() * right.getInt());
170             // mult two floats
171             if (type == Type.FLOAT)
172                 snlo = new SNLObject(this.getFloat() * right.getFloat());
173         }
174         // can also mult float and int
175         else if (type == Type.FLOAT && right.type == Type.INT)
176             snlo = new SNLObject(this.getFloat() * right.getInt());
177         // can also mult int and float
178         else if (type == Type.INT && right.type == Type.FLOAT)
179             snlo = new SNLObject(this.getInt() * right.getFloat());
180         // return is null if something went wrong at runtime
181         return snlo;
182     }
183 
184     // this is the '/' operator
185     // errors like divide by zero caught at runtime
186     public SNLObject div(SNLObject right) {
187         SNLObject snlo = null;
188         // if types match
189         if (typeMatch(this, right)) {
190             // mult two ints
191             if (type == Type.INT)
192                 snlo = new SNLObject(this.getInt() / right.getInt());
193             // mult two floats
194             if (type == Type.FLOAT)
195                 snlo = new SNLObject(this.getFloat() / right.getFloat());
196         }
197         // can also mult float and int
198         else if (type == Type.FLOAT && right.type == Type.INT)
199             snlo = new SNLObject(this.getFloat() / right.getInt());
200         // can also mult int and float
201         else if (type == Type.INT && right.type == Type.FLOAT)
202             snlo = new SNLObject(this.getInt() / right.getFloat());
203         // return is null if something went wrong at runtime
204         return snlo;
205     }
206 
207     // this is the '-' unary operator
208     public SNLObject neg() {
209         SNLObject snlo = null;
```

```
210         // can neg int
211         if (type == Type.INT)
212             snlo = new SNLObject(getInt() * (-1));
213         // can neg float
214         else if (type == Type.FLOAT)
215             snlo = new SNLObject(getFloat() * (-1));
216         // return is null if something went wrong at runtime
217         return snlo;
218     }
219 
220     // this is the '=' binary operator
221     public SNLObject eq(SNLObject right) {
222         SNLObject snlo = null;
223         // if types match
224         if (typeMatch(this, right)) {
225             // eq two ints
226             if (type == Type.INT)
227                 snlo = new SNLObject(this.getInt() == right.getInt());
228             // eq two floats
229             if (type == Type.FLOAT)
230                 snlo = new SNLObject(this.getFloat() == right.getFloat());
231             // eq two bools
232             if (type == Type.BOOL)
233                 snlo = new SNLObject(this.getBool() == right.getBool());
234             // eq two strings
235             if (type == Type.STRING)
236                 snlo = new SNLObject(
237                     this.getString().equals(right.getString()));
238         } else {
239             // eq for a float and an int
240             // 4.0 and 4 evaluate to the same
241             if (type == Type.FLOAT && right.type == Type.INT) {
242                 Integer tmp = new Integer(right.getInt());
243                 snlo = new SNLObject(this.getFloat() == tmp.floatValue());
244             }
245             if (type == Type.INT && right.type == Type.FLOAT) {
246                 Integer tmp = new Integer(getInt());
247                 snlo = new SNLObject(tmp.floatValue() == right.getFloat());
248             }
249         }
250         // return is null if something went wrong at runtime
251         return snlo;
252     }
253 
254     // this is the '!=' binary operator
255     public SNLObject neq(SNLObject right) {
256         SNLObject snlo = null;
257         // if types match
258         if (typeMatch(this, right)) {
259             // neq two ints
260             if (type == Type.INT)
261                 snlo = new SNLObject(this.getInt() != right.getInt());
262             // neq two floats
263             else if (type == Type.FLOAT)
264                 snlo = new SNLObject(this.getFloat() != right.getFloat());
265             // neq two bools
266             else if (type == Type.BOOL)
267                 snlo = new SNLObject(this.getBool() != right.getBool());
268             // neq two strings
269             else if (type == Type.STRING)
```

```
270                 snlo = new SNLObject(
271                     !this.getString().equals(right.getString()));
272         } else {
273             // neq for a float and an int
274             // 4.0 and 4 evaluate to the same
275             if (type == Type.FLOAT && right.type == Type.INT) {
276                 Integer tmp = new Integer(right.getInt());
277                 snlo = new SNLObject(this.getFloat() != tmp.floatValue());
278             }
279             if (type == Type.INT && right.type == Type.FLOAT) {
280                 Integer tmp = new Integer(getInt());
281                 snlo = new SNLObject(tmp.floatValue() != right.getFloat());
282             }
283         }
284         // return is null if something went wrong at runtime
285         return snlo;
286     }
287 
288     // this is the '<' binary operator
289     public SNLObject lt(SNLObject right) {
290         SNLObject snlo = null;
291         // if types match
292         if (typeMatch(this, right)) {
293             // lt two ints
294             if (type == Type.INT)
295                 snlo = new SNLObject(this.getInt() < right.getInt());
296             // lt two floats
297             if (type == Type.FLOAT)
298                 snlo = new SNLObject(this.getFloat() < right.getFloat());
299         } else {
300             // lt for a float and an int
301             if (type == Type.FLOAT && right.type == Type.INT) {
302                 Integer tmp = new Integer(right.getInt());
303                 snlo = new SNLObject(this.getFloat() < tmp.floatValue());
304             }
305             if (type == Type.INT && right.type == Type.FLOAT) {
306                 Integer tmp = new Integer(getInt());
307                 snlo = new SNLObject(tmp.floatValue() < right.getFloat());
308             }
309         }
310         // return is null if something went wrong at runtime
311         return snlo;
312     }
313 
314     // this is the '<=' binary operator
315     public SNLObject leq(SNLObject right) {
316         SNLObject snlo = null;
317         // if types match
318         if (typeMatch(this, right)) {
319             // leq two ints
320             if (type == Type.INT)
321                 snlo = new SNLObject(this.getInt() <= right.getInt());
322             // leq two floats
323             if (type == Type.FLOAT)
324                 snlo = new SNLObject(this.getFloat() <= right.getFloat());
325         } else {
326             // leq for a float and an int
327             if (type == Type.FLOAT && right.type == Type.INT) {
328                 Integer tmp = new Integer(right.getInt());
329                 snlo = new SNLObject(this.getFloat() <= tmp.floatValue());
```

```
330             }
331             if (type == Type.INT && right.type == Type.FLOAT) {
332                 Integer tmp = new Integer(getInt());
333                 snlo = new SNLObject(tmp.floatValue() <= right.getFloat());
334             }
335         }
336         // return is null if something went wrong at runtime
337         return snlo;
338     }
339 
340     // this is the '>' binary operator
341     public SNLObject gt(SNLObject right) {
342         SNLObject snlo = null;
343         // if types match
344         if (typeMatch(this, right)) {
345             // gt two ints
346             if (type == Type.INT)
347                 snlo = new SNLObject(this.getInt() > right.getInt());
348             // gt two floats
349             if (type == Type.FLOAT)
350                 snlo = new SNLObject(this.getFloat() > right.getFloat());
351         } else {
352             // gt for a float and an int
353             if (type == Type.FLOAT && right.type == Type.INT) {
354                 Integer tmp = new Integer(right.getInt());
355                 snlo = new SNLObject(this.getFloat() > tmp.floatValue());
356             }
357             if (type == Type.INT && right.type == Type.FLOAT) {
358                 Integer tmp = new Integer(getInt());
359                 snlo = new SNLObject(tmp.floatValue() > right.getFloat());
360             }
361         }
362         // return is null if something went wrong at runtime
363         return snlo;
364     }
365 
366     // this is the '>=' binary operator
367     public SNLObject geq(SNLObject right) {
368         SNLObject snlo = null;
369         // if types match
370         if (typeMatch(this, right)) {
371             // geq two ints
372             if (type == Type.INT)
373                 snlo = new SNLObject(this.getInt() >= right.getInt());
374             // geq two floats
375             if (type == Type.FLOAT)
376                 snlo = new SNLObject(this.getFloat() >= right.getFloat());
377         } else {
378             // geq for a float and an int
379             if (type == Type.FLOAT && right.type == Type.INT) {
380                 Integer tmp = new Integer(right.getInt());
381                 snlo = new SNLObject(this.getFloat() >= tmp.floatValue());
382             }
383             if (type == Type.INT && right.type == Type.FLOAT) {
384                 Integer tmp = new Integer(getInt());
385                 snlo = new SNLObject(tmp.floatValue() >= right.getFloat());
386             }
387         }
388 
389         // return is null if something went wrong at runtime
```

```
390         return snlo;
391     }
392 
393     // this is the 'and' binary operator
394     public SNLObject and(SNLObject right) {
395         SNLObject snlo = null;
396         // if types match
397         if (typeMatch(this, right)) {
398             // and two bools
399             if (type == Type.BOOL)
400                 snlo = new SNLObject(this.getBool() && right.getBool());
401         }
402         // return is null if something went wrong at runtime
403         return snlo;
404     }
405 
406     // this is the 'or' binary operator
407     public SNLObject or(SNLObject right) {
408         SNLObject snlo = null;
409         // if types match
410         if (typeMatch(this, right)) {
411             // or two bools
412             if (type == Type.BOOL)
413                 snlo = new SNLObject(this.getBool() || right.getBool());
414         }
415         // return is null if something went wrong at runtime
416         return snlo;
417     }
418 
419     // this is the 'not' unary operator
420     public SNLObject not() {
421         SNLObject snlo = null;
422         if (type == Type.BOOL)
423             snlo = new SNLObject(!getBool());
424         // return is null if something went wrong at runtime
425         return snlo;
426     }
427 
428     // this is to append an element to the list
429     public void app(SNLObject obj) {
430         SNLObject[] tmp = new SNLObject[valueList.length + 1];
431         System.arraycopy(valueList, 0, tmp, 0, valueList.length);
432         tmp[tmp.length-1] = obj;
433         valueList = tmp;
434     }
435 
436     // insert into a list
437     public void insert(SNLObject index, SNLObject obj) {
438         int insertLocation = index.getInt();
439         SNLObject[] tmp = new SNLObject[valueList.length + 1];
440         System.arraycopy(valueList, 0, tmp, 0, insertLocation);
441         tmp[insertLocation] = obj;
442         for (int i = insertLocation + 1; i < tmp.length; i++)
443             tmp[i] = valueList[i-1];
444         valueList = tmp;
445     }
446 
447     // remove index from a list
448     public SNLObject remove(SNLObject index) {
449         int rmLocation = index.getInt();
```

```java
        SNLObject[] tmp = new SNLObject[valueList.length - 1];
        System.arraycopy(valueList, 0, tmp, 0, rmLocation);
        SNLObject ret = valueList[rmLocation];
        for(int i = rmLocation; i < tmp.length; i++)
            tmp[i] = valueList[i + 1];
        valueList = tmp;
        return ret;
    }

    // remove from the tail of a list
    public SNLObject remove_back() {
        return remove(new SNLObject(valueList.length - 1));
    }

    // get the length of the list
    public SNLObject length() {
        return new SNLObject(valueList.length);
    }

    public String toString() {
        switch (type) {
        case INT:
            return Integer.toString(getInt());
        case FLOAT:
            return Double.toString(getFloat());
        case BOOL:
            return Boolean.toString(getBool());
        case STRING:
            return getString();
        case LIST:
            String s = "[ ";
            for(int i = 0; i < valueList.length - 1; i++) {
                s = s + valueList[i].toString() + ", ";
            }
            s = s + valueList[valueList.length-1].toString() + " ]";
            return s;
        }
        return null;
    }
}
```

Listing B.9: SNLObject.java

B.10 Test Script

```python
#!/usr/bin/env python

import argparse
import glob
import os
import shutil
import subprocess
import tempfile


AST_BIN = "./snlc"
TOTAL_PASS = 0
TOTAL_FAIL = 0


```

```python
parser = argparse.ArgumentParser(description='Run SNL tests.')
parser.add_argument('-v', action='store_true',
                    help='Print all passing tests.')
args = parser.parse_args()


def run_ast_tests(files, cmd_arg):
    """
    Runs tests to build ASTs from code files.
    The input files must end with the extension '.snl', which may not appear
    anywhere else in the file name.
    The expected output files must be named exactly as the input files except
    that they end with the extension '.out' instead of '.snl'.
    Args:
      files: a list of the names of input files, all of which end in '.snl'.
      cmd_arg: the corresponding argument to pass into the AST-printing binary,
        e.g. '-e' to test expr and '-s' for stmt.
    """
    global TOTAL_PASS
    global TOTAL_FAIL
    for test in files:
        with open(test.replace('.snl', '.out'), 'r') as f:
            expected_output = f.read()
        try:
            output = subprocess.check_output([AST_BIN, cmd_arg, test])
        except subprocess.CalledProcessError as e:
            print 'Error processing %s\n' % test, e
            TOTAL_FAIL += 1
            continue
        if expected_output != output:
            TOTAL_FAIL += 1
            print '\nFAIL: %s' % test
            print 'EXPECTED:\n%s' % expected_output
            print 'ACTUAL:\n%s' % output
        else:
            TOTAL_PASS += 1
            if args.v:
                print 'PASS: %s' % test


def run_expr_tests():
    """
    Runs all the tests in the tests/expr directory.
    """
    print 'Running expr tests...'
    expr_tests = glob.glob('tests/expr/*.snl')
    run_ast_tests(expr_tests, '-e')
    print 'Finished running expr tests.\n'


def run_stmt_tests():
    """
    Runs all the tests in the tests/stmt directory.
    """
    print 'Running stmt tests...'
    stmt_tests = glob.glob('tests/stmt/*.snl')
    run_ast_tests(stmt_tests, '-s')
    print 'Finished running stmt tests.\n'


```

```
76  def run_program_tests():
77      """
78      Runs all the tests in the tests/program directory.
79      """
80      print 'Running program tests...'
81      program_tests = glob.glob('tests/program/*.snl')
82      run_ast_tests(program_tests, '-p')
83      print 'Finished running program tests.\n'
84
85
86  def run_failing_tests():
87      """
88      Runs all the tests in the tests/failing directory.
89      """
90      global TOTAL_PASS
91      global TOTAL_FAIL
92      print 'Running failing tests...'
93      failing_tests = glob.glob('tests/failing/*.snl')
94      with open(os.devnull, 'wb') as DEVNULL:
95          for test in failing_tests:
96              try:
97                  output = subprocess.check_output([AST_BIN, '-j', test,
98                                                   '--output_path', os.devnull],
99                                                  stderr=DEVNULL)
100                 print '\nFAIL: %s' % test
101                 TOTAL_FAIL += 1
102             except subprocess.CalledProcessError as e:
103                 TOTAL_PASS += 1
104                 if args.v:
105                     print 'PASS: %s' % test
106     print 'Finished running failing tests.\n'
107
108
109 def run_java_tests():
110     """
111     Runs all the tests in the tests/java directory.
112     """
113     global TOTAL_PASS
114     global TOTAL_FAIL
115     print 'Running compiler tests...'
116     compiler_tests = glob.glob('tests/java/*.snl')
117     temp_dir = tempfile.mkdtemp()
118     subprocess.call(['javac', '-d', temp_dir, 'SNLObject.java'])
119     with open(os.devnull, 'wb') as DEVNULL:
120         for test in compiler_tests:
121             with open(test.replace('.snl', '.out'), 'r') as f:
122                 expected_output = f.read()
123             try:
124                 name = test[len('tests/java/'):-len('.snl')]
125                 subprocess.call([AST_BIN,
126                                  '-j', test,
127                                  '--output_path', temp_dir],
128                                 stdout=DEVNULL)
129                 subprocess.call(['javac', '-d', temp_dir] +
130                                 glob.glob(temp_dir + "/*.java"))
131                 output = subprocess.check_output(['java',
132                                                   '-classpath', temp_dir,
133                                                   name])
134             except subprocess.CalledProcessError as e:
135                 print 'Error processing %s\n' % test, e
```

```
                TOTAL_FAIL += 1
                continue
            finally:
                for f in os.listdir(temp_dir):
                    if not f.startswith('SNLObject'):
                        os.remove(os.path.join(temp_dir, f))
            if expected_output != output:
                TOTAL_FAIL += 1
                print '\nFAIL: %s' % test
                print 'EXPECTED:\n%s' % expected_output
                print 'ACTUAL:\n%s' % output
            else:
                TOTAL_PASS += 1
                if args.v:
                    print 'PASS: %s' % test
    shutil.rmtree(temp_dir)
    print 'Finished running compiler tests.\n'


def main():
    run_expr_tests()
    run_stmt_tests()
    run_program_tests()
    run_failing_tests()
    run_java_tests()
    print '%d out of %d tests passing.' % (TOTAL_PASS, TOTAL_PASS + TOTAL_FAIL)


if __name__ == '__main__':
    main()
```

Listing B.10: run$_{tests.py}$

B.11 Makefile

```
compiler: snlc.ml objects
	ocamlc -c snlc.ml
	ocamlc -o snlc ast.cmo parser.cmo scanner.cmo str.cma codegen.cmo analyzer.cmo snlc.cmo

objects: scanner parser generator
	ocamlc -c ast.ml sast.ml parser.mli scanner.ml parser.ml analyzer.ml codegen.ml

generator: analyzer.ml codegen.ml

parser: parser.mly
	ocamlyacc -v parser.mly

scanner: scanner.mll
	ocamllex scanner.mll

.PHONY: test
test: compiler
	./run_tests.py

.PHONY: clean
clean:
	rm -f parser.mli scanner.ml parser.ml parser.output *.cmo *.cmi snlc *~
```

Listing B.11: Makefile

B.12 Tests

See ’tests’ folder in source code directory.