W4115 Programming Languages and Translators - Spring 2017

Lava

Project Report

Team members

An Wang (aw3001) - Language Guru
Yimin Wei (yw2907) - System Architect
Jiacheng Liu (jl4784) - Tester
Hongning Yuan (hy2486) - Manager

Table of Contents

1 Introduction

Lava is an imperative programming language. It is static typing and object oriented. Lava provides flexible casting through different primitive types, string manipulations as well as function overloading. As it is very similar with C++, people who have some experience on C++ will find it very easy to get started with.

1.1 Background

The Java island is almost entirely of volcanic origin, and contains 45 active volcanos. Lava is the essence of volcano, and therefore the essence of Java, which will also be the name of our language. Inspired by Java, Lava was originally designed to be a dialect of Java running on JVM. However, for the backward compatibility with microC which we used as a reference, it eventually turned out to be a dialect of C++ on LLVM.

1.2 Goal

1.2.1 Familiarity

The syntax of Lava is similar to Java and C++. As other programming languages do, most essential data types are supported in Lava, including int, float, bool, string and array. In addition, the control flow of Lava is almost the same as C++. The “if … else”, “while” and “for” blocks are used for flow control in an identical way..

1.2.2 Flexibility

Unlike Java, we don’t push users to put everything in classes. Global functions and variables are allowed in Lava. Function overloading has been implemented so that there is no need for users to define different versions of functions with the same behavior. In addition, string manipulations and castings make users be able to convert types freely.

1.2.3 Object Oriented

Due to the existence of classes, Lava is an object oriented language. A class contains fields and methods. Fields can be of primitive types, arrays and other class objects. Methods can also be overloaded. With the help of classes, users can design more flexible custom data types and functionalities.

2 Tutorial

2.1 Environment Setup

Because our compiler will compile Lava to LLVM IR, the LLVM and Clang need to be installed. In Ubuntu 16.04 shell, we can install them with the following instructions:

```
sudo apt-get install llvm
sudo apt-get install clang
```

Note that LLVM 4.0 has been released in the last month, in some environments, the user need to explicitly select the 3.X version to install, otherwise the LLVM 4.0 will be installed. We didn't test our code on LLVM 4.0 so the result may be unexpected.

Because we write our code in Ocaml, Ocaml and opam are also needed to be installed before starting using our compiler.

```
sudo apt-get install ocaml
sudo apt-get install opam
```

In addition, we need to install the llvm library for ocaml as well as the log library.

```
opam init
opam install llvm.3.7
opam install ocamlfind
opam install log
```

Note that the llvm library for ocaml should match your llvm version. For example, if your version is 3.6, you need to install the llvm.3.6 library rather than 3.7.

At last, position yourself in the Lava directory, compile our compiler.

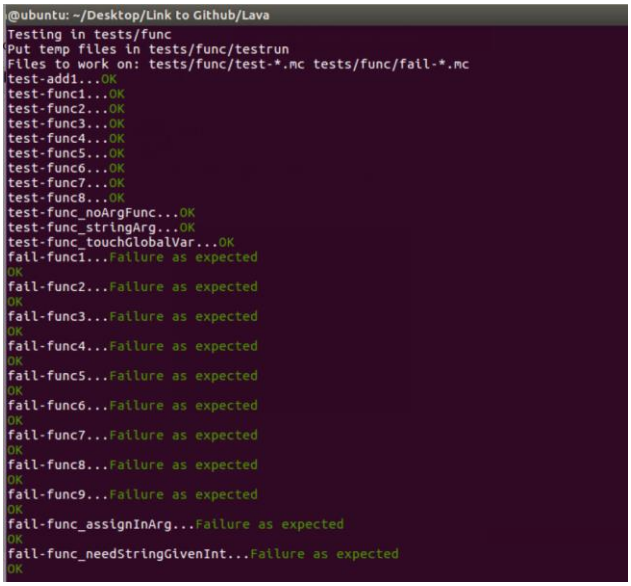
```
./make
```

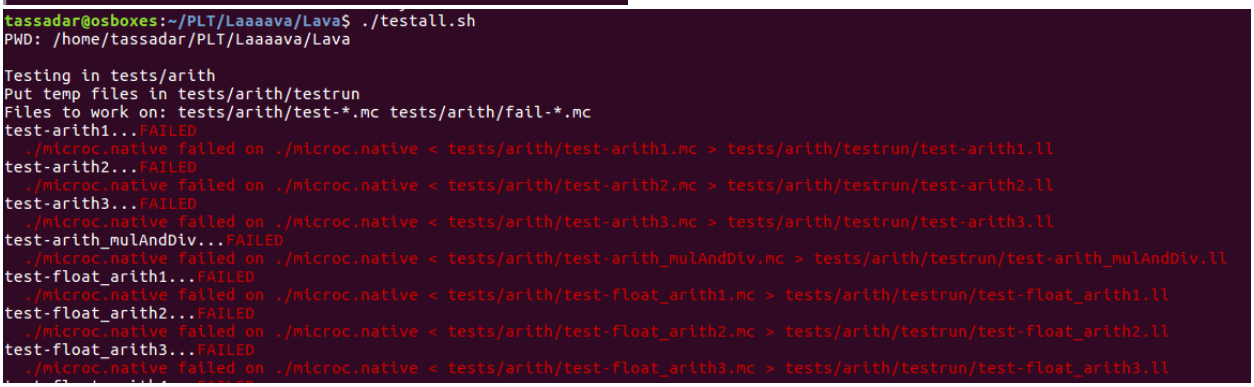
2.2 Testing

To test, the simplest way is to just run the testing script under the root directory.

```
./testall.sh
```

The automated testing suite will recursively run all the tests under the `tests/` directory. If the test result meets the expectation, the output will be in green color. If not, the log will be in red and show in which step(command) the test fails.





2.3 Using the compiler

The executable file of the compiler is lava.native. The program takes a piece of Lava code in and generates the corresponding LLVM code. Here's the basic usage

./lava.native <hello_world.lava > hello_world.ll

The example takes the input from a file by the input redirection and outputs the LLVM code into another file by the output redirection.

In addition, some optional flags could be set to output some debug information.

```
-a    only print the AST
-s    only print the SAST
-l    generate the LLVM_IR, but don't check
-c    generate the LLVM_IR, do check ( the default behavior)
```

2.4 Writing programs

2.4.1 Hello World

```
hello_world.lava
int main() {
      string s;
      s = “Hello World”;
      print(s);
      return 0;
}
```

From the code above, you may find some significant differences between Lava and Java.

First, Although Lava supports class, the main function is defined outside a class. In addition, you can define global functions and variables. To some extent, it is more like C++ than Java.

Second, the string type is a primitive type rather than a class.

Third, you can’t assign to a variable an initial value when you declare it. Note that in every scope, variable declarations must appear prior to any other statements. Once one statement appear, you can’t do no more declarations.

Save your code. Execute the following command in shell, you will see “Hello World” is printed to the console.

```
./lava.native < hello_world.lava > hello_world.ll
lli hello_wordl.ll
Hello World (output)
```

2.4.2 Object-oriented programming

The code example below will demonstrate how to write a piece of object-oriented code. In specific, the user can define a class with fields and functions, initialize objects with the type of the class, access the fields and invoke functions that belong to it.

```
class Node{
    int value;
    int[3] arr;
    /* Can have a field of its own type */
    class Node last;
    /* constructor function */
    constructor(int v){
        value = v;
    }
    /* constructors have the same name */
```

```
    constructor(int v,class Node n){
        value = v;
        last = n;
    }
    /* Convert the list to string */
    string toString(){
       /* Need to declare local var before using them */
        int v;
        string s;

        /* string also needs initialization */
        s = "";
       /* get field from an object */
        v = this.value;
        if(v==0)
            return "r"+s; /* string concatenation */
        s = this.last.toString() + "->" + (string)v;
        return s;
    }
}
int main(){
    /* declare local variables */
    class Node root;
    class Node n1;
    class Node n2;
    class Node n3;
    int v;
    string s;
    int[3] a;
    /* the parameter types must match with declaration */
    root = new Node(0);
    n1 = new Node((int)"1",root);
    n2 = new Node((int)3.0, n1);
    n3 = new Node(5, n2);
    /* uninitialized array have default values */
    a = root.arr;
    print(a[2]); /* Output: 0 */
    /* invoke class function */
    s = n3.toString();
    print(s);
    /* Output "r->0->1->3->5" */
    return 0;
}
```

The code piece can be run in the same way as the last example.

The code piece above exhibits how to define a class and use it in the main function. The class definition includes local variables and functions. The local variables can include primitives, object types, and array. The constructor function takes care of initializing new objects of the class type. In the constructor, class fields are initialized. And the new object will be returned by the constructor and assigned to the left-hand-side variable of assignment if any. Not all fields must be initialized in the constructor but if so, the program will behave differently. If the field is a primitive, accessing uninitialized field, like accessing uninitialized variables in C++, will yield undefined values. If the field is a string, accessing it will give a segmentation fault. Similarly, accessing an uninitialized object reference gives a segmentation fault. However, it the field is array, the values will be default values, as primitives go.

In the main function, the example exhibits how to declare an object, initialize it, access its field, and invoke a class function on it. As the intuition goes, Lava uses dot to denote accessing the field/function belonging to a class.

```
a = root.arr;
s = n3.toString();
```

3 Language Reference Manual

3.1 Introduction

Lava is an object-oriented, statically typed language that serves general purposes. Lava code is compiled to LLVM code and runs on LLVM, which makes it hardware-independent.

Type checking is done at compile time. Programmers will get compilation errors at compile time and thus runtime typing errors are prevented.

Lava is a language that looks quite similar to C++. Therefore the syntax and most basic functionality are very similar to C++. C++ programmers can pick up Lava without much trouble.

3.2 Lexical conventions

3.2.1 Identifiers

Identifiers are sequences of characters used to name classes, variables, and functions.

1. Each identifier must have at least one character.
2. Identifiers can contain letters, digits, and the underscore symbol.
3. Identifiers should not begin with a digit.
4. Keywords cannot be used as identifiers.

5. Identifiers are case sensitive.

3.2.2 Keywords

The table below lists the built-in keywords of Lava.

while	if	else	return
for	break	continue	new
new	void	bool	int
float	string	class	this

3.2.3 Operators

Arithmetic	Relational	Logical	Assignment	Cast	String
+ (Addition)	== (equal to)	&& (logical and)	=	(typename)	+ (string concatenation)
- (Subtraction)	!= (not equal to)	|| (logical or)			== (string comparison)
* (Multiplication)	> (greater than)	! (logical not)			
/ (Division)	< (less than)				
	>= (greater than or equal to)				
	<= (less than or equal to)				

3.2.4 Comments

Lava supports multi-line comments.

Multi-line comments are wrapped between “/*” and “*/”. Everything that are between the start of such a comment, namely “/*”, and the end of the comment, namely “*/”, will be discarded by the compiler and will not execute.

```
int a=5;
/* Everything in the multi-line comment will not execute.
int b=6;
For example, the above line will not execute.
*/
```

The comments will not be treated like Lava code, nor are they designed to be. Comments are for things other than code.

3.2.5 Whitespace

whitespace refers to the following characters: space(‘ ‘), tab(‘\t’), and newline(‘\n’). Any amount of whitespace are allowed outside entities, e.g. identifiers, keywords, and will be ignored.

3.3 Data Types

Lava is a statically typed language. A variable’s type must be specified when declared.

3.3.1 types

3.3.1.1 int

The int type is the type for integers. Capable of containing at least the [-2147483648, +2147483647] range;thus, it is at least 32 bits in size.

```
int num;
num = 3;
```

3.3.1.2 float

The float type represent single-precision floating-point format. It occupies 4 bytes (32 bits) in computer memory and represents a wide dynamic range of values by using a floating point. In IEEE 754-2008 the 32-bit base-2 format is officially referred to as binary32. And Lava doesn’t support distinction between float and double.

```
float num;
num = 3.14
```

3.3.1.3 bool

The boolean type is a binary indicator which can be set to either true or false. A bool value can be one of the two values, either true or false.

```
bool b;
b = false;
```

3.3.1.4 string

string is a sequence of characters. A string's length equals to the number of characters it includes. In Lava, string is passed by reference rather than value.

```
string s;
s  = "Welcome to Lava";
```

Strings could be concatenated by the operator "+".

```
string s;
s = "Welcome" + " to " + "Lava"; /*will get "Welcome to Lava" */
```

Strings could be compared by the operator "==". That operator will return true if the two strings have the same value, no wonder whether their address are the same or not. Otherwise, it will return false;

```
string a;
string b;
a = "foo";
b = "foo";
if (a==b) {
      print("bar"); /* this statement will be executed */
}
```

3.3.1.5 Array

Array is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, the length of array is fixed.

```
int[] array_example;
array_example = new int[10];
```

3.3.1.6 Object

Any class type is considered as an object in Lava. Unlike Java, when referencing to an object type, the type must be started with the keyword “class”. See chapter 6 to know the class in Lava.

```
class Circle foo;     /* variable declaration */
int bar( class Circle foo);/* function declaration */
```

3.3.2 Casting

3.3.2.1 Explicit Casting

Type casting is done by the cast operator, which is a right associative operator. In Lava, casting operator is composed of a left parenthesis, following by the typename and a right parenthesis.

A string could be casted into an int or float. If that string is an valid input, it will returns the casted value, otherwise, 0 will be returned. It has the same behavior as the atoi() and atof() functions in standard C library.

An int or a float could be casted into a string. If the input is a float, the output will have six decimal places.

Casting between the same primitive type is also permitted in Lava, though without any effect.

```
int a;
String b;
float c;
a = (int)”3”;              /*Casting a string to a int */
b = (string)a;             /*Casting an int to a string*/
c = 3.14;
b = (string)c;
/*Casting a float to a string, will return “3.140000” */
```

3.3.2.1 Implicit Casting

In Lava, casting an int to a float is the only permitted implicit casting. If an int type variable and a float type variable appears on two sides of a binary operator, the int type variable will be converted to a float implicitly. In addition, in assignment statements, an int could be implicitly

converted to a float. Note that this implicit casting is asymmetrical. Casting a float to an int need to be carried out with the casting operator.

```
float a;
int b;
a = 3;     /*implicit casting, equivalent to a = (float)3; */
b = 3.14;  /*ERROR, should be b = (int)3.14;*/
a = a + b; /*implicit casting, equivalent to a = a + (float)b; */
```

3.3.2.1 Casting Matrix

from\to	int	float	bool	string	Array	Object
int	implicit	**implicit**		explicit		
float	explicit	implicit		explicit		
bool			implicit	explicit		
string	explicit	explicit		implicit		
Array						
Object						

3.4 Syntax

3.4.1 program structure

At the top level, the program contains the declarations of global variables, global functions and classes. All of them are interchangeable.

```
int a; /*global variable*/
string foo() {   /*global function*/
      return “foo”;
}
class bar() { /*class*/
      constructor(){}
}
```

3.4.2 declarations

A declaration statement declares a variable by specifying its name and type.

```
[class] typename variable_name;
```

The class keyword is required if the variable is an object type. In addition, initial value assignment is not permitted along with the declarations.

```
int x;
bool checked;
class Shape s;

int x = 5; /*error: not permitted*/
```

3.4.3 statements

A statement could be an expression which ends by a semicolon.

```
i = 1;
```

A statement could also be control flow (e.g. if, while and for).

```
if (expression) {
      ...
} else {
      ...
}
```

See the Chapter5 to get more details.

3.4.4 functions

Function declarations includes return type, function name and its parameters.

All the local variables must be declared at the beginning of the function body. After the first statement appears, no variables could be declared any more.

```
return-type function-identifier ([params]) {
      [variable declarations]
      [statements]
}
```

3.4.5 classes

A class declaration include one or more constructors, member field and function declarations. All of them are interchangeable.

The declaration of a constructor is similar to regular functions. However, a constructor don’t need a return type.

```
class classname{
      [variable declarations]
      constructor([params]) {
            [function body]
```

```
      }
      [function declarations]
}
```

3.5 Statements

A statement expresses some action to be carried out.

3.5.1 Expression Statements

An expression statement consists an expression to be executed, ended by a semicolon.

```
i = 1;                        /*Assignment Expression*/
Circle.draw();                /*Method invocation*/
c = new Circle();             /*Class object creation*/
```

3.5.2 Control Flow Statements

Statements are generally executed in sequence, i.e. from top to bottom. The execution flow however can be altered by control flow statements, including condition statements, loop statements and branching statements.

3.5.2.1 Condition statement

The condition statements, also called “if statement”, executes a section of code only if the specified condition is met. An if statement can be followed by an else statement, which executes when the condition in the if statement is not met. These two statements can also be combined and nested.

Plain if statement

```
if (expression) {
      //code to execute when the expression is true
}
```

If and else statement

```
if (expression) {
      //section to execute when the expression is true
} else {
      //code to execute when the expression is false
}
```

Combined if else statement

```
if (expression1) {
      //code to execute when the expression1 is true
} else if (expression2) {
      //code to execute when the expression1 is false and condition2 is
true
} else {
      //code to execute when neither expression1 nor expression2 is met
}
```

An example of *condition statement*

```
/*printing result based on score*/
if (score > 100) {
      print(“Good”);
} else if (score < 60) {
      print(“Poor”)
} else {
      print(“Fair”)
}
```

3.5.3.2 Loop Statements

Lava supports the loop statements defined by for and while keyword.

For statement

The for statement executes a section repeatedly as long as the specified condition is met.

```
for (initialization;condition;loop-execution) {
      //code to execute in the loop
}
```

There are three parameters in a for loop statement: initialization, condition and loop-execution

The initialization initializes the loop variable, whose scope is within the for-loop only. There can be more than one loop variables.

The condition is checked before every loop, if it is met then the loop body executes, otherwise the loop ends.

The loop-execution is executed after every loop, which is usually used to update loop variables.

All three parameters are optional.

An example of *for statement*

```
/*printing 012345678*/
int i;
string s;
S = "";
for (i=0;i<10;i=i+1) {
      s = s + (string)i;
}
print(s);
```

While statement

The while statement executes a section repeatedly as long as the specified condition is met.

```
while (condition) {
      // code to execute in the loop
}
```

The while statement executes the loop body as long as the condition is met.

An example of while statement

```
/*Loading the truck as long as it is not full*/
while (truck.isFull()==false) {
      truck.load();
}
```

3.5.3.3 Branching statements

Lava supports two branching statements, break and continue, which are used inside the loop body.

Break

A break statement terminates the loop of which it is included.

```
/*Breaking for loop*/
int i;
for (i=0;i<1000;i++) {
      if (find(i)== true) {
            print(“Found the item at ” + i);
            break;
      }
}

/*Breaking while loop*/
i = 0;
while(true) {
      if (find(i)== true) {
            print(“Found the item at ” + i);
            break;
      }
      i++;
}
```

3.6 Functions

Functions are a piece of code which can be called to execute.

3.6.1 Function declaration & definition

```
return-type function-identifier ([params]) {
      [variable declarations]
      [statements]
}
```

A function must be declared and defined before being used. Function declaration & definition includes access-modifier, return-type, function-identifier, parameters and function body. In Lava, both global functions and class member functions are supported.

`return-type`: The type of the return value of the function, which can be built-in type or user defined type (class).

`Function-identifier`: The identifier of the function, which must be unique within the class.

`function body`: The code to be executed when the function is called.

`parameters`: The parameters passed to the function, whose scope are within the function body.

A sample function:

```
int square(int i) {
      int a;
      return i*i;
}
```

3.6.2 Function overloading

A function could be overloaded as long as each version has different function signature.

```
int foo(int i) {
      return i*i;
}
int foo(string i) {
      return (int)i*(int)i;
}
```

3.7 Classes

3.7.1 Class Definition

A class definition starts with the keyword ’class’ and follows by the class name. The body of a class is limited in a pair of parentheses. One class should contain a constructor and may contain the following elements: fields, methods. The following is a typical definition of a class.

```
class foo{
      int sampleField;
      /* constructor */
      constructor() {
            /* code */
      }
      void bar() {}
      void sampleMethod() {
            this.sampleField = 1;
            this.bar();
```

```
	}
}
```

3.7.2 Fields and Methods

Fields are variables in class. Methods are member functions in class. Methods could be overloaded as global functions do.

Methods and fields must be referenced with an instance of class, such as

```
	instance.method();
	Instance.field = 3;
```

However, the “this” keyword must be used to qualify a member inside a class function, except in the constructor functions.

```
	/* inside a class function */
	this.method();
	this.field = 3;
```

The “this” variable, as in Java, corresponds to the object that the function is called on. It is passed to the function implicitly as the first parameter, just like how python does this explicitly for class functions:

```
	def foo(self, b):
		self.bar = b
```

3.7.3 Constructor and Initialization

The constructor is the function called when an instance is created. The constructor should have the name “constructor” without a return type.

Constructor could be overloaded. In the constructor, member fields and functions can be referenced without the keyword “this”.

There is no default constructor. The constructors must be defined explicitly.

```
class foo{
	/* constructor */
	constructor(int a) {
		/* no need to write this.sampleField */
		sampleField = a;
	}
```

```
	constructor() {
		/* no need to write this.sampleField */
		sampleField = 0;
	}
	int sampleField;
}
```

An instance of the class could be initialized by one of the constructors.

```
class foo a; /* declaration */
class foo b;
a = new foo(3); /* initialized with constructor foo(int a) */
b = new foo();  /* initialized with constructor foo() */
```

3.8 Built-in functions

Built-in functions in Lava are implemented by invoking the C standard library.

3.8.1 print

The print function takes an argument in int, float, bool and string. This function will output to the stdout. A '\n' character will be appended to the end of the input.

```
	print(3);
	print(3.57);
	print("aaaaa");
	print(false);
```

3.8.2 toString

The toString function takes an argument in int, float, bool and string. This function will convert the input to a string. The behavior of toString is the same to the casting operator (string) if the input is not a string type. If the input is a string type, the toString function will create a new copy of the string rather than returning the string itself.

```
	string a;
	string b;
	string a  = "foo";
	string b = toString(a); /* a and b have different address */
```

4 Project Plan

4.1 Planning Process

We met with the TA every week. At the meeting, we reported to the TA what we have done in the last week and asked him for the advise on what we should do next. After we met with our TA, we usually held another meeting to discuss our work next week. We listed some features that needs to be implemented and then everyone picked some of them. Goals are also adjusted according to the actual progress of every team members.

4.2 Specification Process

We set up our language specification by grabbing the basic features from Java including reflection, and adding to it some support for generic typing.
After discussion with our TA Jacob, we realized that the desired functionality is much more above our head than we had expected. For example, reflection needs to be supported by a runtime which is likely to take a lot of time to implement. So we changed our initial plan, dropping that feature and decided to focus on implementing object features first.
After we finished adding object to Lava, we found it is almost the end of the semester and we did not have enough time to add inheritance. So we reevaluated the schedule and decided to explore some simpler but handy features, and reinforce existing features by adding more tests. Finally, we added casting and string operations, which turned out to be two nice and handy features, while not adding too much overhead to implement. In addition, we achieved function overloading features as the byproduct of the class.
To sum up, we roughly implemented the same volume of features as desired.

4.3 Development Process

We started our projected playing around with the microC program. Two weeks are spent before we officially started our coding. After we understood the microC code, we tried to add the string type to the microC and presented that version to TA as Hello World demo. However, We found harder and harder to put new stuffs into the microC because there are too many hard coded stuffs which are not extendible. Eventually We decided to write a totally new semantic checker and code generator. In the following one month we got a new analyzer to convert the AST to the SAST and built a well-constructed code generator. After that, bunches of new features are added into our language, including casting, array and class, without significant changes in structure of our code. We stucked for two weeks in the implementation of class feature due to the vague llvm documentation and complicated analysis on the context but at last we made it. The function overloading is implemented as the byproduct of class constructed. At last, we planned out one week to add more testing suites and write two demo program for presentation.

4.4 Testing Process

We proceeded our development in a test-driven style. Before adding a feature, we first add tests and settle on the expectation. During the process we keep running the existing regression tests to ensure no existing functionality is corrupted by new code. When the code and new feature pass all tests, we commit to git and the new version is stable. Everyone is responsible for testing his code so all of the team members contributed to the testing suites. In the final week of development we also spent some time on improving the integration tests and re-evaluating existing tests, checking corner cases.

4.5 Team Responsibilities

		Scanner, Parser	Semantic Check	Code Generation	Testing
Honing Yuan	Manager	Class	Class Function overloading		Some test suites
An Wang	Language Guru	Array	Array	Array	Some test suites
Jiaching Liu	Tester			Skeleton code Class Builtin functions	Automated testing scripts Most test suites
Yimin Wei	System Architect	Casting String Float	Skeleton code Casting String Float	Casting String Float Builtin functions	Some test suites

4.6 Style Guide

We made the following convention on our code style when developing our compiler.

1. We use one tab for indent
2. If possible, use parentheses to enclose params for the function call
3. We made all of the variables meaningful so that everyone are able to understand others' code
4. Comments are optional. We believe the most efficient way to figure out a piece of code is to ask that one who wrote it.
5. Never repeat. If something appears twice, we shall write a function to replace them.

4.7 Project Timeline

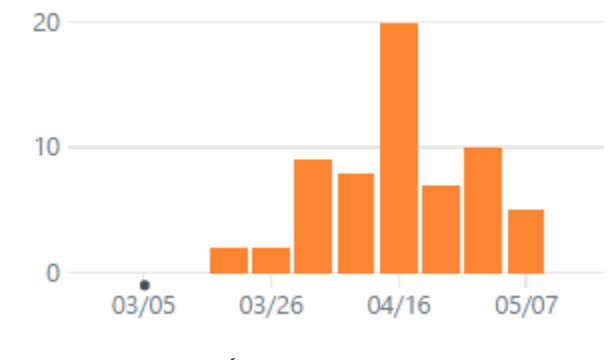
Data	Milestone

February 8	Language Proposal
February 22	Language Reference Manual
March 23	The First Commit
March 27	Hello World Demo
April 3	The first automated testing script
April 6	The new analyzer for SAST generation
April 10	The new scanner, parser and analyzer for class
April 18	The new code generator for class
April 20	Float type, casting and string operations
April 26	Class worked
April 27	Array
May 4	Function overloading
May 6	The demo program
May 9	The final report

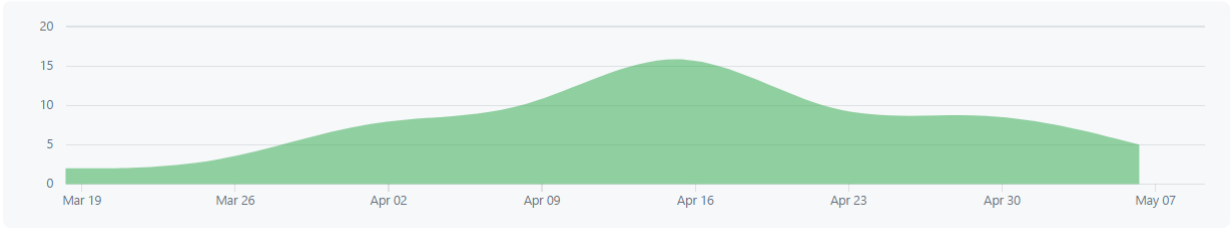
4.8 Software Development Environment

Ocaml: 4.02.3
Operating System: Ubuntu 14.04
LLVM version: 3.4/3.7 (both passed all the tests)
Clang version: 3.4/3.7 (both passed all the tests)
Version control: Github

4.9 Git Commit History



Contributions to master, excluding merge commits




```
commit 52a576ffadb2415a3cf2a369795e6f72dd8132b8
Author: huanyan <happyhn2020@icloud.com>
Date:   Tue May 9 20:16:37 2017 -0400

    Array demo

commit 2199217361c4615fe3ea410c1c741ee9f9c23eaa
Author: huanyan <happyhn2020@icloud.com>
Date:   Tue May 9 17:32:54 2017 -0400

    Update tests

commit 7a644de9f8d5866bd9830226f984193b2349ad05
Merge: 5d06b30 c7bf2dc
Author: anwang0000 <wang.an@columbia.edu>
Date:   Sun May 7 13:58:30 2017 -0400

    Merge branch 'arrayfinal' of github.com:huanyan-hny/Lava into
arrayfinal

commit 5d06b304b93cc120128f1a6369ca64c6e51c0039
Author: anwang0000 <wang.an@columbia.edu>
Date:   Sun May 7 13:53:56 2017 -0400

    add array in obj
```

```
commit c7bf2dcf4aff2af24cdc22985523c27084735fbc
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Sat May 6 08:02:39 2017 +0100

    linkedlist and tree are for the demo

commit f2343289fec57014341a6564cc431a0bfd187d83
Author: huanyan <happyhn2020@icloud.com>
Date:   Thu May 4 01:31:43 2017 -0400

    update analyzer

commit 05d28b668e9aa92efc459b437d3122dbc57197b0
Author: huanyan <happyhn2020@icloud.com>
Date:   Thu May 4 01:06:39 2017 -0400

    class constructors and methods implicit overload stmt

commit 249d14befd844e6ddfa12732ead435c2d1a48aa1
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Thu May 4 05:58:24 2017 +0100

    demo

commit ef881329f2dbfd0c0a3f9d8ae402d612e6ebaee1
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Thu May 4 04:31:35 2017 +0100

    Obj tests pass

commit b3fc669b8ab1b0e2bd5a47961b420a67dfad80bf
Author: huanyan <happyhn2020@icloud.com>
Date:   Wed May 3 16:32:31 2017 -0400

    Implicit method param and method overload

commit 96919753891a554c75c883aabec8289c2e52d027
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Tue May 2 01:20:58 2017 +0100

    Obj functions now work. Arrays in obj not working, not passing
analyzer.

commit 11967715534233e3e055a3a8777d48ed2b73253c
```

```
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Mon May 1 05:32:05 2017 +0100

    Added obj testcases. Some are not working. TBC

commit 4fe26094bedc17d934c5a00e4704cd927b418211
Merge: e7298cc 6b62430
Author: anwang0000 <wang.an@columbia.edu>
Date:   Sun Apr 30 19:15:27 2017 -0400

    Merge branch 'arrayfinal' of github.com:huanyan-hny/Lava into
arrayfinal

commit e7298ccb55438a438a4f587cdb75db5923b47e4e
Author: anwang0000 <wang.an@columbia.edu>
Date:   Sun Apr 30 19:12:07 2017 -0400

    add array test

commit 6b62430765394ca33b7e5060ce8a757b2d306e7e
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Thu Apr 27 21:06:29 2017 +0100

    Added object testcases to the branch

commit bb86165a874f32b97e6cfc41240155c2394fbd13
Author: anwang0000 <wang.an@columbia.edu>
Date:   Thu Apr 27 14:04:56 2017 -0400

    add array

commit 19bb86cf9b82ed740824cece30d2a2fb2ac61d63
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Wed Apr 26 23:03:18 2017 +0100

    Nested obj access and obj param constructors pass tests. IM ON
FIRE!

commit 5cf3eeed522590b97a2f72174d52034799375f15
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Wed Apr 26 21:44:42 2017 +0100

    All testcases including object pass. ABSOFUKINGLUTELY MILESTONE

commit 998192129164230561f94ac7ef0ad0432e99d0ac
```

```
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Wed Apr 26 01:48:26 2017 +0100

    Obj access ok for some test cases. Not fully done yet. Need more
test cases

commit 7b0dc33c1fcb277befc0e2b3779d975644196343
Merge: e48717e e535969
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Mon Apr 24 19:22:58 2017 +0100

    Merged with string function and tests all passed

commit e48717ea592f0488c1603d5ed6645a9fd797bed4
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Mon Apr 24 19:07:55 2017 +0100

    add gitignore

commit 68de41c7b5978d2e561b11c0bb4271f4a5b98a77
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Mon Apr 24 19:06:43 2017 +0100

    temp commit half done obj create

commit e535969acab7a8aa50986ca6234e153700b21f1d
Merge: 3282c6c 4f37ca6
Author: Elega <445092967@qq.com>
Date:   Thu Apr 20 20:19:55 2017 -0400

     Merge branch 'master' of https://github.com/huanyan-hny/Lava

commit 3282c6c7e6afccda4a8c6dc216973a0151c89f65
Author: Elega <445092967@qq.com>
Date:   Thu Apr 20 20:18:45 2017 -0400

    add cast

commit 4f37ca6de70cd9b42e0879bd5629e0102cfefe6c
Author: huanyan <happyhn2020@icloud.com>
Date:   Thu Apr 20 16:23:21 2017 -0400

    constructor implicit this

commit 95dfd591034c1f835b1570855fcb309f3f7d7949
```

```
Author: huanyan <happyhn2020@icloud.com>
Date:   Thu Apr 20 14:24:30 2017 -0400

    Adding "this" field to constructor

commit 5a0d15cb10baa8197ef927462b39b82744a06abe
Author: Elega <445092967@qq.com>
Date:   Thu Apr 20 12:32:05 2017 -0400

    add tests for float and string

commit f3216890acfa9744d62fa9491da6092237045c70
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Thu Apr 20 16:07:55 2017 +0100

    Minor change in gitignore

commit b46ceb9700480c36abf76c26043bfeaa54166006
Author: Elega <445092967@qq.com>
Date:   Thu Apr 20 11:05:16 2017 -0400

    added files

commit bed5c8d0545dc7d930c4b9192dc4477c71a9a132
Merge: 55cef45 34ae699
Author: Elega <445092967@qq.com>
Date:   Thu Apr 20 11:01:55 2017 -0400

    merged

commit 55cef45781de36f612a87112ee90f60d75a42a3b
Author: Elega <445092967@qq.com>
Date:   Thu Apr 20 10:54:27 2017 -0400

    add float and string operations

commit 34ae6998b90d9cca051d91650a2246caf74472f1
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Thu Apr 20 15:52:07 2017 +0100

    Obj access works for simple cases

commit 94498930345c04eb3aefd0ba6048ce300a105a57
Merge: 2e11652 a704e22
Author: jiacheliu3 <jiacheliu3@gmail.com>
```

```
Date:   Thu Apr 20 15:17:27 2017 +0100

    Merge branch 'master' of https://github.com/huanyan-hny/Lava

commit 2e11652fa2eeffad6a226221c6c841c1cfec7b9d
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Thu Apr 20 15:17:17 2017 +0100

    Obj create ok, access needs testing

commit a704e2226f806d607fc88ed1f7eef55d5479b295
Author: huanyan <happyhn2020@icloud.com>
Date:   Thu Apr 20 01:06:49 2017 -0400

    constructor analyzer update

commit 184f731c60ddce203c4975dfbdf295b4a5d95b9a
Merge: 8c0803d bbcf76e
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Wed Apr 19 21:01:56 2017 +0100

    Merge branch 'master' of https://github.com/huanyan-hny/Lava

commit 8c0803d70b67480a1ff9cba677d09ff53faa227e
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Wed Apr 19 21:01:41 2017 +0100

    Obj create works

commit bbcf76e0307c8aeb947d75c7b9e4dd5f5831a62b
Author: huanyan <happyhn2020@icloud.com>
Date:   Wed Apr 19 15:58:18 2017 -0400

    Update Analyzer

commit b71d3e1c94395a12306b12e21de6b675fb437a91
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Wed Apr 19 15:39:02 2017 +0100

    Now assign works

commit 17348d8f9e19be5282e0947a46ae927b1aaa1ca2
Merge: 9d3a898 5b1aaee
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Tue Apr 18 22:23:46 2017 +0100
```

```
    merged

commit 9d3a898188cb00a22524a6c615fca8b0339453d5
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Tue Apr 18 21:07:21 2017 +0100

    Milestone changes in codegen

commit 5b1aaee0fa3db8bf5b6ea5413f0f964be94bc385
Author: huanyan <happyhn2020@icloud.com>
Date:   Tue Apr 18 12:48:32 2017 -0400

    constructor implicit appending

commit 38c7c49616c390b4d8d059465263c7030f3b31fa
Author: huanyan <happyhn2020@icloud.com>
Date:   Tue Apr 18 10:36:33 2017 -0400

    Minor change

    Revert S_Nullexpr back to S_Null in codegen

commit 720c0703b4a2444a9a32601a7c91f67b24be2e9e
Author: huanyan <happyhn2020@icloud.com>
Date:   Tue Apr 18 02:55:01 2017 -0400

    Merge codegen

    1, S_Null -> SNullexpr
    2, Signature change for S_Assign, added fake placeholder to
compile

commit 237c7f8dff88f5fcdccad01cee469925c9d27bba
Author: huanyan <happyhn2020@icloud.com>
Date:   Tue Apr 18 02:48:19 2017 -0400

    Analyzer for class

commit 111ea10ab48bd2d6b6dfb8ddda3313e4946ba160
Author: huanyan <happyhn2020@icloud.com>
Date:   Mon Apr 17 02:15:27 2017 -0400

    Placeholder for codegen to compile
```

```
commit 50c65f3101de836bbf1c89902528c7bef7c56f13
Author: huanyan <happyhn2020@icloud.com>
Date:   Mon Apr 17 02:13:16 2017 -0400

    Class Scaning

commit 0bc968aa0ab736c4432edd28a8a03cacd4f05ef7
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Fri Apr 14 21:40:07 2017 +0100

    1. New Sast
    2. In codegen, migrated to global hash tables declared at the
beginning
    3. Removed _build/ folder

commit 5a628666c9db37a0ed19e229c5265239c9b56e2c
Merge: edb002c f5e46aa
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Fri Apr 14 03:29:17 2017 +0100

    Merge branch 'master' of https://github.com/huanyan-hny/Lava

commit edb002cb27dd89f193f24eaa3cad8643c0bbf460
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Fri Apr 14 03:28:49 2017 +0100

    Updated tests. Added types to AST and SAST. Added code to codegen
but those are not used anywhere

commit f5e46aa6d06f9e7ac9b68b7a50ba514477b595d6
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Wed Apr 12 14:43:38 2017 -0400

    Create sast_sample.txt

commit 52ab4290ce76d936e2402c239191bbc6f96888e1
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Wed Apr 12 00:24:02 2017 +0100

    update gitignore

commit 4d03320173da4d54610c88a7e26834958606e0e5
Merge: 2e685a1 1b15f1c
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Wed Apr 12 00:20:23 2017 +0100
```

```
    Merge branch 'master' of https://github.com/huanyan-hny/Lava

commit 2e685a1e18c5026d7cdf4c97fedebbf10b7525aa
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Wed Apr 12 00:20:13 2017 +0100

    format analyzer

commit 91d3026005380c346e0a07f345234a5eb2219bd1
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Wed Apr 12 00:17:08 2017 +0100

    Generated samples from Dice

commit 1b15f1cdb5f3cddeb63a74f9b9b24a26beef3d4a
Author: huanyan <happyhn2020@icloud.com>
Date:   Tue Apr 11 01:01:41 2017 -0400

    Update sast for class

commit 8afebd764af064f84a89e95b663c349e49aa7fc8
Author: huanyan <happyhn2020@icloud.com>
Date:   Mon Apr 10 02:09:32 2017 -0400

    Update parser, scanner and ast for Class

commit 9a100c6cdb2f8c4cb70d9ac41376f4c8722b2bb9
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Fri Apr 7 04:47:32 2017 +0100

    test sh

commit 6fb6e14d102cf77364f976f64923fd303491a49e
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Fri Apr 7 04:44:13 2017 +0100

    rollback test file expected outputs

commit 09e6fbd64500672cc39c9978be7d588e0806f627
Merge: c72ce0d 1ac5a89
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Fri Apr 7 04:37:46 2017 +0100

    Merge branch 'master' of https://github.com/huanyan-hny/Lava
```

```
commit c72ce0dd5325ce4ba29e6dbf799f026fd37acbf3
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Fri Apr 7 04:35:51 2017 +0100

    fixed test sh error

commit 1ac5a89fcea605ae8674b4b5bd8fd600c72592cb
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Fri Apr 7 03:42:37 2017 +0100

    More cleanup

commit 7f4851a8b36a31cca15a410fa42e5d467c4ac058
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Fri Apr 7 03:36:15 2017 +0100

    Remove meaningless files

commit 4d7f30cb9968cd939cf87bbff972e833a83a3d62
Merge: 15bef27 f5520a2
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Fri Apr 7 03:34:18 2017 +0100

    Merge branch 'master' of https://github.com/huanyan-hny/Lava

commit 15bef27d65834135424314a1fc608304d517ad50
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Fri Apr 7 03:34:09 2017 +0100

    minor formatting

commit f5520a24058156cf4679f8871faf77792a516862
Author: elega <445092967@qq.com>
Date:   Thu Apr 6 20:10:07 2017 -0400

    Add files via upload

commit 6a16721ed951b65b140b2e19c663439ea238ae71
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Thu Apr 6 05:32:31 2017 +0100

    Better test reporting

commit 1c3ac8715164fdd720bcd106865665c397d41e73
```

```
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Tue Apr 4 03:33:22 2017 +0100

    Improved test cases

commit b67349a3e0c098c0ef43fd999cd0e9344b78f2a4
Author: huanyan <happyhn2020@icloud.com>
Date:   Wed Mar 29 17:44:40 2017 -0400

    Codegen

    Implemented String Assignment

commit 1d95250bb6692f4da4c43ded19035b44eb44bc50
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Wed Mar 29 00:27:03 2017 +0100

    Added testing structure and improved tests

commit 4cc24a500718b3a2ecf98eb366b236c6d3a51ed6
Author: jiacheliu3 <jiacheliu3@gmail.com>
Date:   Thu Mar 23 19:54:55 2017 +0000

    Added test case for Hello World

commit 73bdd50e050b1bf7c63cae30be5d2ef35da3716d
Author: Elega <445092967@qq.com>
Date:   Thu Mar 23 14:07:56 2017 -0400

    fisrt commit
```

5 Architecture

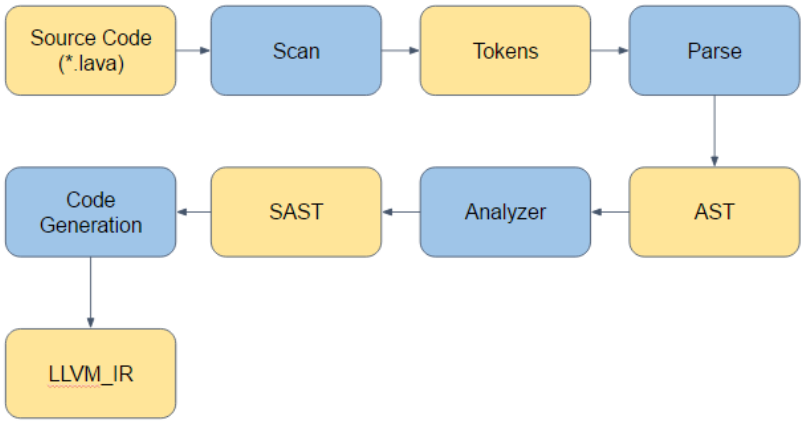
5.1 Overview

We derived the Lava compiler from the MicroC project. The compiler have a total of 7 modules.

- lava.ml: the entry pointer of the compiler
- scanner.mll: scanning the input into tokens
- ast.ml: the abstract syntax tree
- sast.ml: the semantic checked abstract syntax tree

- parser.mly: parsing tokens to a AST
- analyzer.ml: semantic checking, generating the SAST
- codegen.ml: code generation

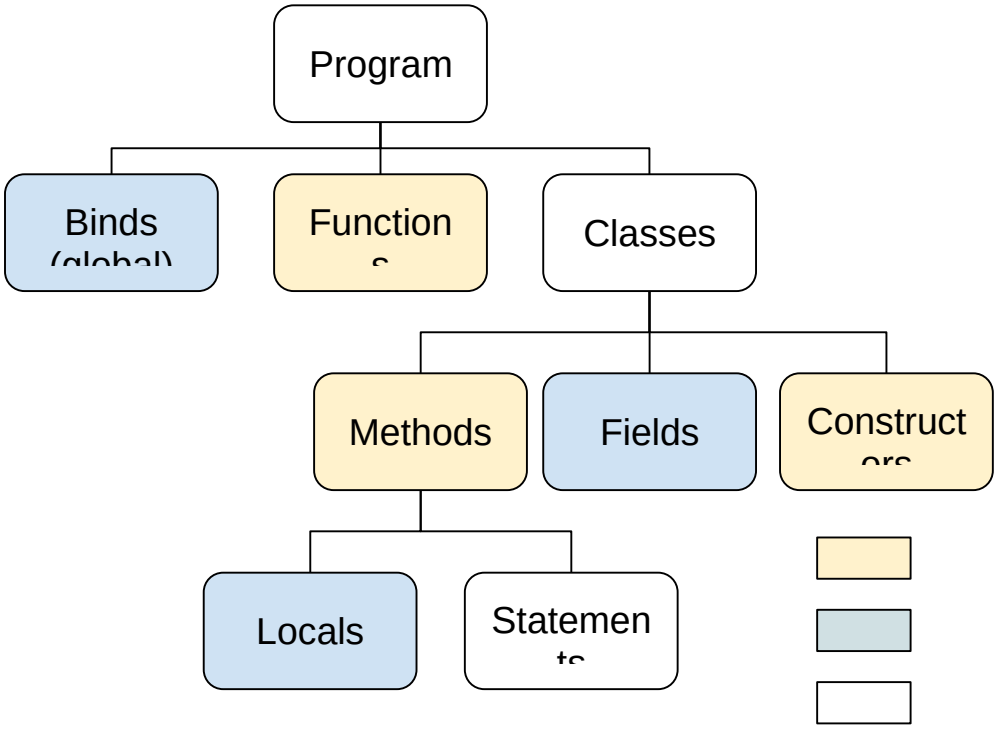
The dataflow of the compiler could be summarized by the graph below.



5.2 The scanner

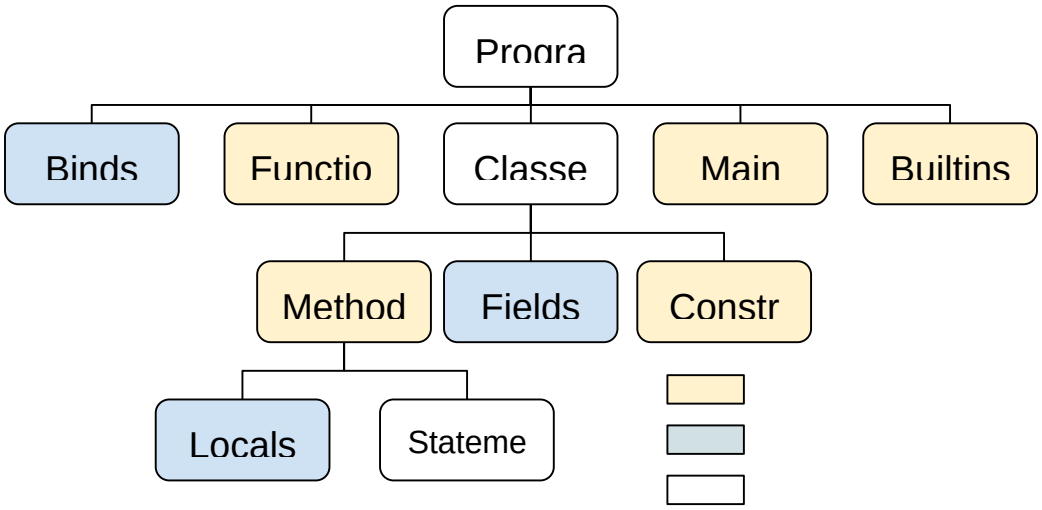
The scanner takes the lava source code, scans the input and generate tokens. Its implementation was quite trivial. We inherited the code of scanner.mll in MicroC and added some new rules for scanning strings and floats. In addition, square brackets was also added for the array feature.

5.3 The parser (Wei, Yuan and Wang)



The parser scans tokens and generate the corresponding AST. Like C++, both functions and variables could be declared globally or as members inside the class. In a class body, constructors must be defined (as no default constructor in Lava). Methods and fields are optional in the class body. Global functions, class methods and constructors are all a list of function_decl, which contains local variables and statements. Binds, class fields and locals are all a list of bind, which is the combination of a variable name and its type.

5.4 The analyzer (Wei, Yuan and Wang)



The analyzer does the semantic checking and generates a semantic checked abstract syntax tree, which is used in the code generation phase.

5.4.1 Generating the SAST (Wei)

We didn't make significant changes in our SAST. We first extract the main function out so that the code generator will be able to find the entry point of the program easier. Then we add some built-in functions to the SAST, including functions for the user (like tostring() and print()) as well as inner used only functions (like cast(), sizeof() and malloc()) for the class feature.
Then we performed an depth-first search on the AST to create one to one mappings in terms of every item in the AST to propagate an SAST. The semantic checking will be done along with the propagation process. We bound statements and expression with their types. We also replaced some expressions and statements with those which could be directly used in the code generations.

5.4.2 Class (Yuan)

There are mainly two parts in the semantic check of class, the creation of object and access of class members(fields and methods). A map of class is generated from ast before checking.

For object creation, i.e using the "new" keyword, first search the class map to check if the class is well defined, then search the associated method map of the class if found to check whether there is a matching constructor or not. If a matching constructor is found, the object creation passes the semantic check.
For access of class members, i.e using the dot operator, the class map is searched to find if there is a matching member, methods are matched first by name then by parameters. The

accessed part can also be another access of class members in pattern matching, therefore chained access is also supported.

Meanwhile, to support overloading as well as making life easier for the code generation part, for each method in a class, the class name and types of parameters are appended to the name of the method. For each constructor, an implicit object variable is declared in the beginning of the body and returned at the end of the body. For each method, an implicit object parameter is added to the parameter list so that the called can be referred in the method, and correspondingly, in each access of method, the caller will be implicitly added to the parameters of the called method.

For example, the code below on the left will be implicitly converted to the one on the right, note that the right one is only a representation of semantically checked abstract syntax tree, it is not a valid lava code.

5.5 The code generator (Wei, Wang and Liu)

5.5.1 Basics (Liu)

From the SAST, we generate the code in order, from globals to classes to functions.

```
let codegen_sprogram (program:S.s_program) =
  (* Build builtin function declarations *)
  let _ = codegen_builtins program.S.builtins in

  let _ = codegen_globals program.S.global_vars in

  let _ = codegen_classes program.S.classes in

  (* Generate functions here *)
  let _ = codegen_function_decls program.S.functions in
  let _ = codegen_function_decls (program.S.main::[]) in
  let _ = codegen_fbody program.S.functions in
  let _ = codegen_fbody (program.S.main::[]) in

  the_module
```

First we generate builtin functions, such as printf, malloc, sizeof etc. These functions will be used underlying. For example string concatenation is done by calling to strcat function.

```
and codegen_builtins func_decls =
  ignore(log_to_file "Codegen builtins\n");
  (* Declare printf(), which the print built-in function will call *)
  let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let printf_func = L.declare_function "printf" printf_t the_module in

  (* Declare malloc *)
  let malloc_ty = L.function_type (i8_pt) [| i32_t |] in
  let malloc_func = L.declare_function "malloc" malloc_ty the_module in

  (* Declare sprintf *)
  let sprintf_ty = L.var_arg_function_type i32_t [| L.pointer_type i8_t; L.pointer_type i8_t  |] in
  let sprintf_func = L.declare_function "sprintf" sprintf_ty the_module in

  (* Declare strcpy *)
  let strcpy_ty = L.function_type(L.pointer_type(i8_t)) [| L.pointer_type i8_t; L.pointer_type i8_t  |] in
  let strcpy_func = L.declare_function "strcpy" strcpy_ty the_module in

  (* Declare strlen *)
  let strlen_ty = L.function_type i32_t [| L.pointer_type i8_t; |] in
  let strlen_func = L.declare_function "strlen" strlen_ty the_module in

  (* Declare strcat *)
  let strcat_ty = L.function_type(L.pointer_type(i8_t)) [| L.pointer_type i8_t; L.pointer_type i8_t  |] in
  let strcat_func = L.declare_function "strcat" strcat_ty the_module in
```

Then we build the globals.
After that, we generate classes.

```
and codegen_classes (classes:S.s_class_decl list) =
  ignore(log_to_file "Codegen classes\n");
  ignore(List.map (fun c -> log_to_file ("To build class "^c.S.s_cname^"\n") ) classes);

  let codegen_one_class (the_class:S.s_class_decl) =
    ignore(log_to_file ("Generating class "^the_class.s_cname^"\n") );

    (* Generate struct decl *)
    let _ = codegen_struct_stub the_class in

    (* Generate class local vars *)
    let _ = codegen_struct the_class in

    (* Generate functions *)
    ignore(log_to_file "Finished class definition. Now start with functions\n");
    ignore(List.map (fun f -> log_to_file ("To build constructor "^f.S.s_fname^"\n") ) the_class.s_constructors);
    ignore(List.map (fun f -> log_to_file ("To build method "^f.S.s_fname^"\n") ) the_class.s_methods);

    let _ = codegen_function_decls the_class.s_constructors in
    let _ = codegen_function_decls the_class.s_methods in
    let _ = codegen_fbody the_class.s_constructors in

    let _ = codegen_fbody the_class.s_methods in
    ignore(log_to_file ("Generated class "^the_class.s_cname^"\n") );
  in
  List.map codegen_one_class classes;
  ignore(log_to_file "Finished generating classes\n")
```

We first generate the class variables then functions one by one.
Finally we generate global functions, including the main function, just as how MicroC works.

5.5.2 Array (Wang)

Basically,there are several functions works on implementing array, such as initialise_array, generate_one_d_array, generate_array_access and others.

```
and initialise_array arr arr_len init_val start_pos llbuilder =
  let new_block label =
    let f = L.block_parent (L.insertion_block llbuilder) in
    L.append_block (context) label f (* global_context () 还是 context ? *)
  in
  let bbcurr = L.insertion_block llbuilder in
  let bbcond = new_block "array.cond" in
  let bbbody = new_block "array.init" in
  let bbdone = new_block "array.done" in
  ignore (L.build_br bbcond llbuilder);
  L.position_at_end bbcond llbuilder;

  (* Counter into the length of the array *)
  let counter = L.build_phi [L.const_int i32_t start_pos, bbcurr] "counter"
  add_incoming ((L.build_add counter (L.const_int i32_t 1) "tmp" llbuilder),
  let cmp = build_icmp Icmp.Slt counter arr_len "tmp" llbuilder in
  ignore (L.build_cond_br cmp bbbody bbdone llbuilder);
  position_at_end bbbody llbuilder;

  (* Assign array position to init_val *)
  let arr_ptr = build_gep arr [| counter |] "tmp" llbuilder in
  ignore (build_store init_val arr_ptr llbuilder);
  ignore (build_br bbcond llbuilder);
  position_at_end bbdone llbuilder

(* Generate 1 dimension array *)
and generate_one_d_array typ size builder =
  let t = ltype_of_typ typ in

  let size = (L.const_int i32_t size) in

  let size_t = L.build_intcast (L.size_of t) i32_t "1tmp" builder in (* 类型的

  let size = L.build_mul size_t size "2tmp" builder in  (* 类型的size 乘以 数组
  let size_real = L.build_add size (L.const_int i32_t 1) "arr_size" builder

  let arr = L.build_array_malloc t size_real "333tmp" builder in
  let arr = L.build_pointercast arr (L.pointer_type t) "4tmp" builder in

  let arr_len_ptr = L.build_pointercast arr (L.pointer_type i32_t) "5tmp" bu

  ignore(L.build_store size_real arr_len_ptr builder);
  initialise_array arr_len_ptr size_real (L.const_int i32_t 0) 0 builder;
  arr
```

It also involved of handling condition of S_ArrayAccess several times like the below one.

```
| S.S_ArrayAccess(e, index, typ) ->
  let vmemory = generate_array_access false e index llbuilder in
  let value = codegen_sexpr llbuilder false rhs in
  ignore (L.build_store value vmemory llbuilder);
  value
| _ ->
  let e1' = codegen_sexpr llbuilder false lhs and e2' = codegen_sexpr llbuild
  ignore (L.build_store e2' e1' llbuilder); e2'
```

One of the key points is to assign memory for the array and compute right offset for each element in the array. The related detail would be different for divergent data type like int and string.

Especially, to implement array in an object, the code generating class object should also be modified. Basically the array data would be in local table, so after assign class attributes to a variable we should look for this data in corresponding table.

Also, array can be realized with C language, here I just chose the LLVM way. Actually LLVM provides a relatively nice service for array implementation. For example, we don't have to pay attention to the layout and padding problem mentioned in the class.

5.5.3 Class (Liu)

Class are generated by using C structs as underlying implementation.

```
(* Generate classes *)
and  codegen_struct_stub class_struct =
  let struct_t = L.named_struct_type context class_struct.S.s_cname in
  H.add object_types class_struct.S.s_cname struct_t


and codegen_struct class_struct =
  let struct_t = H.find object_types class_struct.S.s_cname in
  let type_list = List.map (fun (data_type,name) -> ltype_of_typ data_type) class_struct.S.s_fields in
  let name_list = List.map (fun (data_type, name) -> name) class_struct.S.s_fields in

  let type_array = (Array.of_list type_list) in
  List.iteri (fun index field ->
      let n = class_struct.S.s_cname ^ "." ^ field in
      H.add object_field_indices n index;
    ) name_list;
  L.struct_set_body struct_t type_array true
```

Under the hood, we just create a struct with all class fields embedded.
All class functions are not built into the struct. Instead, their names are translated to [className].[functionName] and all built as global functions. All the function calls are translated to these global function calls, in the Analyzer step.
Class functions invoked on an object are also translated to global function calls in the Analyzer, with the first parameter now being the caller object itself.
Every time a class object is created by a constructor call, as introduced in the Analyzer section, we implicitly malloc one chunk of memory of the size of the class struct. We cast the pointer to the struct type and assign that to the object reference.

```
(* Pass constructor to this function *)
and codegen_obj_create fname expr_list data_type llbuilder =
  (* Call the full name of the function *)
  let class_name = match data_type with
      A.Object(obj_name) -> obj_name
    | d -> raise (Failure ("Data type in Obj_Create is not an object but "^(A.string_of_typ d) ) )
  in
  let string_of_params params =
    String.concat "." (List.map (fun p -> (S.string_of_expr_type p ) ) params )
  in
  let param_str = (match expr_list with
        [] -> ""
      | _ -> "." ^ (string_of_params expr_list)
    )in
  let full_name = (class_name ^ ".constructor"^param_str) in
  let f = func_lookup full_name in
  let generate_param_func builder isReturn param =
    match param with
      S.S_Id(id,A.Object(obj_name)) ->
      lookup id
    | _ ->
      codegen_sexpr llbuilder false param
  in
  let params = List.map (generate_param_func llbuilder false) expr_list in
  let obj = L.build_call f (Array.of_list params) "tmp" llbuilder in
  obj
```

Every time a field is access in an object, we find the index of the field, and access the field by calling to the index.

```
L.build_struct_gep parent_expr field_index id llbuilder
```

5.5.4 String manipulation and casting (Wei)

In Lava, three operators are overloaded for string. The plus operator for string concatenation and the double equal and the not equal operators for string comparison. We implement the string comparison by replacing the double equal operator with the strcmp() function. Note that because the strcmp() function will return 0 rather than 1 if two string have the same value, we need to compare the return value to zero outside the function invocation.

```
| Equal when (type1 = type2 && type1 = String) ->
S_Binop(S_Literal(0),Equal,S_Call("strcmp",[sexpr1;sexpr2;],Int),Bool)
| Neq when (type1 = type2 && type1 = String) ->
S_Unop(Not,S_Binop(S_Literal(0) ,Equal,S_Call("strcmp",[sexpr1;sexpr2;],Int),Bool),Bool)
```

The implementation of the string concatenation is the most trick one. Although we are able to use strcat() to concatenate two strings, it will change the value of the first string instead of returning a new one. The behavior of the plus operator should be like taking two strings and return the concatenation of this two.

```
char* operator + (char* lhs, char* rhs) {
    int l1 = strlen(lhs);
    int l2 = strlen(rhs);
    int len = l1 + l2 + 1;
    char* buf = (char*)malloc(len);
    strcpy(buf, lhs);
    strcat(buf, rhs);
    return buf;
}
```

The pseudo code above shows our implementation of the plus operator. First we get the length of the two strings, sum them up and add it by one, which is the length of the string we return. Then we allocate a piece of memory for the new string. strcpy() is used to copy the first string to the new string. At last strcat() is invoked to concatenate the new string with the second string. We implemented this piece of pseudo code in ocaml, invoking the llvm instructions one by one.

```
and codegen_strcat e1' e2' llbuilder =
  let ll_expr1 = e1' in
  let ll_expr2 = e2' in
  let strlen_func = func_lookup "strlen" in
  let strcpy_func = func_lookup "strcpy" in
  let strcat_func = func_lookup "strcat" in
  let len1 =  L.build_call strlen_func [| (ll_expr1); |] "strlen" llbuilder in
  let len2 =  L.build_call strlen_func [| (ll_expr2); |] "strlen" llbuilder in
  let len_new = L.build_add len1 len2 "tmp" llbuilder in
  let size = L.build_add len_new (L.const_int i32_t 1) "tmp" llbuilder in
  let t = i8_t in
  let buf = L.build_array_malloc t size "to_string_buf" llbuilder in
  let buf = L.build_pointercast buf (L.pointer_type t) "to_string_buf" llbuilder in
  let buf = L.build_call strcpy_func [| buf; ll_expr1; |] "strcpy" llbuilder in
  let buf = L.build_call strcat_func [| buf; ll_expr2; |] "strcat" llbuilder in
  (* free *)
  buf
```

In Lava, primitives are able to be casted from one to another (See the conversion matrix in LRM). Among all of them, castings between int, float and bool are simple as LLVM provides instructions to deal with that. However, on the other hand, string related castings have to be carried out by standard c library functions.

We used atoi() and atof() to cast a string to an integer (or float). These functions will return 0 if the parsing fails, otherwise, the numeric value of the string will be returned. On the other hand, we used sprintf() to cast a number to a string. We intended to use itoa() and ftoa() to do that but unfortunately they are not included in the standard c library. We called malloc to assign an buffer array for the string and then called sprintf to cast the number to the string. As we've bound type with statements in our SAST, we are able to select different sprint format string for different numeric types. After the invocation of sprintf, we simply return.

```
let t = i8_t in
let buf = L.build_array_malloc t size "to_string_buf" llbuilder in
let buf = L.build_pointercast buf (L.pointer_type t) "to_string_buf" llbuilder in
let sprintf_func = func_lookup "sprintf" in
let _ = L.build_call sprintf_func [| buf; sprintf_format ; ll_expr; |]
    "sprintf" llbuilder in
buf
```

5.5.5 Function overloading (Yuan and Liu)

Whenever two functions have the same name, we let them exist at the same time but distinguish them by their parameters. For example, all the constructor functions in a class are named "constructor".

```
class Node{
    int value;
    string label;
    class Node left;
    class Node right;

    constructor(int v, string n){
        value = v;
        label = n;
    }
    constructor(class Node l, class Node r, int v, string n){
        value = v;
        left = l;
        right = r;
        label = n;
    }
```

We tell the functions from one another during the analyzer part. This is done by concatenating the function name with the parameter types. In the generated SAST, these two functions are translated into [className].[functionName].[param1Type].[param2Type]... For a global function the class name part is simply not there.

```
class Node {
Node right;
Node left;
string label;
int value;
Node Node.constructor.Node.Node.int.string(Node l, Node r, int v, string n)
{
Node this;
{$this[Node] = cast(malloc(sizeof($this[Node])[int])[any])[Node][Node]} [Node];
{{Object Access: $this[Node].$value[int] int} = $v[int][int]} [int];
{{Object Access: $this[Node].$left[Node] Node} = $l[Node][Node]} [Node];
{{Object Access: $this[Node].$right[Node] Node} = $r[Node][Node]} [Node];
{{Object Access: $this[Node].$label[string] string} = $n[string][string]} [string];
return $this[Node];
[Node]
}
Node Node.constructor.int.string(int v, string n)
{
Node this;
{$this[Node] = cast(malloc(sizeof($this[Node])[int])[any])[Node][Node]} [Node];
{{Object Access: $this[Node].$value[int] int} = $v[int][int]} [int];
{{Object Access: $this[Node].$label[string] string} = $n[string][string]} [string];
return $this[Node];
[Node]
}
```

In this way we don’t need much effort in the code generation step, as the function calls all call on their distinct names.

6 Testing

6.1 Manual Testing

As we did not find any unit testing suites like JUnit, it’s hard to do unit testing on every single function in our code. And the testing is done mostly by printing manually. This has brought us

some inconvenience, but with careful code control and frequent regression testing, we succeeded to ensure that running code runs.

The AST and SAST are tested by printing as well. We just manually check if the generated tree for our test cases match our expectation.

In fact, although not as convenient as automated unit tests, testing merely by hand only caused us trouble in locating the bug, not in ensuring the quality of code. We implemented integration tests with much care and by ensuring that they all pass, we achieve the same level of confidence in the functionality.

6.2 Integration Testing

By integration testing it means a test passes only if the whole chain works. For example, an AST can be printed only if all functions involved from parsing to tree generation work as expected. One piece of our Lava code can generate the expected result only if the chain from the parser to codegen do the job correctly.
This is our focus in testing for a few reasons:

1. We are provided with the test suite of MicroC. It's convenient and intuitive to work on it.
2. It turned out that making sure the chain works did not bring excessive overhead, with careful segregation of code.
3. The tests can be easily done with diff.

6.2.1 Regression Testing procedure

We use the tests as regression testing procedure and run them every time we add or modify a function. If errors emerge, we modify the function or completely rewind and restart. Every time we commit and push we also have to make sure the pushed tests all pass, meaning all expected features work. In this way, like what DB people do, we maintain the atomicity and durability of our changes.

6.2.2 Automated Testing

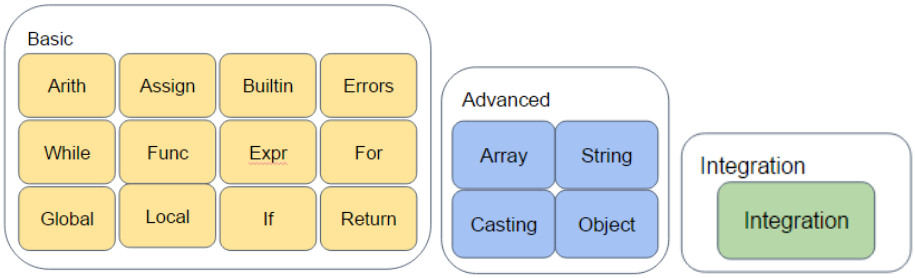
We improved the testing bash script we were provided. Since the number of tests increment fast and it became hard to trace each. We improved on the color and structure of the test log. The green output can be ignored safely as things go well as expected. Every time the test is run we can only locate and read the red lines.
It turned out to be greatly helpful after there being more than 100 positive and negative test cases. As we run the tests very frequently this saved a lot of time and effort.

```
@ubuntu: ~/Desktop/Link to Github/Lava
Testing in tests/func
Put temp files in tests/func/testrun
Files to work on: tests/func/test-*.mc tests/func/fail-*.mc
test-add1...OK
test-func1...OK
test-func2...OK
test-func3...OK
test-func4...OK
test-func5...OK
test-func6...OK
test-func7...OK
test-func8...OK
test-func_noArgFunc...OK
test-func_stringArg...OK
test-func_touchGlobalVar...OK
fail-func1...Failure as expected
OK
fail-func2...Failure as expected
OK
fail-func3...Failure as expected
OK
fail-func4...Failure as expected
OK
fail-func5...Failure as expected
OK
fail-func6...Failure as expected
OK
fail-func7...Failure as expected
OK
fail-func8...Failure as expected
OK
fail-func9...Failure as expected
OK
fail-func_assignInArg...Failure as expected
OK
fail-func_needStringGivenInt...Failure as expected
OK
```

```
tassadar@osboxes:~/PLT/Laaaava/Lava$ ./testall.sh
PWD: /home/tassadar/PLT/Laaaava/Lava

Testing in tests/arith
Put temp files in tests/arith/testrun
Files to work on: tests/arith/test-*.mc tests/arith/fail-*.mc
test-arith1...FAILED
  ./microc.native failed on ./microc.native < tests/arith/test-arith1.mc > tests/arith/testrun/test-arith1.ll
test-arith2...FAILED
  ./microc.native failed on ./microc.native < tests/arith/test-arith2.mc > tests/arith/testrun/test-arith2.ll
test-arith3...FAILED
  ./microc.native failed on ./microc.native < tests/arith/test-arith3.mc > tests/arith/testrun/test-arith3.ll
test-arith_mulAndDiv...FAILED
  ./microc.native failed on ./microc.native < tests/arith/test-arith_mulAndDiv.mc > tests/arith/testrun/test-arith_mulAndDiv.ll
test-float_arith1...FAILED
  ./microc.native failed on ./microc.native < tests/arith/test-float_arith1.mc > tests/arith/testrun/test-float_arith1.ll
test-float_arith2...FAILED
  ./microc.native failed on ./microc.native < tests/arith/test-float_arith2.mc > tests/arith/testrun/test-float_arith2.ll
test-float_arith3...FAILED
  ./microc.native failed on ./microc.native < tests/arith/test-float_arith3.mc > tests/arith/testrun/test-float_arith3.ll
```

6.2.3 Test structure



As the number of tests grow fast, we categorize them into directories. As illustrated above.

```
tassadar@osboxes:~/PLT/Lava/tests$ ls
arith  array  assign  builtin  cast  errors  expr  for  func  global  if  integration  local  object  return  string  while
```

The test script will find every folder under tests/ folder, and recursively run all the positive and negative test cases under them. The log and errors will also be kept under testrun/ under that folder for us to trace in case a test case doesn't pass.

```
tassadar@osboxes:~/PLT/Lava/tests$ ls object/
testall.log                  test-class_inClassIf.ll      test-class_nestedObj.out~          test-class_stringField.mc
test-class_constructor1.mc~  test-class_inClassIf.mc      test-class_noParamConstructor.mc   test-class_stringField.out
test-class_constructor2.mc~  test-class_inClassIf.out     test-class_noParamConstructor.out  test-class_stringFunc.mc
test-class_func.mc           test-class_inClassWhile.ll   test-class_objArrayField.mc        test-class_stringFunc.out
test-class_func.out          test-class_inClassWhile.mc   test-class_objArrayField.out       test-class_withParamsConstructor.mc
test-class_funcWithThis.mc   test-class_inClassWhile.out  test-class_objAsParam.mc           test-class_withParamsConstructor.out
test-class_funcWithThis.mc~  test-class_nestedAccess.mc   test-class_objAsParam.out          test-class_withParamsConstructor.out~
test-class_funcWithThis.out  test-class_nestedAccess.out  test-class_printObjAccess.mc       testrun
test-class_inClassFor.ll     test-class_nestedObj.mc      test-class_printObjAccess.out
test-class_inClassFor.mc     test-class_nestedObj.mc~     test-class_recursion.mc
test-class_inClassFor.out    test-class_nestedObj.out     test-class_recursion.out
tassadar@osboxes:~/PLT/Lava/tests$ ls object/testrun/
test-class_constructor1.diff  test-class_inClassFor.out     test-class_nestedObj.ll             test-class_printObjAccess.diff
test-class_constructor1.ll    test-class_inClassIf.diff     test-class_nestedObj.out            test-class_printObjAccess.ll
test-class_constructor1.out   test-class_inClassIf.ll       test-class_noParamConstructor.diff  test-class_printObjAccess.out
test-class_constructor2.ll    test-class_inClassIf.out      test-class_noParamConstructor.ll    test-class_stringField.diff
test-class_func.diff          test-class_inClassWhile.diff  test-class_noParamConstructor.out   test-class_stringField.ll
test-class_func.ll            test-class_inClassWhile.ll    test-class_objArrayField.ll         test-class_stringField.out
test-class_func.out           test-class_inClassWhile.out   test-class_objAsParam.diff          test-class_stringFunc.diff
test-class_funcWithThis.diff  test-class_nestedAccess.diff  test-class_objAsParamInObjFunc.mc   test-class_stringFunc.ll
test-class_funcWithThis.ll    test-class_nestedAccess.ll    test-class_objAsParamInObjFunc.out  test-class_stringFunc.out
test-class_funcWithThis.out   test-class_nestedAccess.out   test-class_objAsParam.ll            test-class_withParamsConstructor.diff
test-class_inClassFor.diff    test-class_nested.ll          test-class_objAsParam.mc            test-class_withParamsConstructor.ll
test-class_inClassFor.ll      test-class_nestedObj.diff     test-class_objAsParam.out           test-class_withParamsConstructor.out
```

Finally we had more than 150 test cases, running them all takes around 5~10s.

6.2.4 Positive and negative test cases

For each test we try to make its name meaningful, which saves time when we see a test fails. We try to write one test case for each branch of logic decision, like whether the object has another object as field, an array as field, and whether the constructor has parameters. Whenever we add a feature, we also try to make sure it works well with existing features. For example, we make sure that in class functions, if, for and while blocks will also work. Besides the features that we expect to work, there are features that we expect to produce an error. For example, we expect the compiler not to generate code when the parameters passed to a function doesn't meet its definition. Moreover we expect it to give us a meaningful message feedback telling us what is wrong. This is achieved by adding negative test cases. As shown below, following the style in MicroC, all positive test cases start with "test" and all negative ones start with "fail".

```
tassadar@osboxes:~/PLT/Lava/tests$ ls func/
fail-func1.err  fail-func5.err  fail-func9.err                   testall.log     test-func4.out  test-func8.out
fail-func1.mc   fail-func5.mc   fail-func9.mc                    test-func1.mc   test-func5.mc   test-func_noArgFunc.mc
fail-func2.err  fail-func6.err  fail-func_assignInArg.err        test-func1.out  test-func5.out  test-func_noArgFunc.out
fail-func2.mc   fail-func6.mc   fail-func_assignInArg.mc         test-func2.mc   test-func6.mc   test-func_stringArg.mc
fail-func3.err  fail-func7.err  fail-func_needStringGivenInt.err  test-func2.out  test-func6.out  test-func_stringArg.out
fail-func3.mc   fail-func7.mc   fail-func_needStringGivenInt.mc  test-func3.mc   test-func7.mc   test-func_touchGlobalVar.mc
fail-func4.err  fail-func8.err  test-add1.mc                     test-func3.out  test-func7.out  test-func_touchGlobalVar.out
fail-func4.mc   fail-func8.mc   test-add1.out                    test-func4.mc   test-func8.mc   testrun
```

7 Lessons Learned

Yimin Wei

The project is really painful. But we learned bunches of stuff from this course. I've learned the importance of making the development plan. During our development, there was always a question which made us struggling: should we continue using the microc skeleton code or build a totally new one. We didn't discuss this problem with each other and everyone made every effort to figure out how to put more stuffs into the microc. However, we ended up with rewriting

the semantic checker and code generator because we began to realize it seemed to be less and less possible to put our new features to the microc. After that, whenever we were going to develop some new language features, we will spend enough time discussing with our team member. We analyzed the feasibility of the idea, argued with design choices. We didn't get started until we reached a consensus on how to do it. From then on, our project was progressed smoothly. To sum up, my biggest gain from this class is to know whatever we are going to do, we should share the idea with team members and discuss on that. We shall never do it ourselves.

Hongning Yuan

For a semester long project, communicating and scheduling is always a crucial part. Your teammates may be talented but may also be busy and not motivated enough for the long and tedious project. A good communication between team members can help the team to avoid the unnecessary errors, and a good scheduling and task separation can significantly increase the efficiency. And manager should ensure good communication and schedule in a team. This is also the first time that I learned a leader's power comes with responsibilities.

An Wang

For this project, personally I just implemented the Array feature. Fortunately, to realize array feature, I need to work through all compile files involved in this project, such as scanner, parser, ast, analyser, sast and codegen. This helps me understand how does each file work and combine together and give me a better understanding of other parts as well. To be honest, plt project is one of the most interesting projects I have ever done. But I didn't do too much work in parts other than array, and this is really a pity. In the end, thanks to my teammates. This is a really good course, I recommend every student who has seldom CS experience like me to take this course.

Jiacheng Liu

From this project I learned two things. First, a good segregation of duty saves life. It's a lot of pain merging code with people. Especially, Ocaml uses a lot of let...in… blocks which can easily mess up the whole thing. A good coding style and frequent refactor helps, but only when you can read and understand what you are doing. Making sure that each person works on one file at a time helps greatly and have prevented amounts of conflicts. The other thing is the importance of an automated regression testing suite. It cannot be more important that the newly added code chunk does not corrupt the workspace and incur cascading rollbacks.

8 Code

8.1 The compiler

The primary contributors of each module are marked in bold.

Makefile

```
# Make sure ocamlbuild can find opam-managed packages: first run
#
# eval `opam config env`

# Easiest way to build: using ocamlbuild, which in turn uses ocamlfind

.PHONY : lava.native

lava.native :
	ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis,logs  -cflags -w,+a-4 \
		lava.native

# "make clean" removes all generated files

.PHONY : clean
clean :
	ocamlbuild -clean
	rm -rf testall.log *.diff lava scanner.ml parser.ml parser.mli
	rm -rf *.cmx *.cmi *.cmo *.cmx *.o
	rm -rf *.native

# More detailed: build using ocamlc/ocamlopt + ocamlfind to locate LLVM

OBJS = ast.cmx codegen.cmx parser.cmx scanner.cmx semant.cmx lava.cmx

lava : $(OBJS)
	ocamlfind ocamlopt -linkpkg -package llvm -package llvm.analysis $(OBJS) -o lava

scanner.ml : scanner.mll
	ocamllex scanner.mll

parser.ml parser.mli : parser.mly
	ocamlyacc parser.mly

%.cmo : %.ml
	ocamlc -c $<

%.cmi : %.mli
	ocamlc -c $<
```

```
%.cmx : %.ml
	ocamlfind ocamlopt -c -package llvm $<

### Generated by "ocamldep *.ml *.mli" after building scanner.ml and parser.ml
analyzer.cmo : sast.cmo ast.cmo
analyzer.cmx : sast.cmx ast.cmx
ast.cmo :
ast.cmx :
codegen.cmo : ast.cmo
codegen.cmx : ast.cmx
lava.cmo : semant.cmo codegen.cmo ast.cmo
lava.cmx : semant.cmx codegen.cmx ast.cmx
sast.cmo : ast.cmo
sast.cmx : ast.cmx
semant.cmo : ast.cmo
semant.cmx : ast.cmx

# Building the tarball

TESTS = add1 arith1 arith2 arith3 fib for1 for2 func1 func2 func3 \
    func4 func5 func6 func7 func8 gcd2 gcd global1 global2 global3       \
    hello if1 if2 if3 if4 if5 local1 local2 ops1 ops2 var1 var2          \
    while1 while2

FAILS = assign1 assign2 assign3 dead1 dead2 expr1 expr2 for1 for2 \
    for3 for4 for5 func1 func2 func3 func4 func5 func6 func7 func8       \
    func9 global1 global2 if1 if2 if3 nomain return1 return2 while1      \
    while2

TESTFILES = $(TESTS:%=test-%.mc) $(TESTS:%=test-%.out) \
           $(FAILS:%=fail-%.mc) $(FAILS:%=fail-%.err)

TARFILES = ast.ml codegen.ml Makefile lava.ml parser.mly README scanner.mll \
	semant.ml testall.sh $(TESTFILES:%=tests/%)

lava-llvm.tar.gz : $(TARFILES)
	cd .. && tar czf lava-llvm/lava-llvm.tar.gz \
	       $(TARFILES:%=lava-llvm/%)
```

Lava.ml (Yuan)

```
open Sast
(* Top-level of the MicroC compiler: scan & parse the input,
   check the resulting AST, generate LLVM IR, and dump the module *)

type action = Ast | Sast | LLVM_IR | Compile
```

```
let _ =
  let action = if Array.length Sys.argv > 1 then
    List.assoc Sys.argv.(1) [ ("-a", Ast);        (* Print the AST only *)
                  ("-s", Sast); (* Print the SAST only *)
                              ("-l", LLVM_IR);  (* Generate LLVM, don't check *)
                              ("-c", Compile) ] (* Generate, check LLVM IR *)
  else Compile in
  let lexbuf = Lexing.from_channel stdin in
  let ast = Parser.program Scanner.token lexbuf in
  let sast = Analyzer.analyze ast in
  let (globals, function_decls, class_decls, main_decl, builtin_decls) = sast in
  let program = {
                global_vars = globals;
                functions = function_decls;
    classes = class_decls;
    main = main_decl;
    builtins = builtin_decls;
        }
  in
  match action with
    Ast -> print_string (Ast.string_of_program ast)
  | Sast -> print_string (Sast.string_of_s_program sast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.codegen_sprogram program))
  | Compile -> let m = Codegen.codegen_sprogram program in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)
```

Scanner.mly (Wei, Wang and Yuan)

```
(* Ocamllex scanner for Lava *)

{ open Parser }

let whitespace = [' ' '\t' '\r' '\n']
let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let escape = '\\' ['\\' ''' '"' 'n' 'r' 't']
let escape_char = ''' (escape) '''
let alpha = ['a'-'z' 'A'-'Z']
let digit = ['0'-'9']
let int = digit+
let float = (digit+) '.' (digit+)

let char = ''' (ascii) '''

let string_lit = '"'((ascii|escape)* as lxm)'"'

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
```

```
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| '['      { LBRACKET }
| ']'      { RBRACKET }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }
| '.'     { DOT }

(*Flow control*)
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }

(*Data types*)
| "int"    { INT }
| "bool"   { BOOL }
| "void"   { VOID }
| "float"  { FLOAT }
| "string" { STRING }
| "float"  { FLOAT }
| "true"   { TRUE }
| "false"  { FALSE }

(*Class*)
| "class"  { CLASS }
| "new"    { NEW }
| "delete" { DELETE }
| "constructor" { CONSTRUCTOR }

| ['0'-'9']+ as lxm { LITERAL(int_of_string lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| string_lit { STRING_LITERAL(lxm)}
| float as lxm  { FLOAT_LITERAL(float_of_string lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }
```

```
and comment = parse
  "*/" { token lexbuf }
| _    { comment lexbuf }
```

Ast.ml (Wei, Wang and Yuan)

```
(* Abstract Syntax Tree and functions for printing it *)

type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq |
          And | Or

type uop = Neg | Not

type typ = Int | Bool | Void | Float | Null | String | Object of string | Arraytype of typ |
Constructortyp | Any

type bind = typ * string

type expr =
    Literal of int
  | BoolLit of bool
  | StringLit of string
  | FloatLit of float
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of expr * expr
  | Call of string * expr list
  | ObjAccess of expr * expr
  | ObjCreate of string * expr list
  | ArrayCreate of typ * int
  | ArrayAccess of expr * int
  | Delete of expr
  | Noexpr
  | Nullexpr
  | Cast of typ * expr

type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt

type func_decl = {
    typ : typ;
    fname : string;
    formals : bind list;
```

```
    locals : bind list;
    body : stmt list;
  }

type cbody = {
    fields: bind list;
    methods: func_decl list;
    constructors: func_decl list;
}

type class_decl = {
    cname: string;
    cbody: cbody;
}

type program = bind list * func_decl list * class_decl list

(* Pretty-printing functions *)
let rec string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Void -> "void"
  | String -> "string"
  | Float -> "float"
  | Null -> "null"
  | Object(cname) -> cname
  | Arraytype(t) -> "array of " ^ string_of_typ t
  | Constructortyp -> "constructortyp"
  | Any -> "any"

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
```

```
  | StringLit(s) -> s
  | FloatLit(f) -> string_of_float f
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(e1, e2) -> string_of_expr e1 ^ " = " ^ string_of_expr e2
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | ObjCreate(s,el) -> "new " ^ s ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^
")"
  | ObjAccess(e1,e2) -> string_of_expr e1 ^ "." ^ string_of_expr e2
  | ArrayCreate(typ, len) -> "(array of type " ^ string_of_typ typ ^ " with length " ^
string_of_int len ^ ")"
  | ArrayAccess(arrayName, index) -> "(array name: " ^ string_of_expr arrayName ^ " index: "
^ string_of_int index ^ ")"
  | Delete(e) -> "delete" ^ string_of_expr e
  | Cast (t,e) -> "(" ^ (string_of_typ t) ^ ") " ^ (string_of_expr e)
  | Noexpr -> ""
  | Nullexpr -> "null"

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s

let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

let string_of_formal (t, id) = string_of_typ t ^ " " ^ id

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map string_of_formal fdecl.formals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

let string_of_cdecl cdecl =
    "class" ^ " " ^ cdecl.cname ^ " { \n" ^
    String.concat "" (List.map string_of_vdecl cdecl.cbody.fields) ^
    String.concat "" (List.map string_of_fdecl cdecl.cbody.constructors) ^
    String.concat "" (List.map string_of_fdecl cdecl.cbody.methods) ^
    "}\n"
```

```
let string_of_program (vars, funcs, classes) =
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_cdecl classes) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)
```

Parser.mly (Wei, Wang and Yuan)

```
%{ open Ast %}

%token CLASS CONSTRUCTOR NEW DELETE DOT
%token SEMI LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET COMMA
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token RETURN IF ELSE FOR WHILE INT BOOL VOID FLOAT STRING
%token <int> LITERAL
%token <string> STRING_LITERAL
%token <float> FLOAT_LITERAL
%token <string> ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right RPAREN
%right ASSIGN
%left LBRACKET
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%right NOT NEG DOT DELETE

%start program
%type <Ast.program> program

%%

program:
    decls EOF { $1 }

decls:
   /* nothing */ { [], [], [] }
 | decls vdecl { let (vdecl,fdecl,cdecl) = $1 in ($2::vdecl,fdecl,cdecl) }
 | decls fdecl { let (vdecl,fdecl,cdecl) = $1 in (vdecl,$2::fdecl,cdecl) }
 | decls cdecl { let (vdecl,fdecl,cdecl) = $1 in (vdecl,fdecl,$2::cdecl) }
```

```
cdecl:
        CLASS ID LBRACE cbody RBRACE { {
            cname =  $2;
            cbody =  $4;
        } }

cbody:
        /* nothing */ { {
            fields = [];
            methods = [];
            constructors = [];
        } }
    |   cbody vdecl { {
            fields = $2 :: $1.fields;
            methods = $1.methods;
            constructors = $1.constructors
        } }
    |   cbody fdecl { {
            fields = $1.fields;
            methods = $2 :: $1.methods;
            constructors = $1.constructors
        } }
    |   cbody constructor { {
            fields = $1.fields;
            methods = $1.methods;
            constructors = $2 :: $1.constructors
        } }

constructor:
        CONSTRUCTOR LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE { {
            typ = Constructortyp;
            fname = "constructor";
            formals = $3;
            locals = List.rev $6;
            body = List.rev $7;
        } }

fdecl:
    typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE { {
        typ = $1;
        fname = $2;
        formals = $4;
        locals = List.rev $7;
        body = List.rev $8
    } }

formals_opt:
    /* nothing */ { [] }
  | formal_list   { List.rev $1 }

formal_list:
    typ ID                   { [($1,$2)] }
```

```
  | formal_list COMMA typ ID { ($3,$4) :: $1 }

obj:
    CLASS ID { Object($2) }

primitive:
    INT { Int }
  | BOOL { Bool }
  | VOID { Void }
  | STRING { String }
  | FLOAT { Float }

arraytype:
    primitive LBRACKET RBRACKET { Arraytype($1) }

typ:
    primitive { $1 }
  | obj { $1 }
  | arraytype { $1 }

vdecl_list:
    /* nothing */    { [] }
  | vdecl_list vdecl { $2 :: $1 }

vdecl:
   typ ID SEMI { ($1, $2) }

stmt_list:
    /* nothing */  { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
    expr SEMI { Expr $1 }
  | RETURN SEMI { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }

expr_opt:
    /* nothing */ { Noexpr }
  | expr          { $1 }

expr:
    LITERAL          { Literal($1) }
  | STRING_LITERAL   { StringLit($1) }
  | FLOAT_LITERAL    { FloatLit($1) }
  | TRUE             { BoolLit(true) }
  | FALSE            { BoolLit(false) }
  | ID               { Id($1) }
```

```
  | LPAREN typ RPAREN expr { Cast( $2,$4) }
  | expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | expr AND    expr { Binop($1, And,   $3) }
  | expr OR     expr { Binop($1, Or,    $3) }
  | expr DOT    expr { ObjAccess($1, $3) }
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr         { Unop(Not, $2) }
  | expr ASSIGN expr   { Assign($1, $3) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN { $2 }
  | NEW ID LPAREN actuals_opt RPAREN { ObjCreate($2,$4) }
  | NEW primitive LBRACKET LITERAL RBRACKET { ArrayCreate($2, $4) }
  | expr LBRACKET LITERAL RBRACKET { ArrayAccess($1, $3) }
  | DELETE expr      { Delete($2) }

actuals_opt:
    /* nothing */ { [] }
  | actuals_list  { List.rev $1 }

actuals_list:
    expr                    { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }
```

Sast.ml (Wei, Wang and Yuan)

```
open Ast

type s_expr =
    S_Literal of int
  | S_BoolLit of bool
  | S_StringLit of string
  | S_FloatLit of float
  | S_Id of string * typ
  | S_Binop of s_expr * op * s_expr * typ
  | S_Unop of uop * s_expr * typ
  | S_Assign of s_expr * s_expr * typ
  | S_Call of string * s_expr list * typ
  | S_Noexpr
  | S_Null
  | S_Cast of typ * s_expr * typ
  | S_ObjCreate of string * s_expr list * typ
```

```
  | S_ObjAccess of s_expr * s_expr * typ
  | S_ArrayCreate of typ * int
  | S_ArrayAccess of s_expr * int * typ
  | S_Delete of s_expr

type s_stmt =
    S_Block of s_stmt list
  | S_Expr of s_expr * typ
  | S_Return of s_expr * typ
  | S_If of s_expr * s_stmt * s_stmt
  | S_For of s_expr * s_expr * s_expr * s_stmt
  | S_While of s_expr * s_stmt

type s_func_decl = {
    s_typ : typ;
    s_fname : string;
    s_formals : bind list;
    s_locals : bind list;
    s_body : s_stmt list;
}

type s_class_decl = {
    s_cname: string;
    s_fields: bind list;
    s_constructors: s_func_decl list;
    s_methods: s_func_decl list;
}

type s_program = {
    global_vars: bind list;
    functions: s_func_decl list;
    classes: s_class_decl list;
    main: s_func_decl;
    builtins: s_func_decl list;
}

(* Pretty-printing functions *)
let rec string_of_ast_typ = function
    Int -> "int"
    | Bool -> "bool"
    | Void -> "void"
    | Null -> "null"
    | String -> "string"
    | Object(name) -> name
    | Arraytype(t) -> "array of " ^ string_of_ast_typ t
    | Constructortyp -> "constructor"
    | Any -> "any"

let rec string_of_expr_type = function
    S_Literal(_) -> "int"
  | S_BoolLit(_) -> "bool"
  | S_StringLit(_) -> "string"
```

```
  | S_Id(name,data_type) -> string_of_ast_typ data_type
  | S_Binop(lhr,op,rhs,data_type) ->  string_of_ast_typ data_type
  | S_Unop(op,exp,data_type) -> string_of_ast_typ data_type
  | S_Assign(lhs,rhs,data_type) -> string_of_ast_typ data_type
  | S_Call(fname,expr_list,data_type) -> string_of_ast_typ data_type
  | S_Noexpr -> "Noexpr"
  | S_Null -> "null"
  | S_ObjCreate(obj_name,expr_list,data_type) -> string_of_ast_typ data_type
  | S_ObjAccess(obj_name,field,data_type) -> string_of_ast_typ data_type
  | S_ArrayCreate(typ, len) -> "(array of type " ^ string_of_ast_typ typ ^ " with length " ^
string_of_int len ^ ")"
  | S_ArrayAccess(arrayName, index, data_type) ->  "yuanlaishitype..type: " ^
string_of_ast_typ data_type ^ " index: " ^ string_of_int index ^ ")"
  | S_Delete(expr) -> string_of_expr_type expr
  | S_Cast(oldt,expr,newt) -> string_of_ast_typ newt

let rec string_of_s_expr = function
    S_Literal(l) -> string_of_int l
  | S_BoolLit(true) -> "true"
  | S_BoolLit(false) -> "false"
  | S_StringLit(s) -> s
  | S_FloatLit(f) -> string_of_float f
  | S_Id(s,t) -> "$" ^ s ^ "[" ^ string_of_typ t ^ "]"
  | S_Binop(e1, o, e2, t) ->
      "{" ^ string_of_s_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_s_expr e2 ^ "}[" ^
string_of_typ t ^ "]"
  | S_Unop(o, e,t) -> string_of_uop o ^ string_of_s_expr e ^ "[" ^ string_of_typ t ^ "]"
  | S_Assign(e1,e2,t) -> string_of_s_expr e1 ^ " = " ^ string_of_s_expr e2 ^ "[" ^
string_of_typ t ^ "]"
  | S_Call(f, el,t) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_s_expr el) ^ ")" ^ "[" ^
string_of_typ t ^ "]"
  | S_Noexpr -> ""
  | S_Null -> "Null"
  | S_Cast(oldt,expr,newt) ->
     "{" ^ "(" ^ string_of_s_expr(expr) ^ ")" ^ "[" ^ string_of_typ(oldt) ^ "->" ^
string_of_typ(newt) ^ "]" ^ "}"
  | S_ObjAccess(s1,s2,t) -> "{Object Access: " ^ string_of_s_expr s1 ^ "." ^
string_of_s_expr s2 ^ " " ^ string_of_typ t ^ "}"
  | S_ObjCreate(s,sel,t) -> "{Object Create: new " ^ s ^ "(" ^ String.concat "," (List.map
string_of_s_expr sel) ^ ")[" ^ string_of_typ t ^ "]}"
  | S_ArrayCreate(typ, len) -> "(array of type " ^ string_of_typ typ ^ " with length " ^
string_of_int len ^ ")"
  | S_ArrayAccess(arrayName, index, t) ->
        (* match arrayExpr with S_Id(arr,t) -> "(array_name: " ^ arr ^ "type:" ^
string_of_typ t ^ " index: " ^ string_of_int index ^ ")" *)
        "(array_name: " ^ string_of_s_expr arrayName ^ " type: " ^ string_of_typ t ^ "
index: " ^ string_of_int index ^ ")"
  | S_Delete(se) -> "delete " ^ string_of_s_expr se

let rec string_of_s_stmt = function
    S_Block(s_stmts) ->
```

```
      "{\n" ^ String.concat "" (List.map string_of_s_stmt s_stmts) ^ "}\n"
  | S_Expr(s_expr,t) -> "{" ^ string_of_s_expr s_expr ^ "} [" ^ string_of_typ t ^ "]" ^
";\n" ;
  | S_Return(s_expr,t) -> "return " ^ string_of_s_expr s_expr ^ ";\n" ^ "[" ^ string_of_typ
t ^ "]";
  | S_If(e, s, S_Block([])) -> "if (" ^ string_of_s_expr e ^ ")\n" ^ string_of_s_stmt s
  | S_If(e, s1, s2) ->  "if (" ^ string_of_s_expr e ^ ")\n" ^
      string_of_s_stmt s1 ^ "else\n" ^ string_of_s_stmt s2
  | S_For(e1, e2, e3, s) ->
      "for (" ^ string_of_s_expr e1  ^ " ; " ^ string_of_s_expr e2 ^ " ; " ^
      string_of_s_expr e3  ^ ") " ^ string_of_s_stmt s
  | S_While(e, s) -> "while (" ^ string_of_s_expr e ^ ") " ^ string_of_s_stmt s

let string_of_s_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

let string_of_s_formal (t, id) = string_of_typ t ^ " " ^ id

let string_of_s_fdecl fdecl =
  string_of_typ fdecl.s_typ ^ " " ^
  fdecl.s_fname ^ "(" ^ String.concat ", " (List.map string_of_s_formal fdecl.s_formals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.s_locals)^
  String.concat "" (List.map string_of_s_stmt fdecl.s_body) ^
  "\n}\n"

let string_of_s_cdecl cdecl =
    "class" ^ " " ^ cdecl.s_cname ^ " { \n" ^
    String.concat "" (List.map string_of_s_vdecl cdecl.s_fields) ^
    String.concat "" (List.map string_of_s_fdecl cdecl.s_constructors) ^
    String.concat "" (List.map string_of_s_fdecl cdecl.s_methods) ^
    "}\n"

let string_of_s_program (vars, funcs, classes, main, builtins) =
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_s_fdecl funcs) ^ "\n" ^
  String.concat "\n" (List.map string_of_s_cdecl classes) ^ "\n" ^
  String.concat "\n" (List.map string_of_s_fdecl builtins) ^ "\n" ^
  (string_of_s_fdecl main) ^ "\n"
```

Analyzer.ml (**Wei, Yuan** and Wang)

```
open Ast
open Sast



module StringMap = Map.Make(String)

let log_to_file str=
  let oc = open_out_gen [Open_creat; Open_text; Open_append] 0o640 "b.txt" in
```

```
  output_string oc str; close_out oc;;


type class_property_map = {
       field_map : Ast.typ StringMap.t;
       method_map : Ast.func_decl StringMap.t;
}

type env = {
       env_global_map : Ast.typ StringMap.t;
       env_local_map : Ast.typ StringMap.t;
       env_param_map : Ast.typ StringMap.t;
       env_function_map : Ast.func_decl StringMap.t;
       env_class_map : class_property_map StringMap.t;
       env_return_type : Ast.typ;
}

let log_to_file str=
  let oc = open_out_gen [Open_creat; Open_text; Open_append] 0o640 "b.txt" in
  output_string oc str; close_out oc;;

let define_builtin_functions =
       [
          {typ = Void; fname = "cast"; formals = [(String, "x")];
       locals = []; body = []};
       {typ = Void; fname = "malloc"; formals = [(String, "x")];
       locals = []; body = []};
       {typ = Void; fname = "sizeof"; formals = [(String, "x")];
       locals = []; body = []};
              {typ = Void; fname = "print"; formals = [(Any, "x")];
       locals = []; body = [] };
          {typ = String; fname = "toString"; formals = [(Any,"x")];
          locals = []; body = [] };

       ]

let build_maps(functions,builtins,globals) =
(*TODO: check duplicated function declarations here*)
       let function_map = List.fold_left (fun m fd ->
              if (StringMap.mem fd.fname m) then
                     raise (Failure("duplicated function declaration " ^ fd.fname))
              else
                     StringMap.add fd.fname fd m)
                        StringMap.empty (functions @ builtins)  in
       let global_map = List.fold_left (fun m (t,n) ->
       if (StringMap.mem n m) then
                     raise (Failure("duplicated global declaration " ^ n))
       else if (t = Void) then
                     raise(Failure("global " ^ n ^ " shouldn't be void type"))
              else
                     StringMap.add n t m)
                        StringMap.empty (globals)  in
```

```
                                        (function_map,global_map)

let build_class_field_map(fields,globals) =
       List.fold_left (fun m (t,n) ->
       if (StringMap.mem n m) then
                     raise (Failure("duplicated declaration " ^ n ^ ",field " ^
string_of_typ(t) ^ " " ^ n ^ " already exists"))
       else if (t = Void) then
                     raise(Failure(n ^ " shouldn't be void type "))
              else
                     StringMap.add n t m)
                        StringMap.empty (fields)

let build_class_method_map(cname,methods,functions) =
       let find_constructor = (fun f -> match f.typ with Constructortyp -> true  | _ ->
false) in
       let get_constructor  =
              let constructors = (List.find_all find_constructor methods) in
              let count = List.length constructors in
              if List.length constructors < 1 then
                     raise (Failure("Missing constructor for class " ^ cname))
              else List.hd constructors
       in
       ignore(get_constructor);
       List.fold_left (fun m fdecl ->
              if (StringMap.mem fdecl.fname m) then
                     raise (Failure("duplicated method declaration " ^ fdecl.fname))
              else
                     StringMap.add fdecl.fname fdecl m)
                        StringMap.empty (methods)

let build_class_map(globals,functions,classes) =
       let class_map = List.fold_left (fun m cdecl ->
       if (StringMap.mem cdecl.cname m) then
              raise (Failure("duplicated class declaration" ^ cdecl.cname))
       else
              StringMap.add cdecl.cname
              {
                     field_map = build_class_field_map(cdecl.cbody.fields,globals);
                     method_map =
build_class_method_map(cdecl.cname,cdecl.cbody.constructors@cdecl.cbody.methods,functions);
              }
              m)
              StringMap.empty (classes) in
                                  class_map

(* locals could overshadow params, params could overshadow globals *)
let get_id_type(env,s) =
       try StringMap.find s env.env_local_map
       with | Not_found ->
              try StringMap.find s env.env_param_map
              with |Not_found ->
```

```
                    try StringMap.find s env.env_global_map
               with |Not_found -> raise (Failure("undefined_id " ^ s))

let get_function_decl(env,fname) =
       try StringMap.find fname env.env_function_map
       with | Not_found -> raise (Failure("UndefinedFunction " ^ fname))

let get_sexpr_type(sexpr)= match (sexpr) with
    S_Literal(_) -> Int
  | S_BoolLit(_) -> Bool
  | S_StringLit(_) -> String
  | S_FloatLit(_) -> Float
  | S_Id(_,t) -> t
  | S_Binop(_,_,_,t) -> t
  | S_Unop(_,_,t) -> t
  | S_Assign(_,_,t) -> t
  | S_Call(_,_,t) -> t
  | S_Noexpr -> Void
  | S_Null -> Null
  | S_Cast(_,_,t) -> t
  | S_ObjCreate(_,_,t) -> t
  | S_ObjAccess(_,_,t) -> t
  | S_ArrayCreate(t,_) -> Arraytype(t)
  | S_ArrayAccess(_,_,t) -> t
  | S_Delete(_) -> Void

let report_duplicate exceptf list =
let rec helper = function
n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
       | _ :: t -> helper t
       | [] -> ()
in helper (List.sort compare list)

let get_variable_list lst =
       List.fold_left (fun lst (_,n) -> n::lst)
                      [] lst

let get_function_list lst =
       List.fold_left (fun lst fdecl -> fdecl.s_fname::lst)
                      [] lst

let rec exprl_to_sexprl (env,el) =
  let env_ref = ref(env) in
  let rec helper = function
         head::tail ->
              let a_head, env = generate_sexpr(!env_ref,head) in
              env_ref := env;
              a_head::(helper tail)
       | [] -> []
  in (helper el), !env_ref
```

```
and check_call_type (env,fname,exprs) =
	let actuals,_ = exprl_to_sexprl(env,exprs) in
	let fd = get_function_decl(env,fname) in
	let match_params = fun (ft,_) actual ->
		if (get_sexpr_type(actual) = ft || ft = Any ) then
			()
		else raise(Failure ("illegal actual argument found " ^
string_of_typ(get_sexpr_type(actual)) ^
			" expected " ^ string_of_typ ft ^ " in " ^ string_of_s_expr actual))
		in
	let _ = if List.length(exprs) != List.length(fd.formals) then
	raise (Failure ("expecting " ^ string_of_int
		(List.length fd.formals) ^ " argument(s) in " ^
string_of_expr(Call(fname,exprs))))
		else
			List.iter2 match_params fd.formals actuals
		in
			S_Call(fname,actuals,fd.typ)

and check_assign (env,e1,e2) =
	let sexpr1,_ = generate_sexpr(env,e1) in
	let sexpr2,_ = generate_sexpr(env,e2) in
	let type1 = get_sexpr_type (sexpr1) in
	let type2 = get_sexpr_type (sexpr2) in
		if (type1 = type2) then
			S_Assign(sexpr1,sexpr2,type1)
		else if (type1 = Float && type2 = Int) then
			S_Assign(sexpr1,S_Cast(Int,sexpr2,Float) ,type1)
		else if (type1 = Arraytype(Int) && type2 = Int) then
			S_Assign(sexpr1,sexpr2,Int)
		else if (type1 = Arraytype(Bool) && type2 = Bool) then
			S_Assign(sexpr1,sexpr2,Bool)
		else if (type1 = Arraytype(Float) && type2 = Float) then
			S_Assign(sexpr1,sexpr2,Float)
		else if (type1 = Arraytype(String) && type2 = String) then
			S_Assign(sexpr1,sexpr2,String)
		else
			raise(Failure ("illegal assignment " ^ string_of_typ type1 ^
							" = " ^ string_of_typ type2 ^ " in
" ^
							string_of_s_expr sexpr1 ^ " = " ^
string_of_s_expr sexpr2))

and check_unop (env,op,e) =
	let sexpr,_ = generate_sexpr(env,e) in
	let type1 = get_sexpr_type(sexpr) in
	match(op) with
	Neg -> if (type1 = Int || type1 = Float) then S_Unop(op,sexpr,type1)
		else raise (Failure ("illegal unary operator " ^ string_of_uop op ^
				string_of_typ(type1) ^ " in " ^ string_of_expr(Unop(op,e)) ))
		| Not -> if (type1 = Bool) then S_Unop(op,sexpr,type1)
		else raise (Failure ("illegal unary operator " ^ string_of_uop op ^
```

```
                              string_of_typ(type1) ^ " in " ^ string_of_expr(Unop(op,e)) ))
and check_binop (env,e1,op,e2) =
       let origin_sexpr1,_ = generate_sexpr(env,e1) in
       let origin_sexpr2,_ = generate_sexpr(env,e2) in
       let origin_type1 = get_sexpr_type (origin_sexpr1) in
       let origin_type2 = get_sexpr_type (origin_sexpr2) in
       let (sexpr1,type1) =
              if (origin_type1 = Int && origin_type2 = Float) then
                     (S_Cast(Int,origin_sexpr1,Float),Float)
              else
                     (origin_sexpr1,origin_type1) in
       let (sexpr2,type2) =
              if (origin_type1 = Float && origin_type2 = Int) then
                     (S_Cast(Int,origin_sexpr2,Float),Float)
              else
                     (origin_sexpr2,origin_type2) in
       match(op) with
       Add  when ((type1 = Int && type2 = Int) || (type1 = String && type2 = String) ||
(type1 = Float && type2 = Float))
         -> S_Binop(sexpr1,op,sexpr2,type1)
              | Sub | Mult | Div when ((type1 = Int && type2 = Int) || (type1 = Float &&
type2 = Float))
              -> S_Binop(sexpr1,op,sexpr2,type1)
           | Equal when (type1 = type2 && type1 = String) ->
              S_Binop(S_Literal(0),Equal,S_Call("strcmp",[sexpr1;sexpr2;],Int),Bool)
              | Neq when (type1 = type2 && type1 = String) ->

       S_Unop(Not,S_Binop(S_Literal(0) ,Equal,S_Call("strcmp",[sexpr1;sexpr2;],Int),Bool),Bo
ol)
              | Equal | Neq when (type1 = type2) ->
                      S_Binop(sexpr1,op,sexpr2,Bool)
           | Less | Leq | Greater | Geq when ((type1 = Int && type2 = Int) || (type1 = Float
&& type2 = Float) )
                     -> S_Binop(sexpr1,op,sexpr2,Bool)
           | And | Or when (type1 = Bool && type2 = Bool) -> S_Binop(sexpr1,op,sexpr2,Bool)
              | _ -> raise (Failure ("illegal binary operator " ^
                            string_of_typ type1 ^ " " ^ string_of_op op ^ " " ^
                            string_of_typ type2 ^ " in " ^
string_of_expr(Binop(e1,op,e2))))
and check_cast (env,t,e) =
       let sexpr,_ = generate_sexpr(env,e) in
       let oldt = get_sexpr_type(sexpr) in
              match (oldt,t) with
               (Int,Int) | (Float,Float) | (Bool,Bool) | (String,String)
               | (Float,Int) | (Int,Float) | (String,Int) | (String,Float)
               | (Float,String) | (Int,String) | (Bool,String) ->
              S_Cast(oldt,sexpr,t)
              | _ ->
                     raise(Failure("Covert from " ^ string_of_typ(oldt) ^ " to " ^
string_of_typ(t) ^ "is not permitted" ) )
and check_objaccess(env,e1,e2) =
       ignore(log_to_file("Checking Obj access\n\n"));
```

```
	let sexpr1,_ = generate_sexpr(env,e1) in
	let objtyp = get_sexpr_type sexpr1 in
	match(objtyp) with
			Object(cname) ->
check_objaccess_class_property(env,cname,sexpr1,e1,e2)
		|	_ -> raise (Failure ("illegal accessing:" ^ string_of_expr e1 ^ " is
not an object"))
and check_objaccess_class_property(env,cname,sexpr1,e1,e2) =
	match(e2) with
			Id s -> check_objaccess_class_field(env,cname,sexpr1,s)
		|   Call(s, exprs) ->
check_objaccess_class_method(env,cname,sexpr1,e1,e2,s,exprs)
		|	ObjAccess(e3,e4) -> let e1' = ObjAccess(e1,e3) in
check_objaccess(env,e1',e4)
		|	_ -> raise (Failure ("illegal accessing:" ^ string_of_expr e2 ^ " is
not a class property"))
and check_objaccess_class_field(env,cname,sexpr1,s) =
	let property_map = try StringMap.find cname env.env_class_map with | Not_found ->
raise (Failure ("UndefinedClass" ^ cname)) in
	let e2_typ = try StringMap.find s property_map.field_map with | Not_found -> raise
(Failure("illegal accessing: class " ^ cname ^ " has no field " ^ s)) in
	let sexpr2 = S_Id(s,e2_typ) in S_ObjAccess(sexpr1, sexpr2, e2_typ)
and check_objaccess_class_method(env,cname,sexpr1,e1,e2,s,exprs) =
	if cname = "constructor" then raise(Failure("illegal accessing: constructors cannot
be accessed in this way")) else
	let property_map = try StringMap.find cname env.env_class_map with | Not_found ->
raise (Failure ("UndefinedClass" ^ cname)) in
	let e2_fdecl = try StringMap.find s property_map.method_map with | Not_found -> raise
(Failure("illegal accessing: class " ^ cname ^ " has no method " ^ s)) in
	let actuals,_ = exprl_to_sexprl(env,exprs) in
	let match_params = fun (ft,_) actual ->
		if (get_sexpr_type(actual) = ft) then
			()
		else raise(Failure ("illegal actual argument found " ^
string_of_typ(get_sexpr_type(actual)) ^
			" expected " ^ string_of_typ ft ^ " in " ^
string_of_expr(Call(s,exprs))))
		in
	let _ = if List.length(exprs) != List.length(e2_fdecl.formals) then
		raise (Failure ("expecting " ^ string_of_int
		(List.length e2_fdecl.formals) ^ " argument(s) in " ^ string_of_expr e1 ^ "."
^ string_of_expr(Call(s,exprs))))
		else
			List.iter2 match_params e2_fdecl.formals actuals
		in
			S_ObjAccess(sexpr1,S_Call(s,actuals,e2_fdecl.typ),e2_fdecl.typ)

and is_typ_match(actuals,formals) =
	(match actuals with [] -> Bool | h::l -> get_sexpr_type(h)) = (match formals with []
-> Bool | (ft,_)::l -> ft) &&
	is_typ_match((match actuals with [] -> [] | h::l -> l),(match formals with [] -> [] |
h::l -> l))
```

```
and is_fdecl_match(env,fdecl,exprs) =
       if List.length(exprs) != List.length(fdecl.formals) then false else
       let actuals,_ = exprl_to_sexprl(env,exprs) in
       is_typ_match(actuals,fdecl.formals)

and find_and_test_constructor(fname,cname,sel) =
       let appended_constructor_name = cname ^ ".constructor" ^
              if List.length sel > 0 then
                     "." ^ String.concat "." (List.map (fun sexpr -> string_of_typ
(get_sexpr_type sexpr)) sel)
              else
                     ""
       in
       fname = appended_constructor_name

and check_objcreate(env,cname,el) =
       let property_map = try StringMap.find cname env.env_class_map with | Not_found ->
raise (Failure ("UndefinedClass " ^ cname)) in

       let sel = List.rev(List.fold_left (fun sl e -> let sexpr, _ = generate_sexpr(env,e)
in sexpr :: sl) [] el) in

       if StringMap.exists (fun k v -> find_and_test_constructor(k,cname,sel))
property_map.method_map then
              S_ObjCreate(cname,sel,Object(cname))
       else
              raise (Failure("No matching constructor for " ^ cname ^ "(" ^ String.concat
"," (List.map string_of_expr el) ^ ")"))

and check_array_init(datatype,size) =
       S_ArrayCreate(datatype, size)

and check_array_access(env,e,index) =
       let se,_ = generate_sexpr(env, e) in
       let typ = match e with
         Id s -> get_id_type(env,s)
       in
       S_ArrayAccess(se, index, typ)

and check_delete(env,e) =
       let se,_ = generate_sexpr(env,e) in
       let t = get_sexpr_type(se) in
       match t with
              Object(_) | Arraytype(_) -> S_Delete(se)
       |      _ -> raise(Failure("illegal delete: " ^ string_of_expr e ^ " is not an
object"))
and generate_sexpr (env,expr) =
       match (expr) with
              Literal i -> S_Literal(i), env
       |      BoolLit b -> S_BoolLit(b), env
       |      StringLit f -> S_StringLit(f), env
```

```
        |       FloatLit f -> S_FloatLit(f), env
        |       Id s -> S_Id(s, get_id_type(env,s)), env
        |       Call(s, exprs) -> check_call_type(env,s,exprs), env
        |       Assign(e1, e2) -> check_assign(env,e1,e2), env
        |       Unop(op, e)-> check_unop(env,op,e), env
        |       Binop(e1, op, e2)-> check_binop(env,e1,op,e2),env
        |       Noexpr -> S_Noexpr,env
        |       Nullexpr -> S_Null,env
        |       Cast(t,e) -> check_cast(env,t,e),env
    |   ObjAccess(e1, e2) -> check_objaccess(env,e1,e2),env
        |       ObjCreate(cname, el) -> check_objcreate(env,cname,el), env
        |    ArrayCreate(t, n) -> check_array_init(t,n),env
        |    ArrayAccess(e, index) -> check_array_access(env,e,index),env
        |    Delete(e) -> check_delete(env,e), env


let rec stmtl_to_sstmtl (env,stmt_list) =
        let env_ref = ref(env) in
        let rec iter = function
                head::tail ->
                        let a_head, env = generate_sstmt(!env_ref,head) in
                        env_ref := env;
                        a_head::(iter tail)
        |       [] -> []
        in
        let sstmt_list = (iter stmt_list), !env_ref in
        sstmt_list

and check_block (env,sl) =  match sl with
                [] -> S_Block([S_Expr(S_Noexpr,Void)]), env
        |       _  ->
                let s_sl, _ = stmtl_to_sstmtl(env,sl) in
                (S_Block(s_sl), env)

and check_expr_stmt (env,e) =
        let se, env = generate_sexpr(env,e) in
        let t = get_sexpr_type(se) in
        S_Expr(se, t), env

and check_return (env,e) =
        let se, _ = generate_sexpr (env,e) in
        let t = get_sexpr_type(se) in
        if (t = env.env_return_type) then
                S_Return(se,t), env
        else
                raise (Failure ("return gives " ^ string_of_typ t ^ " expected " ^
                        string_of_typ env.env_return_type ^ " in " ^ string_of_s_expr se))

and check_if (env,p,b1,b2) =
        let se, _ = generate_sexpr(env,p) in
        let t = get_sexpr_type(se) in
        let ifbody, _ = generate_sstmt (env,b1) in
```

```
        let elsebody, _ = generate_sstmt (env,b2) in
        if t = Bool
                then S_If(se, ifbody, elsebody), env
    else raise (Failure ("expected Boolean expression in " ^ string_of_s_expr se))
and check_for (env,e1,e2,e3,s) =
        let se1, _ = generate_sexpr(env,e1) in
        let se2, _ = generate_sexpr(env,e2) in
        let se3, _ = generate_sexpr(env,e3) in
        let forbody, _ = generate_sstmt(env,s) in
        let conditional = get_sexpr_type(se2) in
        let sfor =
                if (conditional = Bool) then (* Could be void too *)
                        S_For(se1, se2, se3, forbody)
                else raise (Failure ("invaild statement type in For"))
        in
        (sfor, env)
and check_while (env,p,s) =
        let se, _ = generate_sexpr(env,p) in
        let t = get_sexpr_type(se) in
        let sstmt, _ = generate_sstmt(env,s) in
        let swhile =
                if (t = Bool ) then (* Could be void too *)
                        S_While(se, sstmt)
                else raise (Failure ("invaild statement type in While"))
        in
                (swhile, env)
and  generate_sstmt (env,stmt) =
 match (stmt) with
                Block sl ->check_block(env,sl)
        |       Expr e -> check_expr_stmt(env,e)
    |   Return e -> check_return(env,e)
    |   If(p, b1, b2) -> check_if(env,p,b1,b2)
    |   For(e1, e2, e3, st) -> check_for(env,e1,e2,e3,st)
    |    While(p, s) -> check_while(env,p,s)

let check_fbody(fname,fbody,return_type) =
        let len = List.length fbody in
        if len = 0 then () else
        let final_stmt = List.hd (List.rev fbody) in
        match return_type, final_stmt with
                Void, _ -> ()
        |       Constructortyp, _ -> ()
        |       _, S_Return(_, _) -> ()
        |       _ -> raise(Failure ("Missing return statement: " ^ fname))

(*
let check_foramls (fname, formals) =
        if (List.length(List.sort_uniq(formals)) <> List.length(formals)) then
                raise(Failure ("duplicate formal in ")^fname)
        else ()
*)
```

```
let generate_sfdecl (fdecl,function_map,global_map,class_map) =
	let local_map = List.fold_left (fun m (t,n) ->
		if (t = Void) then
			raise(Failure("local " ^ n ^ " shouldn't be type void"))
		else
			StringMap.add n t m)
		StringMap.empty fdecl.locals in
	let param_map = List.fold_left (fun m (t,n) ->
		if (t = Void) then
			raise(Failure("formal " ^ n ^ " shouldn't be type void"))
		else
			StringMap.add n t m)
	 StringMap.empty fdecl.formals in
	let env =
	{
		env_class_map = class_map;
		env_global_map = global_map;
		env_local_map = local_map;
		env_param_map = param_map;
		env_function_map = function_map;
		env_return_type = fdecl.typ;
	} in
	let sfbody = List.fold_left (fun lst stmt ->
		let sstmt, _ = generate_sstmt(env,stmt) in
		lst@[sstmt] ) [] fdecl.body in
	ignore(check_fbody(fdecl.fname,sfbody,fdecl.typ));
	ignore(report_duplicate (fun n -> "duplicate formals " ^ n) (List.map snd
fdecl.formals));
	ignore(report_duplicate (fun n -> "duplicate locals " ^ n) (List.map snd
fdecl.locals));

	(*ignore(check_foramls(fdecl.fname,fdecl.formals));*)
	{
		s_typ = fdecl.typ;
		s_fname = fdecl.fname;
		s_formals = fdecl.formals;
		s_locals = fdecl.locals;
		s_body = sfbody;
	}

let get_main(functions) =
	let find_main = (fun f -> match f.fname with "main" -> true  | _ -> false) in
	let mains = (List.find_all find_main functions) in
		if List.length mains < 1 then
			raise (Failure("Main didn't defined"))
		else if List.length mains > 1 then
			raise (Failure("MultipleMainsDefined"))
		else List.hd mains

let append_to_name(cname,formals,name) =
	cname ^ "." ^ name ^
	if List.length formals > 0 then
```

```
               "." ^ String.concat "." (List.map (fun (typ,_) -> string_of_typ typ) formals)
       else

               ""

let append_to_callname(formals,name) =
       name ^
       if List.length formals > 0 then
               "." ^ String.concat "." (List.map (fun (typ,_) -> string_of_typ typ) formals)
       else

               ""

(* type s_expr =
    S_Literal of int
  | S_BoolLit of bool
  | S_StringLit of string
  | S_FloatLit of float
  | S_Id of string * typ
  | S_Binop of s_expr * op * s_expr * typ
  | S_Unop of uop * s_expr * typ
  | S_Assign of s_expr * s_expr * typ
  | S_Call of string * s_expr list * typ
  | S_Noexpr
  | S_Null
  | S_ObjCreate of string * s_expr list * typ
  | S_ObjAccess of s_expr * s_expr * typ
  | S_Delete of s_expr *)

let rec implicit_this_in_constructor_sexpr(c_typ,s_expr,property_map) =
       match (s_expr) with
               S_Id(s,typ) -> if StringMap.mem s property_map.field_map then
S_ObjAccess(S_Id("this",c_typ),S_Id(s,typ),typ) else S_Id(s,typ)
       |   S_Binop(se1,op,se2,typ) ->
S_Binop(implicit_this_in_constructor_sexpr(c_typ,se1,property_map),op,implicit_this_in_const
ructor_sexpr(c_typ,se2,property_map),typ)
       |   S_Unop(uop,se,typ) ->
S_Unop(uop,implicit_this_in_constructor_sexpr(c_typ,se,property_map),typ)
       |       S_Assign(se1,se2,typ) ->
S_Assign(implicit_this_in_constructor_sexpr(c_typ,se1,property_map),implicit_this_in_constru
ctor_sexpr(c_typ,se2,property_map),typ)
       |   S_Call(s,sel,typ) -> if StringMap.mem s property_map.method_map then
                                                       S_Call(s,List.rev (List.fold_left
(fun lst se -> implicit_this_in_constructor_sexpr(c_typ,se,property_map) :: lst) []
sel),typ)
                                                  else S_Call(s,sel,typ)
       |   S_ObjCreate(s,sel,typ) -> S_ObjCreate(s,List.rev (List.fold_left (fun lst se ->
implicit_this_in_constructor_sexpr(c_typ,se,property_map) :: lst) [] sel),typ)
       |   S_ObjAccess(se1,se2,typ) -> (match se1 with
                                                               S_Id("this",_) ->
S_ObjAccess(se1,se2,typ)
                                                       |       _ ->
```

```
S_ObjAccess(implicit_this_in_constructor_sexpr(c_typ,se1,property_map),implicit_this_in_cons
tructor_sexpr(c_typ,se2,property_map),typ))
	|	S_Delete(se) ->
S_Delete(implicit_this_in_constructor_sexpr(c_typ,se,property_map))
	|	_ -> s_expr


let rec implicit_this_in_constructor_stmt(c_typ,s_stmt,property_map) =
	match (s_stmt) with
		S_Block(sl) ->
S_Block(implicit_this_in_constructor_body(c_typ,sl,property_map))
	|	S_Expr(s,t) ->
S_Expr(implicit_this_in_constructor_sexpr(c_typ,s,property_map),t)
	|	S_Return(_,_) -> raise (Failure("Constructors cannot return a value"))
	|	S_If(s,st1,st2) ->
S_If(implicit_this_in_constructor_sexpr(c_typ,s,property_map),implicit_this_in_constructor_s
tmt(c_typ,st1,property_map),

	implicit_this_in_constructor_stmt(c_typ,st2,property_map))
	|	S_For(s1,s2,s3,st) ->
S_For(implicit_this_in_constructor_sexpr(c_typ,s1,property_map),implicit_this_in_constructor
_sexpr(c_typ,s2,property_map),

	implicit_this_in_constructor_sexpr(c_typ,s3,property_map),implicit_this_in_constructo
r_stmt(c_typ,st,property_map))
	|	S_While(s,st) ->
S_While(implicit_this_in_constructor_sexpr(c_typ,s,property_map),implicit_this_in_constructo
r_stmt(c_typ,st,property_map))

and implicit_this_in_constructor_body (c_typ,s_body,property_map) =
	let modified_s_body =
		List.fold_left (fun lst s_stmt ->
implicit_this_in_constructor_stmt(c_typ,s_stmt,property_map)::lst) [] s_body
	in
	List.rev modified_s_body

let implicit_constructor_body (c_typ,s_body,property_map) =
	[S_Expr(S_Assign(
		S_Id("this",c_typ),
		S_Call("cast",
				[
					S_Call("malloc",
							[
								S_Call("sizeof",
										[

	S_Id("this",c_typ)
										],Int)
							],Any)
				],c_typ),
		c_typ
	),c_typ)]
```

```
	@ implicit_this_in_constructor_body(c_typ,s_body,property_map) @
	[S_Return(S_Id("this",c_typ),c_typ)]


let update_sconstrdecl(sfdecl,cname,property_map) =
	{
		s_typ = Object(cname);
		s_fname = sfdecl.s_fname;
		s_formals = sfdecl.s_formals;
		s_locals =  sfdecl.s_locals;
		s_body = implicit_constructor_body(Object(cname),sfdecl.s_body,property_map);
	}

let update_sfdecl(sfdecl,cname) =
	{
		s_typ = sfdecl.s_typ;
		s_fname = cname ^ "." ^ sfdecl.s_fname;
		s_formals = sfdecl.s_formals;
		s_locals = sfdecl.s_locals;
		s_body = sfdecl.s_body;
	}

let generate_scdecl (cdecl,class_map) =
	let property_map = try StringMap.find cdecl.cname class_map with | Not_found -> raise
(Failure ("UndefinedClass " ^ cdecl.cname)) in
	{
		s_cname = cdecl.cname;
		s_fields = cdecl.cbody.fields;
		s_constructors = (let raw_s_constructors = List.fold_left (fun lst fdecl ->

	generate_sfdecl(fdecl,property_map.method_map,property_map.field_map,class_map)::lst)
							[] cdecl.cbody.constructors in
							List.fold_left (fun lst sfdecl ->

	update_sconstrdecl(sfdecl,cdecl.cname,property_map)::lst)
							[] raw_s_constructors);
		s_methods = (let raw_s_methods = List.fold_left (fun lst fdecl ->

	generate_sfdecl(fdecl,property_map.method_map,property_map.field_map,class_map)::lst)
							[] cdecl.cbody.methods in
							List.fold_left (fun lst sfdecl ->
							update_sfdecl(sfdecl,cdecl.cname)::lst)
							[] raw_s_methods);
	}

let generate_sast (globals,functions,builtins,classes,function_map,global_map,class_map) =
	let smain = generate_sfdecl(get_main(functions),function_map,global_map,class_map) in
	let sfdecls = List.fold_left (fun lst fdecl ->
		if (fdecl.fname = "main") then lst else
		generate_sfdecl(fdecl,function_map,global_map,class_map)::lst) [] functions in
	let sbuiltins = List.fold_left (fun lst fdecl ->
generate_sfdecl(fdecl,function_map,global_map,class_map)::lst) [] builtins in
```

```
	let scdecls = List.fold_left (fun lst cdecl -> generate_scdecl(cdecl,class_map)::lst)
[] classes in
	(globals,sfdecls,scdecls,smain,sbuiltins)

let overload_class(classes) =
	List.fold_left (fun lst cdecl ->
	{
		cname = cdecl.cname;
		cbody =
		{
			fields = cdecl.cbody.fields;
			methods = List.fold_left (fun lst fdecl -> {
				typ = fdecl.typ;

				fname =
append_to_callname((Object(cdecl.cname),"this")::fdecl.formals,fdecl.fname);

				formals = (Object(cdecl.cname),"this")::fdecl.formals;
				locals = fdecl.locals;
				body = fdecl.body;
			}::lst) [] cdecl.cbody.methods;
			constructors = List.fold_left (fun lst condecl -> {
					typ = condecl.typ;
				fname =
append_to_name(cdecl.cname,condecl.formals,condecl.fname);
				formals = condecl.formals;
				locals = (Object(cdecl.cname),"this") :: condecl.locals;
					body = condecl.body;
			}::lst) [] cdecl.cbody.constructors;
		}
	}::lst)
	[] classes

let overload_callname(s,el,env) =
	let newS =
	s ^
	if List.length el > 0 then
		"." ^ String.concat "." (List.map (fun expr -> let sexpr,_ =
generate_sexpr(env,expr) in string_of_typ(get_sexpr_type(sexpr))) el)
	else
		""
	in
	ignore(log_to_file(newS^"\n\n"));
	newS

let rec overload_expr(expr,env) =
	ignore(log_to_file("Overloading " ^ string_of_expr(expr) ^ "\n\n"));
	match (expr) with
		ObjAccess(e1,e2) -> (
			match e2 with
				Call(s,el) -> let modified_el = List.rev(List.fold_left(fun lst
expr -> overload_expr(expr,env)::lst) [] (e1::el)) in
```

```
ObjAccess(e1,Call(overload_callname(s,modified_el,env),modified_el))
                    |   ObjAccess(e3,e4) -> let e1' = ObjAccess(e1,e3) in
overload_expr(ObjAccess(e1',e4),env)
                    |      _ -> expr
              )
       |      Binop(e1,op,e2) -> Binop(overload_expr(e1,env),op,overload_expr(e2,env))
       |   Unop(uop,e) -> Unop(uop,overload_expr(e,env))
       |      Assign(e1,e2) -> Assign(overload_expr(e1,env),overload_expr(e2,env))
       |      Call(s,el) -> Call(s,List.rev (List.fold_left (fun lst e ->
overload_expr(e,env)::lst)[] el))
       |      ObjCreate(s,el) -> ObjCreate(s,List.rev (List.fold_left (fun lst e ->
overload_expr(e,env)::lst)[] el))
       |      _ -> expr

let rec overload_stmt(stmt,env) =
       match (stmt) with
              Block(sl) -> Block(overload_function_body(sl,env))
       |      Expr(e) -> Expr(overload_expr(e,env))
       |      Return(e) -> Return(overload_expr(e,env))
       |   If(e,st1,st2) ->
If(overload_expr(e,env),overload_stmt(st1,env),overload_stmt(st2,env))
       |   For(e1,e2,e3,st) ->
For(overload_expr(e1,env),overload_expr(e2,env),overload_expr(e3,env),overload_stmt(st,env))
       |      While(e,st) -> While(overload_expr(e,env),overload_stmt(st,env))

and overload_function_body(body,env) =
       let modified_body =
              List.fold_left (fun lst stmt -> overload_stmt(stmt,env)::lst) [] body
       in
       List.rev modified_body

let overload_function(functions,function_map,global_map,class_map) =
       List.fold_left (fun lst fdecl ->
       let local_map = List.fold_left (fun m (t,n) ->
              if (t = Void) then
                     raise(Failure("local " ^ n ^ " shouldn't be type void"))
              else
                     StringMap.add n t m)
              StringMap.empty fdecl.locals in
       let param_map = List.fold_left (fun m (t,n) ->
              if (t = Void) then
                     raise(Failure("formal " ^ n ^ " shouldn't be type void"))
              else
                     StringMap.add n t m)
        StringMap.empty fdecl.formals in
       let env =
       {
              env_class_map = class_map;
              env_global_map = global_map;
              env_local_map = local_map;
              env_param_map = param_map;
```

```
                env_function_map = function_map;
                env_return_type = fdecl.typ;
        } in
        {
                typ = fdecl.typ;
                fname = fdecl.fname;
                formals = fdecl.formals;
                locals = fdecl.locals;
                body = overload_function_body(fdecl.body,env);
        }::lst)
        [] functions


let overload_body_class(classes,class_map) =
        List.fold_left (fun lst cdecl ->
        let property_map = try StringMap.find cdecl.cname class_map with | Not_found -> raise
(Failure ("UndefinedClass " ^ cdecl.cname)) in
        {
                cname = cdecl.cname;
                cbody =
                {
                        fields = cdecl.cbody.fields;
                        methods =
overload_function(cdecl.cbody.methods,property_map.method_map,property_map.field_map,class_m
ap);
                        constructors =
overload_function(cdecl.cbody.constructors,property_map.method_map,property_map.field_map,cl
ass_map);
                }
        }::lst)
        [] classes

(* Main method for analyzer *)
let analyze program =
match program with
        (globals, functions, classes) ->
                let builtins = define_builtin_functions in
                let modified_classes = overload_class(classes) in
                let function_map, global_map = build_maps(functions,builtins,globals) and
class_map = build_class_map(globals,functions@builtins,modified_classes) in

                let modified_body_classes = overload_body_class(modified_classes,class_map) in
                let modified_functions =
overload_function(functions,function_map,global_map,class_map) in
                let modified_function_map, _ = build_maps(modified_functions,builtins,globals)
in

        generate_sast(globals,modified_functions,builtins,modified_body_classes,modified_func
tion_map,global_map,class_map)
```

Codegen.ml (**Liu**, Wei and Wang)

```
open Llvm

module L = Llvm
module A = Ast
module S = Sast
module ANA = Analyzer

module StringMap = Map.Make(String)

module H = Hashtbl

module P=Printf

(* global hashtables that store everything we need *)
let object_types:(string, L.lltype) H.t = H.create 10;;
let object_field_indices:(string, int) H.t = H.create 50;;
let globals_table:(string, L.llvalue) H.t = H.create 50;;
let locals_table:(string, L.llvalue) H.t = H.create 50;;
let params_table:(string, L.llvalue) H.t = H.create 50;;
let func_table:(string, L.llvalue * S.s_func_decl) H.t = H.create 50;;

let log_to_file str=
  let oc = open_out_gen [Open_creat; Open_text; Open_append] 0o640 "a.txt" in
  output_string oc str; close_out oc;;

let print_hashtable_int table =
  H.iter (fun k i -> log_to_file (k^":"^(string_of_int i)^"\n")) table;;

let print_hashtable table =
  H.iter (fun k v -> log_to_file (k^(L.string_of_llvalue v)^"\n")) table;;

let context = L.global_context ()
let the_module = L.create_module context "MicroC"
and i32_t  = L.i32_type  context (* integer *)
and i8_t   = L.i8_type   context (* printf_format_string *) (* char *)
and i1_t   = L.i1_type   context (* boolean *)
and i8_pt  = L.pointer_type (L.i8_type context) (* string *)
and void_t = L.void_type context  (* void *)
and f_t = L.double_type context;;


let rec find_object_type name =
  try Hashtbl.find object_types name
  with | Not_found -> raise (Failure("no class def of " ^ name))

(* convert sast type to underlying *)
and get_underlying_type_of_sexpr = function
    S.S_Literal(_) -> i32_t
  | S.S_BoolLit(_)  -> i1_t
  | S.S_FloatLit(_) -> f_t
```

```
  | S.S_StringLit(_) -> i8_pt
  | S.S_Id(_, data_type) -> ltype_of_typ data_type
  | S.S_Unop(_, _, data_type) -> ltype_of_typ data_type
  | S.S_Binop(_, _, _, data_type) -> ltype_of_typ data_type
  | S.S_Assign(_, _, data_type) -> ltype_of_typ data_type
  | S.S_Call(_, _, data_type) -> ltype_of_typ data_type
  | S.S_Noexpr -> void_t
  | S.S_Cast(_,_,data_type) ->  ltype_of_typ data_type
  | S.S_ObjAccess(_, _, data_type)          -> ltype_of_typ data_type
  | S.S_ObjCreate(_, _, data_type)   ->  ltype_of_typ data_type
  (*| S.SArrayPrimitive(_, d)-> d*)
  | S.S_Null -> i32_t
  |    d -> raise(Failure ("Met unknown type in get_underlying_type_of_sexpr\n") )

and get_ast_type_of_sexpr = function
    S.S_Literal(_) -> A.Int
  | S.S_BoolLit(_)  -> A.Bool
  | S.S_FloatLit(_) -> A.Float
  | S.S_StringLit(_) -> A.String
  | S.S_Id(_, data_type) -> data_type
  | S.S_Unop(_, _, data_type) -> data_type
  | S.S_Binop(_, _, _, data_type) -> data_type
  | S.S_Assign(_, _, data_type) -> data_type
  | S.S_Call(_, _, data_type) -> data_type
  | S.S_Noexpr -> A.Void
  | S.S_ObjAccess(_, _, data_type)          -> data_type
  | S.S_ObjCreate(_, _, data_type)   ->  data_type
  | S.S_Cast(_,_,data_type) ->  data_type
  (*| S.SArrayPrimitive(_, d)-> d*)
  | S.S_ArrayAccess(arrayName, index, t) -> t
  | S.S_ArrayCreate(typ, size) -> typ
  | S.S_Null -> A.Int
  |    d -> raise(Failure ("Met unknown type in get_ast_type_of_sexpr\n") )

and ltype_of_typ = function
    A.Int -> i32_t
  | A.Bool -> i1_t
  | A.Void -> void_t
  | A.String -> i8_pt
  | A.Float -> f_t
  | Object(name) ->  L.pointer_type(find_object_type name)
  | A.Arraytype(t) -> L.pointer_type (ltype_of_typ t)
  |    d -> raise(Failure ("Met unknown type in ltype_of_typ\n") )

(* Return the value for a variable or formal or global argument *)
and lookup the_name =
  try H.find locals_table the_name
  with | Not_found ->
    (try  H.find params_table the_name
     with | Not_found ->
       (try H.find globals_table the_name
        with | Not_found -> raise (Failure("Unbound var " ^ the_name))
```

```
      )
    )

(* Get the built function from the module *)
and  func_lookup fname =
  match (L.lookup_function fname the_module) with
    None      -> raise (Failure ("No function "^fname) )
  | Some f    -> f

let rec codegen_globals globals =
  ignore(log_to_file "Codegen globals\n");

  (* Declare each global variable; remember its value in a map *)
  let add_to_global_table table (t, n) =
    let init = L.const_int (ltype_of_typ t) 0 in
    H.add table n (L.define_global n init the_module)
  in
  ignore(List.map (add_to_global_table globals_table) globals);
  ignore(print_hashtable globals_table);
  log_to_file "Generated globals\n"

(* Define each function (arguments and return type) so we can call it *)
and codegen_function_decls functions =
  ignore(log_to_file "Codegen function declarations\n");
  ignore(List.map (fun f -> log_to_file ("This time build function "^f.S.s_fname^"\n") )
functions);

  let add_to_func_table table fdecl =
    let name = fdecl.S.s_fname
    and formal_types =
      Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.S.s_formals)
    in
    let ftype = L.function_type (ltype_of_typ fdecl.S.s_typ) formal_types in
    H.add table name (L.define_function name ftype the_module, fdecl)
  in
  ignore(List.map (add_to_func_table func_table) functions);
  log_to_file "Generated function declarations\n"


and codegen_builtins func_decls =
  ignore(log_to_file "Codegen builtins\n");
  (* Declare printf(), which the print built-in function will call *)
  let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let printf_func = L.declare_function "printf" printf_t the_module in

  (* Declare malloc *)
  let malloc_ty = L.function_type (i8_pt) [| i32_t |] in
  let malloc_func = L.declare_function "malloc" malloc_ty the_module in

  (* Declare sprintf *)
  let sprintf_ty = L.var_arg_function_type i32_t [| L.pointer_type i8_t; L.pointer_type i8_t
|] in
```

```
  let sprintf_func = L.declare_function "sprintf" sprintf_ty the_module in

  (* Declare strcpy *)
  let strcpy_ty = L.function_type(L.pointer_type(i8_t)) [| L.pointer_type i8_t;
L.pointer_type i8_t  |] in
  let strcpy_func = L.declare_function "strcpy" strcpy_ty the_module in

  (* Declare strlen *)
  let strlen_ty = L.function_type i32_t [| L.pointer_type i8_t; |] in
  let strlen_func = L.declare_function "strlen" strlen_ty the_module in

  (* Declare strcat *)
  let strcat_ty = L.function_type(L.pointer_type(i8_t)) [| L.pointer_type i8_t;
L.pointer_type i8_t  |] in
  let strcat_func = L.declare_function "strcat" strcat_ty the_module in

  (* Declare strcmp *)
  let strcmp_ty = L.function_type i32_t  [| L.pointer_type i8_t; L.pointer_type i8_t  |] in
  let strcmp_func = L.declare_function "strcmp" strcmp_ty the_module in

  (* Declare atoi *)
  let atoi_ty = L.function_type i32_t  [| L.pointer_type i8_t; |] in
  let atoi_func = L.declare_function "atoi" atoi_ty the_module in

  (* Declare atof *)
  let atof_ty = L.function_type f_t  [| L.pointer_type i8_t; |] in
  let atof_func = L.declare_function "atof" atof_ty the_module in

  (* Dump the module to check builtin functions *)
  ignore(log_to_file "Generated builtin functions\n")
(* L.dump_module the_module *)

(* Generate classes *)
and  codegen_struct_stub class_struct =
  let struct_t = L.named_struct_type context class_struct.S.s_cname in
  H.add object_types class_struct.S.s_cname struct_t


and codegen_struct class_struct =
  let struct_t = H.find object_types class_struct.S.s_cname in
  let type_list = List.map (fun (data_type,name) -> ltype_of_typ data_type)
class_struct.S.s_fields in
  let name_list = List.map (fun (data_type, name) -> name) class_struct.S.s_fields in

  let type_array = (Array.of_list type_list) in
  List.iteri (fun index field ->
      let n = class_struct.S.s_cname ^ "." ^ field in
      H.add object_field_indices n index;
    ) name_list;
  L.struct_set_body struct_t type_array true

(* Pass constructor to this function *)
```

```
and codegen_obj_create fname expr_list data_type llbuilder =
  (* Call the full name of the function *)
  let class_name = match data_type with
      A.Object(obj_name) -> obj_name
    | d -> raise (Failure ("Data type in Obj_Create is not an object but "^(A.string_of_typ
d) ) )
  in
  let string_of_params params =
    String.concat "." (List.map (fun p -> (S.string_of_expr_type p ) ) params )
  in
  let param_str = (match expr_list with
        [] -> ""
      | _ -> "." ^ (string_of_params expr_list)
    )in
  let full_name = (class_name ^ ".constructor"^param_str) in
  let f = func_lookup full_name in
  let generate_param_func builder isReturn param =
    match param with
      S.S_Id(id,A.Object(obj_name)) ->
      lookup id
    | _ ->
      codegen_sexpr llbuilder false param
  in
  let params = List.map (generate_param_func llbuilder false) expr_list in
  let obj = L.build_call f (Array.of_list params) "tmp" llbuilder in
  obj

and codegen_id id d llbuilder =
  try  (
    let _val = H.find locals_table id in
    ignore(log_to_file ("Found "^id^" in local table\n") );
    L.build_load _val id llbuilder
  )
  with | Not_found ->
    (try let _val = H.find params_table id in
       ignore(log_to_file ("Found "^id^" in params\n") );
       _val
     with | Not_found ->
       (try let _val = H.find globals_table id in
          ignore(log_to_file ("Found "^id^" in global table\n") );
          L.build_load _val id llbuilder
        with | Not_found -> raise (Failure("Unbound var " ^ id))
       )
    )
and codegen_id_obj_in_param id d llbuilder =
  lookup id

(* Very possibly we need to rewrite this *)
and codegen_obj_access isAssign the_obj the_field data_type llbuilder =
  ignore(log_to_file "Building obj access\n");
  ignore(log_to_file ((S.string_of_s_expr the_obj)^"\n") );
  ignore(log_to_file ((S.string_of_s_expr the_field)^"\n") );
```

```
  ignore(log_to_file "Now we need object field index table\n");
  ignore(print_hashtable_int object_field_indices);

  (* LHS can be ID or a nested Obj_Access *)
  ignore(log_to_file "Generating parent\n");
  let codegen_lhs = function
      S.S_Id(id, data_type) ->
      ignore(log_to_file "LHS of obj access is just the obj\n");
      let _found = lookup id in
      _found
    | S.S_ArrayAccess(arrayName, index, t)  -> generate_array_access true arrayName index
llbuilder
    | S.S_ObjAccess(o,f,t)->
      (* Need to load lhs before accessing further *)
      let _val = codegen_obj_access isAssign o f t llbuilder in
      L.build_load _val "load_nested_obj" llbuilder
    |  se     ->
      ignore(log_to_file "Got unhandled LHS in obj access:\n");
      ignore(log_to_file ("Type is "^(S.string_of_s_expr se)^"\n" ) );
      raise( Failure "Unhandled case in LHS in obj_access\n ")
  in
  let parent = codegen_lhs the_obj in
  ignore(log_to_file ("Generated parent: "^(L.string_of_llvalue parent)^"\n") );

  (* RHS can be ID *)
  ignore(log_to_file "Generating field\n");

  let rec codegen_rhs (parent_expr:L.llvalue) (parent_type:A.typ) field_expr=
    match field_expr with
      S.S_Id(id,data_type) ->
      (
        (* Find type of parent *)
        let search_term = ((S.string_of_ast_typ parent_type) ^ "." ^ id) in
        let field_index = Hashtbl.find object_field_indices search_term in
        let _val = L.build_struct_gep parent_expr field_index id llbuilder in
        let _mat = match data_type with
            A.Object(name)->
            let _store =
              if isAssign then
                _val
              else begin
                let _load = L.build_load _val id llbuilder in
                _load
              end
            in
            _store
          | _ ->
              _val
        in
        _mat
      )
```

```
    | S_ArrayAccess(arrayName, index, t) ->
      let ce  = codegen_rhs parent_expr parent_type arrayName in
      let index = L.const_int i32_t index in
      let index = L.build_add index (L.const_int i32_t 1) "temp_afterAdd1" llbuilder in
      let _val = build_gep ce [| index |] "tmp" llbuilder in
      if isAssign
          then _val
          else build_load _val "tmp" llbuilder

    | S.S_Call (fname, act, _) ->
      ignore(log_to_file ("Build object function "^fname^" with expr: \n") );
      ignore(List.map (fun expr -> log_to_file ((S.string_of_s_expr expr) ^ "\n") ) act );
      let class_name = A.string_of_typ parent_type in
      let func_name = class_name^"."^fname in
      ignore(log_to_file ("Function name is "^func_name^"\n"));
      let (fdef, fdecl) = H.find func_table func_name in
      let actuals = List.rev (List.map (codegen_sexpr llbuilder true) (List.rev act)) in
      let result = (match fdecl.S.s_typ with A.Void -> ""
                                           | _ -> func_name ^ "_result") in
      L.build_call fdef (Array.of_list actuals) result llbuilder
    | se ->
      ignore(log_to_file "Got unhandled RHS in obj access:\n");
      ignore(log_to_file ("Type is "^(S.string_of_s_expr se)^"\n" ) );
      raise( Failure "Unhandled case in RHS in obj_access\n ")
  in


  let lhs_type = get_ast_type_of_sexpr the_obj in
  let lhs = codegen_lhs the_obj in
  let rhs = codegen_rhs lhs lhs_type the_field in
  rhs

(* Check type of obj field *)

and get_value deref vname builder =
  if deref then
    let var = try H.find locals_table vname with
      | Not_found -> try H.find params_table vname with
        | Not_found -> try H.find globals_table vname with
          | Not_found -> raise (Failure("unknown variable name " ^ vname))
    in
    L.build_load var vname builder

  else
    let var = try H.find locals_table vname with
      | Not_found -> try H.find params_table vname with
        | Not_found -> try H.find globals_table vname with
          | Not_found -> raise (Failure("unknown variable name " ^ vname))
    in
    var

and initialise_array arr arr_len init_val start_pos llbuilder =
```

```
  let new_block label =
    let f = L.block_parent (L.insertion_block llbuilder) in
    L.append_block (context) label f
  in
  let bbcurr = L.insertion_block llbuilder in
  let bbcond = new_block "array.cond" in
  let bbbody = new_block "array.init" in
  let bbdone = new_block "array.done" in
  ignore (L.build_br bbcond llbuilder);
  L.position_at_end bbcond llbuilder;

  (* Counter into the length of the array *)
  let counter = L.build_phi [L.const_int i32_t start_pos, bbcurr] "counter" llbuilder in
  add_incoming ((L.build_add counter (L.const_int i32_t 1) "tmp" llbuilder), bbbody)
counter;
  let cmp = build_icmp Icmp.Slt counter arr_len "tmp" llbuilder in
  ignore (L.build_cond_br cmp bbbody bbdone llbuilder);
  position_at_end bbbody llbuilder;

  (* Assign array position to init_val *)
  let arr_ptr = build_gep arr [| counter |] "tmp" llbuilder in
  ignore (build_store init_val arr_ptr llbuilder);
  ignore (build_br bbcond llbuilder);
  position_at_end bbdone llbuilder

(* Generate 1 dimension array *)
and generate_one_d_array typ size builder =
  let t = ltype_of_typ typ in

  let size = (L.const_int i32_t size) in

  let size_t = L.build_intcast (L.size_of t) i32_t "1tmp" builder in

  let size = L.build_mul size_t size "2tmp" builder in  (* size * length *)
  let size_real = L.build_add size (L.const_int i32_t 1) "arr_size" builder in

  let arr = L.build_array_malloc t size_real "333tmp" builder in
  let arr = L.build_pointercast arr (L.pointer_type t) "4tmp" builder in

  let arr_len_ptr = L.build_pointercast arr (L.pointer_type i32_t) "5tmp" builder in

  ignore(L.build_store size_real arr_len_ptr builder);
  initialise_array arr_len_ptr size_real (L.const_int i32_t 0) 0 builder;
  arr

and generate_array_access deref arrayName index builder =
  (* let _ = print_int index in *)
  let index = L.const_int i32_t index in
  let index = L.build_add index (L.const_int i32_t 1) "temp_afterAdd1" builder in
  let arr = match arrayName with
    | S.S_Id(name, _) -> get_value true name builder
    | _ -> raise(Failure("No such arrayName:"))
```

```
  in
  let _val = L.build_gep arr [| index |] "2tmp" builder in
  if deref
  then L.build_load _val "3tmp" builder
  else _val

(* Construct code for an expression; return its value *)
and codegen_assign lhs rhs llbuilder =
  ignore(log_to_file ("Building assignment: "^(S.string_of_s_expr lhs)^" =
"^(S.string_of_s_expr rhs)^"\n" ) );
  (match lhs with
     S.S_Id (s,t) ->
     (* ignore(print_string ("print type is " ^ A.string_of_typ t)); *)
     let e2' = codegen_sexpr llbuilder false rhs in
     if (String.compare s "this")=0 then begin
       ignore(log_to_file ("LHS of assign is "^s^". Ignore.\n"));e2'
     end
     else begin
       ignore(log_to_file ("LHS of assign is Id "^s^"\n") );
       (* If the rhs is object type, load it first *)
       let rhs_type = get_ast_type_of_sexpr rhs in

       (* Check AST type of expr *)
       ignore(log_to_file ("RHS has AST type "^(A.string_of_typ rhs_type)^"\n"));

       (* Decide whether to load by checking rhs type *)
       match rhs with
         S.S_Literal(_)| S.S_BoolLit(_)| S.S_FloatLit(_)| S.S_Unop(_, _, _)| S.S_Binop(_, _,
_, _)
       | S.S_Cast(_,_,_) ->
         ignore (L.build_store e2' (lookup s) llbuilder);
         e2'
       | S.S_Call(fname, _, data_type) ->
         ignore(log_to_file ("RHS of assign is function call of type "^(A.string_of_typ
data_type)^".\n"));
         let _val = L.build_store e2' (lookup s) llbuilder in
         ignore(log_to_file ("Load the value makes: "^(L.string_of_llvalue _val)^"\n"));
         _val
       | S.S_Id(name, data_type) -> (
           match data_type with
             A.Object(name) ->
             ignore(L.build_alloca (ltype_of_typ data_type) name llbuilder);
             ignore (L.build_store e2' (lookup s) llbuilder);
             e2'
           | _ ->
             ignore (L.build_store e2' (lookup s) llbuilder);
             e2'
         )
       | S.S_StringLit(_) | S.S_ArrayCreate(_, _) ->
         ignore (L.build_store e2' (lookup s) llbuilder);
         e2'
       | S.S_ObjAccess(obj,field,data_type) ->
```

```
         ignore(log_to_file ("RHS of assign is object access: "^(S.string_of_s_expr
rhs)^"\n"));
         (* Construct function name to see if it exists *)
         let field_name_trans sexpr = match sexpr with
             S.S_Call(f, el,t) -> f
           | S.S_Id(s,t) -> s
         in
         let class_of_obj sexpr =
           ignore(log_to_file ("Matching sexpr "^(S.string_of_s_expr sexpr)^"\n"));
           match sexpr with
             S.S_Id(_,t) -> A.string_of_typ t
           | S.S_ObjAccess(_,_,t) -> A.string_of_typ t
         in
         let obj_type = class_of_obj obj in
         let pseudo_func_name = obj_type^"."^(field_name_trans field) in
         ignore(log_to_file ("Accessing "^pseudo_func_name^"\n"));

         (* If rhs is function call, do not load *)
         let obj_field =  (match (L.lookup_function pseudo_func_name the_module) with
               None   ->
               (* The field is an id. Load it. *)
               ignore(log_to_file ("Not a function. Load as id: "^(L.string_of_llvalue
e2')^"\n"));
               let _val = L.build_load e2' "tmp" llbuilder in
               ignore(log_to_file ("Loaded: "^(L.string_of_llvalue _val)^"\n"));
               ignore (L.build_store _val (lookup s) llbuilder);
               _val
             | Some f        ->
               (* The field is a function. Do not load it. *)
               ignore(log_to_file "A function. Do not load it.\n");
               ignore (L.build_store e2' (lookup s) llbuilder);
               e2'
           )
         in
         obj_field
       | _ ->
         ignore(log_to_file "The rhs to be assigned to lhs is not literal. Load it first.
\n");
         ignore(log_to_file ("RHS expr is "^(S.string_of_s_expr rhs)^"\n"));
         let _val = L.build_load e2' "tmp" llbuilder in
         ignore (L.build_store _val (lookup s) llbuilder);
         _val
     end
   | S.S_ObjAccess(the_obj,the_field,data_type) ->
     ignore(log_to_file ("Building object access in LHS of assign: type  "^(A.string_of_typ
data_type)^"\n") );
     let e1' = codegen_obj_access true the_obj the_field data_type llbuilder in
     ignore(log_to_file ("Built LHS ObjAccess: "^(L.string_of_llvalue e1')^"\n"));
     let e2' = codegen_sexpr llbuilder true rhs in
     ignore(log_to_file ("Built RHS ObjAccess: "^(L.string_of_llvalue e2')^"\n"));
     let _val = L.build_store e2' e1' llbuilder in
     ignore(log_to_file ("Built store: "^(L.string_of_llvalue _val)^"\n"));
```

```
    e2'

  | S.S_ArrayAccess(e, index, typ) ->
    let vmemory = generate_array_access false e index llbuilder in
    let value = codegen_sexpr llbuilder false rhs in
    ignore (L.build_store value vmemory llbuilder);
    value
  | _ ->
    let e1' = codegen_sexpr llbuilder false lhs and e2' = codegen_sexpr llbuilder false rhs
in
    ignore (L.build_store e2' e1' llbuilder); e2'
  )

(* Construct code for an expression; return its value *)
and codegen_sexpr builder isReturn (sexpr:S.s_expr) =
  ignore(log_to_file ("Building expression: "^(S.string_of_s_expr sexpr)^"\n") );

  match sexpr with
    S.S_Literal i ->
      L.const_int i32_t i
  | S.S_BoolLit b -> L.const_int i1_t (if b then 1 else 0)
  | S.S_FloatLit(f)                -> L.const_float f_t f
  | S.S_StringLit s  ->      L.build_global_stringptr s "tmp_string" builder
  | S.S_Noexpr -> L.const_int i32_t 0
  | S.S_Null -> L.const_null i32_t
  | S.S_Id (s,data_type) ->
    if isReturn then begin
      ignore(log_to_file ("Id "^s^" is to be returned.\n"));
      let to_ret = match data_type with
          A.Object(name) ->
          lookup s
        |_ ->
          L.build_load (lookup s) s builder
      in
      to_ret
    end
    else begin
      ignore(log_to_file ("The Id is "^s^"\n"));
      L.build_load (lookup s) s builder
    end
  | S.S_ArrayCreate(typ, size) -> generate_one_d_array typ size builder
  | S.S_ArrayAccess(arrayName, index, t) -> generate_array_access true arrayName index
builder
  | S.S_Binop (e1, op, e2,_) ->
    ignore(log_to_file ("Building binop "^(S.string_of_s_expr e1)^","^(A.string_of_op
op)^","^(S.string_of_s_expr e2)^"\n" ) );

    let e1' = codegen_sexpr builder false e1
    and e2' = codegen_sexpr builder false e2 in

    ignore(log_to_file ( ("\n"^L.string_of_llvalue e1')^"\n" ) );
    ignore(log_to_file ( (L.string_of_llvalue e2')^"\n" ) );
```

```
  if (ANA.get_sexpr_type(e1) = A.String ) then
    (match op with
       A.Add -> codegen_strcat e1' e2' builder
     | _ -> raise(Failure("unsupported string opreators"))
    )
  else if (ANA.get_sexpr_type(e1) = A.Float ) then
    (match op with
       A.Add     -> L.build_fadd
     | A.Sub     -> L.build_fsub
     | A.Mult    -> L.build_fmul
     | A.Div     -> L.build_fdiv
     | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
     | A.Neq     -> L.build_fcmp L.Fcmp.One
     | A.Less    -> L.build_fcmp L.Fcmp.Olt
     | A.Leq     -> L.build_fcmp L.Fcmp.Ole
     | A.Greater -> L.build_fcmp L.Fcmp.Ogt
     | A.Geq     -> L.build_fcmp L.Fcmp.Oge
     | _ -> raise(Failure("unsupported float opreators"))
    ) e1' e2' "tmp" builder
  else
    (match op with
       A.Add     -> L.build_add
     | A.Sub     -> L.build_sub
     | A.Mult    -> L.build_mul
     | A.Div     -> L.build_sdiv
     | A.And     -> L.build_and
     | A.Or      -> L.build_or
     | A.Equal   -> L.build_icmp L.Icmp.Eq
     | A.Neq     -> L.build_icmp L.Icmp.Ne
     | A.Less    -> L.build_icmp L.Icmp.Slt
     | A.Leq     -> L.build_icmp L.Icmp.Sle
     | A.Greater -> L.build_icmp L.Icmp.Sgt
     | A.Geq     -> L.build_icmp L.Icmp.Sge
    ) e1' e2' "tmp" builder
| S.S_Unop(op, e, _) ->
  ignore(log_to_file ("Building unop "^(S.string_of_s_expr e)^"\n" ) );
  let e' = codegen_sexpr builder false e in
  (match (op)  with
     A.Neg     ->
     if (ANA.get_sexpr_type(e) = A.Float ) then
       L.build_fneg
     else
       L.build_neg
   | A.Not     -> L.build_not
  ) e' "tmp" builder

| S.S_Assign (lhs,rhs, _) ->
  codegen_assign lhs rhs builder
| S.S_Cast(t1,e,t2) ->
  (
    match (t1,t2) with
```

```
        (Int,Int) | (Float,Float) | (Bool,Bool) | (String,String) -> codegen_sexpr builder
false e
      | (Float,Int) ->
        let e' = codegen_sexpr builder false e in
        L.build_fptosi e' i32_t "float_to_int" builder
      | (Int,Float) ->
        let e' = codegen_sexpr builder false e in
        L.build_sitofp e' f_t "int_to_float" builder
      | (String,Int) ->
        let atoi_func = func_lookup "atoi" in
        L.build_call atoi_func [| (codegen_sexpr builder false e) |] "atoi" builder
      | (String,Float) ->
        let atof_func = func_lookup "atof" in
        L.build_call atof_func [| (codegen_sexpr builder false e) |] "atof" builder
      | (Float,String) -> codegen_tostring e builder
      | (Int,String) -> codegen_tostring e builder
      | (Bool,String) -> codegen_tostring e builder
      | _ -> raise(Failure("cast not implemented"))
    )
  |  S.S_Call ("print",[e],_ ) ->
    let printf_format =
      let temp = ANA.get_sexpr_type(e) in
      (match temp with
         A.Int | A.Bool -> int_format_str
       | A.String -> string_format_str
       | A.Float -> float_format_str
       | A.Arraytype(t) -> (match t with
            A.Int | A.Bool -> int_format_str
          | A.String -> string_format_str
          | A.Float -> float_format_str
        )
       | _ -> raise(Failure("print only supports primitive types") )
      ) in
    let printf_func = func_lookup "printf" in
    let _val =
      match e with
        S.S_ObjAccess(_,_,_) ->
        L.build_load ((codegen_sexpr builder false e)) "var_for_print" builder
      | _ ->
        codegen_sexpr builder false e
    in
    L.build_call printf_func [| (printf_format builder) ; _val |]
      "printf" builder

  | S.S_Call ("toString",[e],_) ->
    codegen_tostring e builder
  | S.S_Call ("strcmp",e,_) ->
    let strcmp_func = func_lookup "strcmp" in
    let actuals = List.rev (List.map (codegen_sexpr builder false) (List.rev e)) in
    L.build_call strcmp_func (Array.of_list actuals) "strcmp" builder
  (* Add sizeof *)
  | S.S_Call ("sizeof",[e],data_type) ->
```

```
    codegen_sizeof e data_type builder
  (* Add malloc *)
  | S.S_Call ("malloc",[e],data_type) ->
    codegen_malloc e data_type builder
  (* Add cast *)
  | S.S_Call ("cast",[e],data_type) ->
    codegen_cast e data_type builder


  | S.S_Call (fname, act, _) ->
    ignore(log_to_file ("Build function "^fname^" with expr: \n") );
    ignore(List.map (fun expr -> log_to_file ((S.string_of_s_expr expr) ^ "\n") ) act );

    let (fdef, fdecl) = H.find func_table fname in
    let actuals = List.rev (List.map (codegen_sexpr builder true) (List.rev act)) in
    let result = (match fdecl.S.s_typ with A.Void -> ""
                                         | _ -> fname ^ "_result") in
    L.build_call fdef (Array.of_list actuals) result builder
  |   S_ObjCreate(id, expr_list, data_type) -> codegen_obj_create id expr_list data_type
builder
  |   S_ObjAccess(e1, e2, d) -> codegen_obj_access (not isReturn) e1 e2 d builder


(* sizeof *)
and codegen_sizeof sexpr data_type llbuilder =
  ignore(log_to_file ("Building sizeof with data_type:"^(A.string_of_typ data_type)^"\n") );
  ignore(log_to_file ("Expr: "^(S.string_of_s_expr sexpr )^"\n"   ) );

  let type_of_el = get_underlying_type_of_sexpr sexpr in
  let size_of_el = L.size_of type_of_el in
  L.build_bitcast size_of_el i32_t "tmp" llbuilder


and codegen_tostring e llbuilder =
  let ll_expr = codegen_sexpr llbuilder false e in
  let t = ANA.get_sexpr_type(e) in
  let (size, sprintf_format) =
    (match (t) with
       A.Int | A.Bool ->
       (L.const_int i32_t 20, int_sprintf_str(llbuilder))
     | A.Float ->
       (L.const_int i32_t 20, float_sprintf_str(llbuilder))
     | A.String ->
       let strlen_func = func_lookup "strlen" in
       let len =  L.build_call strlen_func [| (ll_expr); |] "strlen" llbuilder in
       let len = L.build_add len (L.const_int i32_t 1) "tmp" llbuilder in
       (L.const_int i32_t 20, string_sprintf_str(llbuilder))
     | _ ->  raise (Failure ("toString() function only supports primitive types") )
    ) in
  let t = i8_t in
  let buf = L.build_array_malloc t size "to_string_buf" llbuilder in
  let buf = L.build_pointercast buf (L.pointer_type t) "to_string_buf" llbuilder in
```

```
  let sprintf_func = func_lookup "sprintf" in
  let _ = L.build_call sprintf_func [| buf; sprintf_format ; ll_expr; |]
      "sprintf" llbuilder in
  buf

and codegen_strcat e1' e2' llbuilder =
  let ll_expr1 = e1' in
  let ll_expr2 = e2' in
  let strlen_func = func_lookup "strlen" in
  let strcpy_func = func_lookup "strcpy" in
  let strcat_func = func_lookup "strcat" in
  let len1 =  L.build_call strlen_func [| (ll_expr1); |] "strlen" llbuilder in
  let len2 =  L.build_call strlen_func [| (ll_expr2); |] "strlen" llbuilder in
  let len_new = L.build_add len1 len2 "tmp" llbuilder in
  let size = L.build_add len_new (L.const_int i32_t 1) "tmp" llbuilder in
  let t = i8_t in
  let buf = L.build_array_malloc t size "to_string_buf" llbuilder in
  let buf = L.build_pointercast buf (L.pointer_type t) "to_string_buf" llbuilder in
  let buf = L.build_call strcpy_func [| buf; ll_expr1; |] "strcpy" llbuilder in
  let buf = L.build_call strcat_func [| buf; ll_expr2; |] "strcat" llbuilder in
  buf

and codegen_malloc sexpr data_type llbuilder =
  ignore(log_to_file ("Building malloc with data_type:"^(A.string_of_typ data_type)^"\n") );
  ignore(log_to_file ("Expr: "^(S.string_of_s_expr sexpr )^"\n"   ) );

  let f = func_lookup "malloc" in
  let params = List.map (codegen_sexpr llbuilder false ) [sexpr] in
  match data_type with
    A.Void -> L.build_call f (Array.of_list params) "" llbuilder
  |    _ ->                         L.build_call f (Array.of_list params) "tmp" llbuilder

and codegen_cast sexpr data_type llbuilder =
  ignore(log_to_file ("Building cast with data_type:"^(A.string_of_typ data_type)^"\n") );
  ignore(log_to_file ("Expr: "^(S.string_of_s_expr sexpr )^"\n"   ) );

  let cast_malloc_to_objtype lhs currType newType llbuilder = match newType with
      A.Object(name)->
      let obj_type = ltype_of_typ (A.Object(name)) in
      ignore(log_to_file ("In cast obj type is "^(L.string_of_lltype obj_type)^"\n")) ;
      L.build_pointercast lhs obj_type "tmp" llbuilder
    |   _ as t -> raise (Failure("I don't know why Im here but this is wrong"))
  in
  let t = ANA.get_sexpr_type sexpr in
  let lhs = match sexpr with
    |   S.S_Id(id, data_type) ->
      lookup id
    | _ -> codegen_sexpr llbuilder false sexpr
  in
  cast_malloc_to_objtype lhs t data_type llbuilder
```

```
(* Need to be moved*)
and int_format_str builder = L.build_global_stringptr "%d\n" "fmt" builder
and string_format_str builder = L.build_global_stringptr "%s\n" "fmt" builder
and float_format_str builder = L.build_global_stringptr "%f\n" "fmt" builder
and int_sprintf_str builder = L.build_global_stringptr "%d" "tostr_fmt" builder
and string_sprintf_str builder = L.build_global_stringptr "%s" "tostr_fmt" builder
and float_sprintf_str builder = L.build_global_stringptr "%f" "tostr_fmt" builder


and codegen_classes (classes:S.s_class_decl list) =
  ignore(log_to_file "Codegen classes\n");
  ignore(List.map (fun c -> log_to_file ("To build class "^c.S.s_cname^"\n") ) classes);

  let codegen_one_class (the_class:S.s_class_decl) =
    ignore(log_to_file ("Generating class "^the_class.s_cname^"\n") );

    (* Generate struct_ decl *)
    let _ = codegen_struct_stub the_class in

    (* Generate class_ local vars *)
    let _ = codegen_struct the_class in

    (* Generate functions *)
    ignore(log_to_file "Finished class definition. Now start with functions\n");
    ignore(List.map (fun f -> log_to_file ("To build constructor "^f.S.s_fname^"\n") )
the_class.s_constructors);
    ignore(List.map (fun f -> log_to_file ("To build method "^f.S.s_fname^"\n") )
the_class.s_methods);

    let _ = codegen_function_decls the_class.s_constructors in
    let _ = codegen_function_decls the_class.s_methods in
    let _ = codegen_fbody the_class.s_constructors in

    let _ = codegen_fbody the_class.s_methods in
    ignore(log_to_file ("Generated class "^the_class.s_cname^"\n") );
  in
  List.map codegen_one_class classes;
  ignore(log_to_file "Finished generating classes\n")

and find_struct name =
  try Hashtbl.find object_types name
  with | Not_found -> raise(Failure ("Cannot find object type "^name) )

and codegen_fbody functions =

  (* Fill in the body of the given function *)
  let build_function_body fdecl =
    let _ = log_to_file ("Building function " ^ fdecl.S.s_fname ^ "\n" ) in

    (* First clear local and param tables *)
    let _ = H.clear locals_table; H.clear params_table in
```

```
    (* Find target function *)
    let (the_function, _) = H.find func_table fdecl.S.s_fname in
    let builder = L.builder_at_end context (L.entry_block the_function) in

    (* Construct the function's "locals": formal arguments and locally
       declared variables.  Allocate each on the stack, initialize their
       value, if appropriate, and remember their values in the "locals" map *)
    let codegen_locals local_vars =
      let add_formal table (the_type,name) value =
        L.set_value_name name value;
        let local = match the_type with
            A.Object(name) ->
            value
          | _ ->
            let _val = L.build_alloca (ltype_of_typ the_type) name builder in
            ignore (L.build_store value _val builder); (* local is the pointer we store at
*)
            _val
        in
        H.add table name local
      in
      let add_local table (the_type,name) =
        let t =
          match the_type with
            A.Object(cname) ->
            find_struct cname
          | _ ->
            ltype_of_typ the_type
        in

        let local_var = L.build_alloca t name builder
        in
        H.add table name local_var
      in
      let _ = List.map2 (add_formal params_table) fdecl.S.s_formals (Array.to_list (L.params
the_function))
      in
      ignore(List.map (add_local locals_table) local_vars);

      (* Log to file all the locals and formals *)
      ignore(log_to_file "Logging local table\n");
      print_hashtable locals_table
    in
    codegen_locals fdecl.S.s_locals;

    (* Invoke "f builder" if the current block doesn't already
       have a terminal (e.g., a branch). *)
    let add_terminal builder f =
      match L.block_terminator (L.insertion_block builder) with
        Some _ -> ()
      | None -> ignore (f builder)
    in
```

```
    (* Build the code for the given statement; return the builder for
       the statement's successor *)
    let rec codegen_stmt builder stmt =
      match stmt with
        S.S_Block sl ->
        List.fold_left codegen_stmt builder sl
      | S.S_Expr (e,_) ->
        ignore (codegen_sexpr builder false e); builder
      | S.S_Return (e,_) ->
        ignore(log_to_file ("Building return expr "^(S.string_of_s_expr e)^"\n" ) );
        ignore (match fdecl.S.s_typ with
              A.Void -> L.build_ret_void builder
            | _ -> L.build_ret (codegen_sexpr builder true e) builder); builder
      | S.S_If (predicate, then_stmt, else_stmt) ->
        let bool_val = codegen_sexpr builder false predicate in
        let merge_bb = L.append_block context "merge" the_function in

        let then_bb = L.append_block context "then" the_function in
        add_terminal (codegen_stmt (L.builder_at_end context then_bb) then_stmt)
          (L.build_br merge_bb);

        let else_bb = L.append_block context "else" the_function in
        add_terminal (codegen_stmt (L.builder_at_end context else_bb) else_stmt)
          (L.build_br merge_bb);

        ignore (L.build_cond_br bool_val then_bb else_bb builder);
        L.builder_at_end context merge_bb

      | S.S_While (predicate, body) ->
        let pred_bb = L.append_block context "while" the_function in
        ignore (L.build_br pred_bb builder);

        let body_bb = L.append_block context "while_body" the_function in
        add_terminal (codegen_stmt (L.builder_at_end context body_bb) body)
          (L.build_br pred_bb);

        let pred_builder = L.builder_at_end context pred_bb in
        let bool_val = codegen_sexpr pred_builder false predicate in

        let merge_bb = L.append_block context "merge" the_function in
        ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
        L.builder_at_end context merge_bb

      | S.S_For (e1, e2, e3, body) -> codegen_stmt builder
                                        ( S.S_Block [S.S_Expr(e1,ANA.get_sexpr_type(e1)) ;
                                                     S.S_While (e2, S.S_Block [body ;
S.S_Expr(e3,ANA.get_sexpr_type(e3))]) ] )
    in

    (* Build the code for each statement in the function *)
    let builder_stmt_built = codegen_stmt builder (S.S_Block fdecl.S.s_body)
    in
```

```
    (* Add a return if the last block falls off the end *)
    add_terminal builder_stmt_built (match fdecl.S.s_typ with
          A.Void -> L.build_ret_void
        | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in
  List.iter build_function_body functions

let codegen_sprogram (program:S.s_program) =
  (* First wipe out log file *)
  let oc = open_out_gen [Open_creat; Open_trunc; Open_text; Open_wronly] 0o640 "a.txt" in
  let _ = output_string oc "Start\n"; close_out oc in

  (* Build builtin function declarations *)
  let _ = codegen_builtins program.S.builtins in

  let _ = codegen_globals program.S.global_vars in

  let _ = codegen_classes program.S.classes in

  (* Generate functions here *)
  let _ = codegen_function_decls program.S.functions in
  let _ = codegen_function_decls (program.S.main::[]) in
  let _ = codegen_fbody program.S.functions in
  let _ = codegen_fbody (program.S.main::[]) in

  the_module
```

8.2 Test suites

Testall.sh (Liu)

```
#!/bin/sh

# Regression testing script for MicroC
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"
```

```
# Path to the microc compiler.  Usually "./microc.native"
# Try "_build/microc.native" if ocamlbuild was unable to create a symbolic link.
MICROC="./lava.native"
#MICROC="_build/microc.native"

# Set time limit for all operations
ulimit -t 30

# Global log
globallog=testall.log
rm -f $globallog

# Error flags
error=0
globalerror=0

# Default keep temp files
keep=1

# Report purpose
pos_pass=0
pos_fail=0
neg_pass=0
neg_fail=0
RED='\033[0;31m'
GREEN='\033[0;32m'
CYAN='\033[0;36m'
WHITE='\033[0m'

Usage() {
    echo "Usage: testall.sh [options] [.mc files]"
    echo "-k    Keep intermediate files"
    echo "-h    Print this help"
    exit 1
}

SignalError() {
    if [ $error -eq 0 ] ; then
       echo "${RED}FAILED${WHITE}"
       error=1
    fi
    echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
       SignalError "$1 differs from $2"
       echo "FAILED $1 differs from $2" 1>&2
```

```
    }
}

CompareFail() {
    generatedfiles="$generatedfiles $3"
    s1=`cat $1`
    # echo "Generated err msg: $s1"
    s1=`cat $2` # This doesn't work, dunno why'
    # echo "Expected err msg: $s2"
    echo diff -b $1 $2 ">" $3 1>&2

    expected="Fatal error:"
    if [ -z "${s1##*$expected*}" ]; then
    # if [[ $s1 =~ .*Fatal error:.* ]];then
    # if [[ $s1 == *"Fatal error: exception"* ]];then
        echo "${GREEN}Failure as expected${WHITE}"
    else
        SignalError "$1 doesn't contain an error!"
        echo "Output: $s1" 1>&2
    fi
}


# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    eval $* || {
       SignalError "${RED}$1 failed on $*${WHITE}"
       return 1
    }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
    echo $* 1>&2
    eval $* && {
       SignalError "${RED}failed: $* did not report an error${WHITE}"
       return 1
    }
    return 0
}

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                             s/.mc//'`
    reffile=`echo $1 | sed 's/.mc$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."
```

```
    echo 1>&2
    echo "###### Testing $basename" 1>&2

    # Generate target dir
    dirname=$2/testrun/

    generatedfiles=""

    generatedfiles="$generatedfiles ${dirname}${basename}.ll ${dirname}${basename}.out" &&
    Run "$MICROC" "<" $1 ">" "${dirname}${basename}.ll" &&
    Run "$LLI" "${dirname}${basename}.ll" ">" "${dirname}${basename}.out" &&
    Compare ${dirname}${basename}.out ${reffile}.out ${dirname}${basename}.diff

    if [ $error -eq 0 ] ; then
       if [ $keep -eq 0 ] ; then
           rm -f $generatedfiles
       fi
        echo "${GREEN}OK${WHITE}"
        echo "###### SUCCESS" 1>&2
        ((pos_pass++))
        # echo "Now passed pos tests: $pos_pass"
    else
        # echo -n "${RED}$basename...${WHITE}"
        echo "###### FAILED" 1>&2
        globalerror=$error
        ((pos_fail++))
    fi
}

CheckFail() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                             s/.mc//'`
    reffile=`echo $1 | sed 's/.mc$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    # Generate target dir
    dirname=$2/testrun/
    locallog=$2/testall.log

    generatedfiles=""

    generatedfiles="$generatedfiles ${dirname}${basename}.err ${dirname}${basename}.diff" &&
    RunFail "$MICROC" "<" $1 "2>" "${dirname}${basename}.err" ">>" $locallog &&
    CompareFail ${dirname}${basename}.err ${reffile}.err ${dirname}${basename}.diff
```

```
    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
         echo "${GREEN}OK${WHITE}"
         echo "###### SUCCESS" 1>&2
         let neg_pass=neg_pass+1
    else
            # echo -n "${RED}$basename...${WHITE}"
         echo "${RED}###### FAILED${WHITE}" 1>&2
            globalerror=$error
         let neg_fail=neg_fail+1
    fi
}

while getopts kdpsh c; do
    case $c in
        k) # Keep intermediate files
            keep=1
            ;;
        h) # Help
            Usage
            ;;
    esac
done

shift `expr $OPTIND - 1`

LLIFail() {
  echo "Could not find the LLVM interpreter \"$LLI\"."
  echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
  exit 1
}

which "$LLI" >> $globallog || LLIFail


if [ $# -ge 1 ]
then
    folders=$@
else
    # files="tests/**/test-*.mc tests/**/fail-*.mc"
    folders="tests/*"
fi

# Generate temp folders for holding temp files
echo "PWD: $PWD"
for folder in $folders
do
    temp_folder="$PWD/$folder/testrun/"
```

```
    mkdir -p $temp_folder
    rc=$? # return code of mkdir
    if [ $rc != 0 ]
    then echo "mkdir in $temp_folder exited with code $rc"
    fi
done

# Test folder by folder
for folder in $folders
do
    echo "\nTesting in $folder"
    # log path
    locallog="$folder/testall.log"
    rm -f $locallog

    # temp file path
    tempfilefolder=$folder/testrun
    echo "Put temp files in $tempfilefolder"
    rm -Rf tempfilefolder
    mkdir -p tempfilefolder

    # test files
    files="$folder/test-*.mc $folder/fail-*.mc"
    echo "Files to work on: $files"

    for file in $files
    do
        case $file in
        *test-*)
            Check $file $folder 2>> $locallog
            ;;
        *fail-*)
            CheckFail $file $folder 2>> $locallog
            ;;
        *)
            echo "${RED}unknown file type $file${WHITE}"
            globalerror=1
            ;;
        esac
    done
done

# reporting
# echo "Passed pos tests: $pos_pass"
# echo "Failed pos tests: $pos_fail"
# echo "Passed neg tests: $neg_pass"
# echo "Failed neg tests: $neg_fail"


exit $globalerror
```

Test cases (Liu, Wei, Yuan, Wang)

To demonstrate all the test cases clearly, we put all the stuffs together. Note that actually each test case is in an independent file.

```
/*      fail-float_arith1.mc  */
int main()
{
        float f;
        f = 1.0;
        print(f + "aaa");
        return 0;
}

/*
Output
        Fatal error: exception Failure("illegal binary operator float + string in f + aaa")

*/

/*      fail-float_arith2.mc  */
int main()
{
        float f;
        f = 1.0;
        print(f * true);
        return 0;
}

/*
Output
        Fatal error: exception Failure("illegal binary operator float * bool in f * true")

*/

/*      fail-float_arith3.mc  */
int main()
{
        float f;
        f = 1.0;
        print(f - "bbb");
        return 0;
}

/*
Output
        Fatal error: exception Failure("illegal binary operator float - string in f - bbb")

*/
```

```
/*      fail-float_arith4.mc  */
int main()
{
        float f;
        f = 1.0;
        print(f / false);
        return 0;
}

/*
Output
        Fatal error: exception Failure("illegal binary operator float / bool in f / false")

*/

/*      test-arith1.mc */
int main()
{
  print(39 + 3);
  return 0;
}
/*
Output
        42

*/

/*      test-arith2.mc */
int main()
{
  print(1 + 2 * 3 + 4);
  return 0;
}
/*
Output
        11

*/

/*      test-arith3.mc */
int foo(int a)
{
  return a;
}

int main()
{
  int a;
  a = 42;
  a = a + 5;
  print(a);
  return 0;
```

```
}
/*
Output
       47

*/

/*     test-arith_mulAndDiv.mc      */
int main()
{
  print(2*3);
  print(5/1);
  print(2/3);
  return 0;
}

/*
Output
       6

       5

       0

*/

/*     test-float_arith1.mc  */
int main()
{
  /* assign string */
  float a;
  float b;
  a = -3.5 + 1.0;
  b = a + 1.0;
  print(b);

  return 0;
}

/*
Output
       -1.500000
*/

/*     test-float_arith2.mc  */
int main()
{
  /* assign string */
  float a;
  float b;
  a = -3.5 + 1.0;
  b = a + 1.0;
```

```
  print(1.0+b*a);

  return 0;
}

/*
Output
       4.750000
*/

/*     test-float_arith3.mc  */
int main()
{
  /* assign string */
  float a;
  float b;
  a = -3.5 + 1.0;
  b = a + 1.0;
  print(b/a);

  return 0;
}

/*
Output
       0.600000
*/

/*     test-float_arith4.mc  */
int main()
{
  /* assign string */
  float a;
  float b;
  a = -3.5 + 1.0;
  b = a + 1.0;
  print(b-a);

  return 0;
}

/*
Output
       1.000000
*/

/*     test-float_arith5.mc  */
int main()
{
  /* assign string */
  float a;
  float b;
```

```
  a = -3.5 + 1.0;
  b = 0.0;
  print(a/b);

  return 0;
}
/*
Output
        -inf
*/

/*      test-float_arith_cmp1.mc      */
int main()
{
  float a;
  float b;
  a = 1.0;
  b = 1.0;
  if (a == b)
        print("correct");
  else
        print("not correct");

  return 0;
}
/*
Output
        correct
*/

/*      test-float_arith_cmp2.mc      */
int main()
{
  float a;
  float b;
  a = 1.0;
  b = 1.2;
  if (a < b)
        print("correct");
  else
        print("not correct");

  return 0;
}
/*
Output
        correct
*/

/*      test-float_arith_cmp3.mc      */
int main()
{
```

```
  float a;
  float b;
  a = 1.0;
  b = 1.0;
  if (a <= b)
       print("correct");
  else
       print("not correct");

  return 0;
}
/*
Output
       correct
*/

/*     test-float_arith_cmp4.mc     */
int main()
{
  float a;
  float b;
  a = 2.0;
  b = 1.0;
  if (a > b)
       print("correct");
  else
       print("not correct");

  return 0;
}
/*
Output
       correct
*/

/*     test-float_arith_cmp5.mc     */
int main()
{
  float a;
  float b;
  a = 1.0;
  b = 1.0;
  if (a >= b)
       print("correct");
  else
       print("not correct");

  return 0;
}
/*
Output
       correct
```

```
*/

/*     test-float_arith_cmp6.mc     */
int main()
{
  float a;
  float b;
  a = 2.0;
  b = 1.0;
  if (a != b)
       print("correct");
  else
       print("not correct");

  return 0;
}
/*
Output
       correct
*/

/*     test-array1.mc */
int main() {
  int[] array;
  array = new int[5];
  print(array[0]);
  print(array[4]);
  return 0;
}
/*
Output
       0

       0

*/

/*     test-array2.mc */
int main() {
  int[] array;
  array = new int[5];
  array[3] = 88;
  print(array[3]);
  return 0;
}
/*
Output
       88

*/

/*     test-array3.mc */
```

```
int main()
{
  float[] array;
  array = new float[10];
  array[5] = 3.14;
  print(array[0]);
  print(array[5]);
  return 0;
}
/*
Output
       0.000000

       3.140000

*/

/*     test-array4.mc */
int main()
{
  string[] array;
  array = new string[5];
  array[3] = "plt is a nice course";
  print(array[0]);
  print(array[3]);
  return 0;
}
/*
Output
       (null)

       plt is a nice course

*/

/*     fail-assign1.mc       */
int main()
{
  int i;
  bool b;

  i = 42;
  i = 10;
  b = true;
  b = false;
  i = false; /* Fail: assigning a bool to an integer */
}
/*
Output
       Fatal error: exception Failure("illegal assignment int = bool in $i[int] = false")

*/
```

```
/*      fail-assign2.mc       */
int main()
{
  int i;
  bool b;

  b = 48; /* Fail: assigning an integer to a bool */
}
/*
Output
       Fatal error: exception Failure("illegal assignment bool = int in $b[bool] = 48")

*/

/*      fail-assign3.mc       */
void myvoid()
{
  return;
}

int main()
{
  int i;

  i = myvoid(); /* Fail: assigning a void to an integer */
}
/*
Output
       Fatal error: exception Failure("illegal assignment int = void in $i[int] =
myvoid()[void]")

*/

/*      fail-assign_boolToFloat.mc    */
int main()
{
  float b;
  b = false;
  return 0;
}

/*
Output
       Fatal error: exception Failure("illegal assignment float = bool in $b[float] =
false")

*/

/*      fail-assign_boolToString.mc   */
int main()
{
```

```
  bool b;
  string s;
  b = false;
  s = b; /* assign bool to string */

  return 0;
}

/*
Output
       Fatal error: exception Failure("illegal assignment string = bool in $s[string] =
$b[bool]")

*/

/*     fail-assign_FloatToInt.mc    */
int main()
{
       int a;
       a = 3.5;
       return 0;
}

/*
Output
       Fatal error: exception Failure("illegal assignment int = float in $a[int] = 3.5")

*/

/*     fail-assign_voidToString.mc  */
void voidFunc(){
  return void;
}


int main()
{
  string s;
  s = voidFunc();
  return 0;
}

/*
Output
       Fatal error: exception Parsing.Parse_error

*/

/*     test-assign_float1.mc */
int main()
{
  /* assign string */
```

```
  float a;
  a = 3.5;
  print(a);

  return 0;
}

/*
Output
        3.500000
*/

/*      test-assign_float2.mc */
int main()
{
  /* assign string */
  float a;
  float b;
  a = -3.5;
  b = a;
  print(b);

  return 0;
}

/*
Output
        -3.500000
*/

/*      test-assign_string.mc */
int main()
{
  /* assign string */
  string s;
  s = "hello";
  print(s);

  return 0;
}

/*
Output
        hello

*/

/*      test-hello.mc  */
int main()
{
  print(42);
  print(71);
```

```
  print(1);
  return 0;
}
/*
Output
       42

       71

       1

*/

/*     test-hello_world.mc   */
int main()
{
  print("Hello World");
  return 0;
}
/*
Output
       Hello World

*/

/*     test-ops1.mc   */
int main()
{
  print(1 + 2);
  print(1 - 2);
  print(1 * 2);
  print(100 / 2);
  print(99);
  print(1 == 2);
  print(1 == 1);
  print(99);
  print(1 != 2);
  print(1 != 1);
  print(99);
  print(1 < 2);
  print(2 < 1);
  print(99);
  print(1 <= 2);
  print(1 <= 1);
  print(2 <= 1);
  print(99);
  print(1 > 2);
  print(2 > 1);
  print(99);
  print(1 >= 2);
  print(1 >= 1);
  print(2 >= 1);
```

```
  return 0;
}
/*
Output
        3

        -1

        2

        50

        99

        0

        1

        99

        1

        0

        99

        1

        0

        99

        1

        1

        0

        99

        0

        1

        99

        0

        1

        1
```

```
*/

/*     test-ops2.mc   */
int main()
{
  print(true);
  print(false);
  print(true && true);
  print(true && false);
  print(false && true);
  print(false && false);
  print(true || true);
  print(true || false);
  print(false || true);
  print(false || false);
  print(!false);
  print(!true);
  print(-10);
  print(--42);

  return 0;
}

/*
Output
       1

       0

       1

       0

       0

       0

       1

       1

       1

       0

       1

       0

       -10

       42
```

```
*/

/*      fail-cast1.mc  */
int main()
{
        bool a;
        a = (bool)"a";
        return 0;
}/*
Output
        Fatal error: exception Failure("Covert from string to boolis not permitted")

*/

/*      fail-cast2.mc  */
class foo
{
        int aaa;
}

int main()
{
        int a;
        foo f;
        f = (foo)a;
        return 0;
}/*
Output
        Fatal error: exception Parsing.Parse_error

*/

/*      fail-cast3.mc  */
int main()
{
        if (3 > false)
        {
                print("hello world");
        }
        return 0;
}/*
Output
        Fatal error: exception Failure("illegal binary operator int > bool in 3 > false")

*/

/*      test-cast1.mc  */
int main()
{
        string a;
        float b;
```

```
	string c;
	a = "aaa";
	b = ((int)"1") + 1.7;
	c = "aaa";
	if (b > 2 && (a==c))
	{
		print("variable b is larger than 2 value:" + (string)b);
	}
	else
		print("variable b is not larger than 2 value:" + (string)b);

	return 0;
}/*
Output
	variable b is larger than 2 value:2.700000
*/

/*	test-cast2.mc */
int main()
{
	int a;
	a = ((int)3.5);
	print(a);
	return 0;
}/*
Output
	3
*/

/*	test-cast3.mc */
int main()
{
	int a;
	a = ((int)3.5);
	print((float)a);
	return 0;
}/*
Output
	3.000000
*/

/*	test-cast4.mc */
int main()
{
	int a;
	float b;
	b = (float)"3.5";
	a = (int)b;
	print(a);
	return 0;
}/*
Output
```

```
        3
*/

/*      test-implicit_conversion1.mc */
int main()
{
        float a;
        float b;
        a = 3;
        b = (float)3;
        print(a);
        print(b);
        return 0;
}/*
Output
        3.000000

        3.000000
*/

/*      test-implicit_conversion2.mc */
int main()
{
        int a;
        float b;
        a = 3;
        b = 3.5;
        if (a<b)
                print("a is smaller than b");
        else
                print("a is larger than b");
        return 0;
}/*
Output
        a is smaller than b
*/

/*      test-implicit_conversion3.mc */
int main()
{
        int a;
        float b;
        a = (int)"10";
        b = (float)"10";
        if (a == b)
        {
                print("a equals to b");
        }
        else
                print("a doesn't equal to b");
        return 0;
}/*
```

```
Output
        a equals to b
*/

/*      fail-expr1.mc  */
int a;
bool b;

void foo(int c, bool d)
{
  int dd;
  bool e;
  a + c;
  c - a;
  a * 3;
  c / 2;
  d + a; /* Error: bool + int */
}

int main()
{
  return 0;
}
/*
Output
        Fatal error: exception Failure("illegal binary operator bool + int in d + a")

*/

/*      fail-expr2.mc  */
int a;
bool b;

void foo(int c, bool d)
{
  int d;
  bool e;
  b + a; /* Error: bool + int */
}

int main()
{
  return 0;
}
/*
Output
        Fatal error: exception Failure("illegal binary operator bool + int in b + a")

*/

/*      test-expr1.mc  */
int a;
```

```
bool b;

void foo(int c, bool d)
{
  int dd;
  bool e;
  a + c;
  c - a;
  a * 3;
  c / 2;
}

int main()
{
  return 0;
}
/*
Output
*/

/*     fail-for1.mc   */
int main()
{
  int i;
  for ( ; true ; ) {} /* OK: Forever */

  for (i = 0 ; i < 10 ; i = i + 1) {
    if (i == 3) return 42;
  }

  for (j = 0; i < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}
/*
Output
       Fatal error: exception Analyzer.UndefinedID("j")

*/

/*     fail-for13_varNotInit.mc     */
int main()
{
  /* Uninitialized var in body */
  int i;
  for(i=0;i<2;i++)
    print(k);

  return 0;
}

/*
```

```
Output
       Fatal error: exception Parsing.Parse_error

*/

/*     fail-for2.mc   */
int main()
{
  int i;

  for (i = 0; j < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}
/*
Output
       Fatal error: exception Analyzer.UndefinedID("j")

*/

/*     fail-for3.mc   */
int main()
{
  int i;

  for (i = 0; i ; i = i + 1) {} /* i is an integer, not Boolean */

  return 0;
}
/*
Output
       Fatal error: exception Failure("invaild statement type in For")

*/

/*     fail-for4.mc   */
int main()
{
  int i;

  for (i = 0; i < 10 ; i = j + 1) {} /* j undefined */

  return 0;
}
/*
Output
       Fatal error: exception Analyzer.UndefinedID("j")

*/

/*     fail-for5.mc   */
int main()
```

```
{
  int i;

  for (i = 0; i < 10 ; i = i + 1) {
    foo(); /* Error: no function foo */
  }

  return 0;
}
/*
Output
       Fatal error: exception Analyzer.UndefinedFunction("foo")

*/

/*     fail-for_noSemicolon.mc      */
int main()
{
  for(){ /* nothing in () */
    print(0);
  }

  return 0;
}

/*
Output
       Fatal error: exception Parsing.Parse_error

*/

/*     fail-for_tooFewSemicolon.mc  */
int main()
{
  for(;){ /* only 1 semi-colon in () */
    print(0);
  }

  return 0;
}

/*
Output
       Fatal error: exception Parsing.Parse_error

*/

/*     fail-for_tooManySemicolons.mc*/
int main()
{
  for(;;;){ /* too many semi-colon in () */
    print(0);
```

```
  }

  return 0;
}

/*
Output
       Fatal error: exception Parsing.Parse_error

*/

/*     test-for1.mc   */
int main()
{
  int i;
  for (i = 0 ; i < 5 ; i = i + 1) {
    print(i);
  }
  print(42);
  return 0;
}
/*
Output
       0

       1

       2

       3

       4

       42

*/

/*     test-for2.mc   */
int main()
{
  int i;
  i = 0;
  for ( ; i < 5; ) {
    print(i);
    i = i + 1;
  }
  print(42);
  return 0;
}
/*
Output
       0
```

```
       1

       2

       3

       4

       42

*/

/*     test-for_modifyOuterVar.mc   */
int main()
{
  /* init, access and modify in for body */
  int i;
  int k;
  i = 3;
  for(i=0;i<2;i=i+1){
    k = 10 + i;
    print(k);
  }
  print(k);
  return 0;
}

/*
Output
       10

       11

       11

*/

/*     fail-func1.mc  */
int foo() {}

int bar() {}

int baz() {}

void bar() {} /* Error: duplicate function bar */

int main()
{
  return 0;
}
/*
```

```
Output
       Fatal error: exception Failure("duplicated function declaration bar")

*/

/*     fail-func2.mc  */
int foo(int a, bool b, int c) { }

void bar(int a, bool b, int a) {} /* Error: duplicate formal a in bar */

int main()
{
  return 0;
}
/*
Output
       Fatal error: exception Failure("duplicate formals a")

*/

/*     fail-func3.mc  */
int foo(int a, bool b, int c) { }

void bar(int a, void b, int c) {} /* Error: illegal void formal b */

int main()
{
  return 0;
}
/*
Output
       Fatal error: exception Failure("formal b shouldn't be type void")

*/

/*     fail-func4.mc  */
int foo() {}

void bar() {}

int print() {} /* Should not be able to define print */

void baz() {}

int main()
{
  return 0;
}
/*
Output
       Fatal error: exception Failure("duplicated function declaration print")
```

```
*/

/*     fail-func5.mc  */
int foo() {}

int bar() {
  int a;
  void b; /* Error: illegal void local b */
  bool c;

  return 0;
}

int main()
{
  return 0;
}
/*
Output
       Fatal error: exception Failure("local b shouldn't be type void")

*/

/*     fail-func6.mc  */
void foo(int a, bool b)
{
}

int main()
{
  foo(42, true);
  foo(42); /* Wrong number of arguments */
}
/*
Output
       Fatal error: exception Failure("expecting 2 argument(s) in foo(42)")

*/

/*     fail-func7.mc  */
void foo(int a, bool b)
{
}

int main()
{
  foo(42, true);
  foo(42, true, false); /* Wrong number of arguments */
}
/*
Output
       Fatal error: exception Failure("expecting 2 argument(s) in foo(42, true, false)")
```

```
*/

/*     fail-func8.mc  */
void foo(int a, bool b)
{
}

void bar()
{
}

int main()
{
  foo(42, true);
  foo(42, bar()); /* int and void, not int and bool */
}
/*
Output
       Fatal error: exception Failure("illegal actual argument found void expected bool in
bar()[void]")

*/

/*     fail-func9.mc  */
void foo(int a, bool b)
{
}

int main()
{
  foo(42, true);
  foo(42, 42); /* Fail: int, not bool */
}
/*
Output
       Fatal error: exception Failure("illegal actual argument found int expected bool in
42")

*/

/*     fail-func_assignInArg.mc     */
void func(int i = 5){ /* assignment in argument */
  print(i);
}
int main()
{
  func(3);
  return 0;
}

/*
```

```
Output
        Fatal error: exception Parsing.Parse_error

*/

/*      fail-func_needStringGivenInt.mc      */
void func(string s){
  prints("Got string");
}
int main()
{
  func(1); /* need string but given int */
  return 0;
}

/*
Output
        Fatal error: exception Failure("illegal actual argument found int expected string in
1")

*/

/*      test-add1.mc   */
int add(int x, int y)
{
  return x + y;
}

int main()
{
  print( add(17, 25) );
  return 0;
}
/*
Output
        42

*/

/*      test-func1.mc  */
int add(int a, int b)
{
  return a + b;
}

int main()
{
  int a;
  a = add(39, 3);
  print(a);
  return 0;
}
```

```
/*
Output
        42

*/

/*      test-func2.mc  */
/* Bug noticed by Pin-Chin Huang */

int fun(int x, int y)
{
  return 0;
}

int main()
{
  int i;
  i = 1;

  fun(i = 2, i = i+1);

  print(i);
  return 0;
}

/*
Output
        2

*/

/*      test-func3.mc  */
void printem(int a, int b, int c, int d)
{
  print(a);
  print(b);
  print(c);
  print(d);
}

int main()
{
  printem(42,17,192,8);
  return 0;
}
/*
Output
        42

        17

        192
```

```
       8

*/

/*     test-func4.mc  */
int add(int a, int b)
{
  int c;
  c = a + b;
  return c;
}

int main()
{
  int d;
  d = add(52, 10);
  print(d);
  return 0;
}
/*
Output
       62

*/

/*     test-func5.mc  */
int foo(int a)
{
  return a;
}

int main()
{
  return 0;
}
/*
Output
*/

/*     test-func6.mc  */
void foo() {}

int bar(int a, bool b, int c) { return a + c; }

int main()
{
  print(bar(17, false, 25));
  return 0;
}
/*
Output
```

```
        42

*/

/*      test-func7.mc  */
int a;

void foo(int c)
{
  a = c + 42;
}

int main()
{
  foo(73);
  print(a);
  return 0;
}
/*
Output
        115

*/

/*      test-func8.mc  */
void foo(int a)
{
  print(a + 3);
}

int main()
{
  foo(40);
  return 0;
}
/*
Output
        43

*/

/*      test-func_noArgFunc.mc        */
void noArg(){
  print(0);
}

int main()
{
  /* function with no arg */
  noArg();
  return 0;
}
```

```
/*
Output
        0

*/

/*      test-func_stringArg.mc        */
void printString(string s){
  print(s);
}
int main()
{
  printString("Hi");
  return 0;
}

/*
Output
        Hi

*/

/*      test-func_touchGlobalVar.mc   */
int a;

void printGlobal(){
  print(a);
  a = 4;
  print(a);
}
int main()
{
  /* init, access and modify global var */
  a = 3;
  printGlobal();
  a = 5;
  print(a);
  return 0;
}

/*
Output
        3

        4

        5

*/

/*      fail-global1.mc       */
```

```
int c;
bool b;
void a; /* global variables should not be void */


int main()
{
  return 0;
}
/*
Output
       Fatal error: exception Failure("global a shouldn't be void type")

*/

/*     fail-global2.mc       */
int b;
bool c;
int a;
int b; /* Duplicate global variable */

int main()
{
  return 0;
}
/*
Output
       Fatal error: exception Failure("duplicated global declaration b")

*/

/*     test-global1.mc       */
int a;
int b;

void printa()
{
  print(a);
}

void incab()
{
  a = a + 1;
  b = b + 1;
}

int main()
{
  a = 42;
  b = 21;
  printa();
  incab();
```

```
  printa();
  return 0;
}
/*
Output
        42

        43

*/

/*      test-global2.mc       */
bool i;

int main()
{
  int i; /* Should hide the global i */

  i = 42;
  print(i + i);
  return 0;
}
/*
Output
        84

*/

/*      test-global3.mc       */
int i;
bool b;
int j;

int main()
{
  i = 42;
  j = 10;
  print(i + j);
  return 0;
}
/*
Output
        52

*/

/*      test-var2.mc   */
int a;

void foo(int c)
{
  a = c + 42;
```

```
}

int main()
{
  foo(73);
  print(a);
  return 0;
}
/*
Output
       115

*/

/*     fail-if1.mc    */
int main()
{
  if (true) {}
  if (false) {} else {}
  if (42) {} /* Error: non-bool predicate */
}
/*
Output
       Fatal error: exception Failure("expected Boolean expression in 42")

*/

/*     fail-if11.mc   */
int main()
{
  if(true){
    else{
      print(0); /* dangling else */
    }
  }

  return 0;
}

/*
Output
       Fatal error: exception Parsing.Parse_error

*/

/*     fail-if12.mc   */
int main()
{
  if(true)
    else /* dangling else, no semicolon */
      print(0);
```

```
  return 0;
}

/*
Output
       Fatal error: exception Parsing.Parse_error

*/

/*     fail-if2.mc    */
int main()
{
  if (true) {
    foo; /* Error: undeclared variable */
  }
}
/*
Output
       Fatal error: exception Analyzer.UndefinedID("foo")

*/

/*     fail-if3.mc    */
int main()
{
  if (true) {
    42;
  } else {
    bar; /* Error: undeclared variable */
  }
}
/*
Output
       Fatal error: exception Analyzer.UndefinedID("bar")

*/

/*     test-if1.mc    */
int main()
{
  if (true) print(42);
  print(17);
  return 0;
}
/*
Output
       42

       17

*/
```

```
/*     test-if2.mc    */
int main()
{
  if (true) print(42); else print(8);
  print(17);
  return 0;
}
/*
Output
       42

       17

*/

/*     test-if3.mc    */
int main()
{
  if (false) print(42);
  print(17);
  return 0;
}
/*
Output
       17

*/

/*     test-if4.mc    */
int main()
{
  if (false) print(42); else print(8);
  print(17);
  return 0;
}
/*
Output
       8

       17

*/

/*     test-if5.mc    */
int cond(bool b)
{
  int x;
  if (b)
    x = 42;
  else
    x = 17;
  return x;
```

```
}

int main()
{
 print(cond(true));
 print(cond(false));
 return 0;
}
/*
Output
       42

       17

*/

/*     test-if_nestedIf.mc   */
int main()
{
  /* nested if */
  bool outer;
  bool inner;
  outer = true;
  inner = true;

  if(outer){
    if(inner){
      print(0);
    }else{
      print(1);
    }
  }

  return 0;
}

/*
Output
       0

*/

/*     fail-nomain.mc */
/*
Output
       Fatal error: exception Analyzer.MainNotDefined

*/

/*     test-fib.mc    */
int fib(int x)
{
```

```
  if (x < 2) return 1;
  return fib(x-1) + fib(x-2);
}

int main()
{
  print(fib(0));
  print(fib(1));
  print(fib(2));
  print(fib(3));
  print(fib(4));
  print(fib(5));
  return 0;
}
/*
Output
        1

        1

        2

        3

        5

        8

*/

/*     test-gcd.mc    */
int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}

int main()
{
  print(gcd(2,14));
  print(gcd(3,15));
  print(gcd(99,121));
  return 0;
}
/*
Output
        2

        3
```

```
        11

*/

/*      test-gcd2.mc   */
int gcd(int a, int b) {
  while (a != b)
    if (a > b) a = a - b;
    else b = b - a;
  return a;
}

int main()
{
  print(gcd(14,21));
  print(gcd(8,36));
  print(gcd(99,121));
  return 0;
}
/*
Output
        7

        4

        11

*/

/*      test-local1.mc */
void foo(bool i)
{
  int i; /* Should hide the formal i */

  i = 42;
  print(i);
}

int main()
{
  foo(true);
  return 0;
}
/*
Output
        42

*/

/*      test-local2.mc */
int foo(int a, bool b)
{
```

```
  int c;
  bool d;

  c = a;

  return c + 10;
}

int main() {
 print(foo(37, false));
 return 0;
}
/*
Output
       47

*/

/*     test-local_localShadowArg.mc */
void printLocal(int a){
  int a;
  a = 6;
  print(a);
}

int main(){
  /* inner scope shadows outer scope var */
  int a;
  a = 5;
  printLocal(a);
  print(a);
  return 0;
}
/*
Output
       6

       5

*/

/*     test-var1.mc   */
int main()
{
  int a;
  a = 42;
  print(a);
  return 0;
}
/*
Output
       42
```

```
*/

/*     test-class_func.mc    */
class Test{
    constructor(){}
    int getInt(){
        return 3;
    }
}
int main(){
    class Test t;
    int i;
    t = new Test();
    i = t.getInt();
    print(i);

    return 0;
}/*
Output
       3
*/

/*     test-class_funcWithThis.mc    */
class Line {
    int x;
       constructor(int a){
        x = a;
    }
    int getX() {
      int a;
      a = this.x;
      return a;
    }
}


int main() {
    class Line e;
    int i;
    e = new Line(5);
       i = e.getX();
    print(i);
    return 0;
}
/*
Output
       5
*/

/*     test-class_inClassFor.mc     */
class Test{
```

```
    int value;
    constructor(int v){
        value = v;
    }
    int getValue(){
        int v;
        int i;

        v = this.value;
        for(i=0;i<3;i = i+1){
            v = v*2;
        }

        return v;
    }
}
int main(){
    class Test t1;
    class Test t2;
    int v1;
    int v2;

    t1 = new Test(3);
    t2 = new Test(-1);
    v1 = t1.getValue();
    v2 = t2.getValue();

    print(v1);
    print(v2);

    return 0;
}/*
Output
        24

        -8
*/

/*      test-class_inClassIf.mc       */
class Test{
    int value;
    constructor(int v){
        value = v;
    }
    int getValue(){
        int v;

        v = this.value;
        if(v < 3)
            v = v*2;
        return v;
    }
```

```
}
int main(){
    class Test t1;
    class Test t2;
    int v1;
    int v2;

    t1 = new Test(1);
    t2 = new Test(3);
    v1 = t1.getValue();
    v2 = t2.getValue();

    print(v1);
    print(v2);

    return 0;
}/*
Output
       2

       3
*/

/*     test-class_inClassWhile.mc   */
class Test{
    int value;
    constructor(int v){
        value = v;
    }
    int getValue(){
        int v;

        v = this.value;
        while(v>10){
            v = v/2;
        }

        return v;
    }
}
int main(){
    class Test t1;
    class Test t2;
    int v1;
    int v2;

    t1 = new Test(100);
    t2 = new Test(9);
    v1 = t1.getValue();
    v2 = t2.getValue();

    print(v1);
```

```
    print(v2);

    return 0;
}/*
Output
       6

       9
*/

/*     test-class_nestedAccess.mc   */
class A{
  int x;
  constructor(int the_x){
    x = the_x;
  }
}
class B{
  class A a;
  constructor(class A the_a){
    a = the_a;
  }
}

int main(){
  int c;
  class A a;
  class B b;
  a = new A(3);
  b = new B(a);
  c = b.a.x;
  print(c);
  return 0;
}
/*
Output
       3

*/

/*     test-class_nestedObj.mc      */
class Dot{
  int x;
  int y;
  constructor(int a, int b){
    x = a;
    y = b;
  }
}
class Circle{
  class Dot center;
  constructor(class Dot d){
```

```
    center = d;
  }
}

int main(){
  class Dot d;
  class Circle c;
  d = new Dot(3,5);
  c = new Circle(d);
  print(c.center.x);
  print(c.center.y);
  return 0;
}
/*
Output
        3

        5

*/

/*      test-class_noParamConstructor.mc     */
class Circle{
  int x;
  int y;
  constructor(){}
}
int main(){
  class Circle c;
  c = new Circle();
  return 0;
}
/*
Output
*/

/*      test-class_objArrayField.mc  */
class Test{
    int[] array;
    constructor(){
        array = new int[10];
    }
}

int main(){
    class Test t;
    int[] a;
    t = new Test();
    a = t.array;
    a[3] = 33;
    print(a[0]);
    print(a[3]);
```

```
    print(a[9]);
    return 0;
}/*
Output
        0

        33

        0
*/

/*      test-class_objAsParam.mc      */
class Line {
        constructor(){}
}


int foo(class Line e) {
        return 0;
}

int main() {
    class Line e;
    e = new Line();
        print(foo(e));
    return 0;
}/*
Output
        0
*/

/*      test-class_printObjAccess.mc */
class Circle{
  int x;
  int y;
  constructor(int a, int b){
    x = a;
    y = b;
  }
}
int main(){
  class Circle c;
  c = new Circle(3,5);
  print(c.x);
  return 0;
}
/*
Output
        3

*/
```

```
/*     test-class_recursion.mc      */
class Node{
    int value;
    string s;
    class Node left;
    class Node right;

    constructor(int v, string ss){
        value = v;
        s = ss;
    }
    constructor(class Node l, class Node r, int v, string ss){
        value = v;
        left = l;
        right = r;
        s = ss;
    }

    string getValue(){
        string ss;
        string lVal;
        string rVal;
        int v;
        string sss;

        v = this.value;
        if(v == 0){
            sss = this.s;
            return sss;
        }

        lVal = this.left.getValue();
        rVal = this.right.getValue();
        sss = this.s;
        ss = "" + lVal +  "<-" + sss + "->" + rVal;

        return ss;
    }
}
int main(){
    class Node root;
    class Node left;
    class Node right;
    string s;

    left = new Node((int)(float)"0.0","foo");
    right = new Node(0,"bar");
    root = new Node(left, right, 4, "foobar");
    s = root.getValue();
    print(s);

    return 0;
```

```
}/*
Output
        foo<-foobar->bar
*/

/*      test-class_stringField.mc     */
class Circle{
  string s;
  constructor(string t){
    s = t;
  }
}
int main(){
  class Circle c;
  string a;
  c = new Circle("hello");
  a = c.s;
  print(a);
  return 0;
}
/*
Output
        hello

*/

/*      test-class_stringFunc.mc      */
class Test{
    constructor(){}
    string getString(){
        return "Hello";
    }
}
int main(){
    class Test t;
    string s;
    t = new Test();
    s = t.getString();
    print(s);

    return 0;
}/*
Output
        Hello
*/

/*      test-class_withParamsConstructor.mc  */
class Circle{
  int x;
  int y;
  constructor(){}
  constructor(int a, int b){
```

```
    x=a;
    y=b;
  }
}
int main(){
  class Circle c;
  int the_x;
  c = new Circle(3,5);
  the_x=c.x;
  print(the_x);
  return 0;
}
/*
Output
        3

*/

/*      fail-return1.mc       */
int main()
{
  return true; /* Should return int */
}
/*
Output
        Fatal error: exception Failure("return gives bool expected int in true")

*/

/*      fail-return2.mc       */
void foo()
{
  if (true) return 42; /* Should return void */
  else return;
}

int main()
{
  return 42;
}
/*
Output
        Fatal error: exception Failure("return gives int expected void in 42")

*/

/*      test-return_inWhileIf.mc     */
int main(){
  int i;
  i = 0;
  while(i<3){
    if(i > 2)
```

```
      return 0;
    i = i + 1;
    print(i);
  }
  return 0;
}
/*
Output
       1

       2

       3

*/

/*     test-return_returnInWhile.mc */
int main(){
  /* return in while */
  int i;
  i = 10;
  while(i<11){
    print("Now return");
    return 0;
  }
  print(3);
  return 1;
}
/*
Output
       Now return

*/

/*     fail-stringaddint.mc  */
int main()
{
       string a;
       a = "foo";
       print(a + 3);
}



/*
Output
       Fatal error: exception Failure("illegal binary operator string + int in a + 3")

*/

/*     fail-stringequaltoint.mc     */
int main()
```

```
{
	string a;
	a = "foo";
	if (a == 0)
	{
		print(a + 3);
	}
}



/*
Output
	Fatal error: exception Failure("illegal binary operator string == int in a == 0")

*/

/*	fail-stringneqtoint.mc	*/
int main()
{
	string a;
	a = "foo";
	if (a != 0)
	{
		print(a + 3);
	}
}



/*
Output
	Fatal error: exception Failure("illegal binary operator string != int in a != 0")

*/

/*	test-string1.mc	*/
int main()
{
	string a;
	a = "foobar";
	print(a);
	print("foobar");
	return 0;
}
/*
Output
	foobar

	foobar
*/
```

```
/*      test-string2.mc       */
int main()
{
        string a;
        a = "foobar";
        print(a + "foobar");
        return 0;
}
/*
Output
        foobarfoobar
*/

/*      test-string3.mc       */
int main()
{
        string a;
        string b;
        a = "foo";
        b = "bar";
        print(a + b + "foobar");
        return 0;
}
/*
Output
        foobarfoobar
*/

/*      test-stringequal1.mc  */
int main()
{
        string a;
        string b;
        a = "aaa";
        b = "aaa";
        if (a == b)
        {
                print("equal");
        }
        else
        {
                print("not equal");
        }
        return 0;
}
/*
Output
        equal
*/

/*      test-stringequal2.mc  */
int main()
```

```
{
	string a;
	string b;
	a = "aaa";
	b = "bbb";
	if (a == b)
	{
		print("equal");
	}
	else
	{
		print("not equal");
	}
	return 0;
}
/*
Output
	not equal
*/

/*	test-stringequal3.mc  */
int main()
{
	string a;
	string b;
	a = "aaa";
	b = "bbb";
	if (a != b)
	{
		print("not equal");
	}
	else
	{
		print("equal");
	}
	return 0;
}
/*
Output
	not equal
*/

/*	test-tostring1.mc	*/
int main()
{
	int a;
	a = 3;
	print(toString(a));
	return 0;
}
/*
Output
```

```
        3
*/

/*      test-tostring2.mc     */
int main()
{
        float a;
        a = 3.5;
        print("output" + toString(a));
        return 0;
}
/*
Output
        output3.500000
*/

/*      test-tostring3.mc     */
int main()
{
        bool a;
        a = false;
        print(toString(a));
        return 0;
}
        /*
Output
        0
*/

/*      test-tostring4.mc     */
int main()
{
        string a;
        a = "foo ";
        print(toString(a)+toString(3)+" "+ toString(3.5)+ "bar" );
        return 0;
}
/*
Output
        foo 3 3.500000bar
*/

/*      fail-while1.mc */
int main()
{
  int i;

  while (true) {
    i = i + 1;
  }

  while (42) { /* Should be boolean */
```

```
    i = i + 1;
  }

}
/*
Output
       Fatal error: exception Failure("invaild statement type in While")

*/

/*     fail-while11.mc       */
int main()
{
  while() /* Nothing in () */
    ;
  return 0;
}

/*
Output
       Fatal error: exception Parsing.Parse_error

*/

/*     fail-while2.mc */
int main()
{
  int i;

  while (true) {
    i = i + 1;
  }

  while (true) {
    foo(); /* foo undefined */
  }

}
/*
Output
       Fatal error: exception Analyzer.UndefinedFunction("foo")

*/

/*     test-while1.mc */
int main()
{
  int i;
  i = 5;
  while (i > 0) {
    print(i);
    i = i - 1;
```

```
  }
  print(42);
  return 0;
}
/*
Output
        5

        4

        3

        2

        1

        42

*/

/*      test-while2.mc */
int foo(int a)
{
  int j;
  j = 0;
  while (a > 0) {
    j = j + 2;
    a = a - 1;
  }
  return j;
}

int main()
{
  print(foo(7));
  return 0;
}
/*
Output
        14

*/

Positive tests: 95 passed
Negative tests: 52 passed
```

8.3 Demo Code

Tree.lava (Liu)

```
class Node{
    int value;
    string label;
    class Node left;
    class Node right;

    constructor(int v, string n){
        value = v;
        label = n;
    }
    constructor(class Node l, class Node r, int v, string n){
        value = v;
        left = l;
        right = r;
        label = n;
    }

    string toString(){
        string lVal;
        string rVal;
        int v;
        string s;

        s = this.label;
        v = this.value;
        /* A node is closed in square brackets */
        s = "[" + s + ":" + ((string)v) + "]";

        /* Check whether the node is leaf */
        if(v == 0){
            return s;
        }

        lVal = this.left.toString();
        rVal = this.right.toString();
        s = "(" + lVal +  ")<-" + s + "->(" + rVal + ")";

        return s;
    }
}
int main(){
    class Node l0_root;
    class Node l1_left;
    class Node l1_right;
    class Node l2_n1;
    class Node l2_n2;
    class Node l2_n3;
    class Node l2_n4;
    string s;
```

```
    string[] sa;
    sa = new string[10];

    sa[2] = "Hello world";
    print(sa[2]);

    l2_n1 = new Node(0, "C1");
    l2_n2 = new Node(0, "C2");
    l2_n3 = new Node(0, "C3");
    l2_n4 = new Node(0, "C4");

    l1_left = new Node(l2_n1, l2_n2, (int)(float)"3.0","B1");
    l1_right = new Node(l2_n3, l2_n4, (int)"4","B2");
    l0_root = new Node(l1_left, l1_right, 10, "A");
    s = l0_root.toString();
    print(s);

    return 0;
}
```

Linkedlist.lava (Liu)

```
class Node{
    int value;
    class Node last;

    /* Construct a single node or the root */
    constructor(int v){
        value = v;
    }

    /* Construct a node and connect it to the last node */
    constructor(int v,class Node n){
        value = v;
        last = n;
    }

    /* Calculate the sum of all nodes in the list */
    int getSum(){
        int v;

        v = this.value;
        if(v == 0){
            return 0;
        }

        v = v + this.last.getSum();
        return v;
    }
```

```
    string toString(){
        int v;
        string s;

        /* string also needs initialization */
        s = "";
        v = this.value;
        if(v==0)
            return "r"+s;

        s = this.last.toString() + "->" + (string)v;
        return s;
    }
}
int main(){
    class Node root;
    class Node n1;
    class Node n2;
    class Node n3;
    class Node n4;
    class Node n5;
    int v;
    string s;

    root = new Node(0);

    /* Cast string to int */
    n1 = new Node((int)"1",root);

    /* Cast float to int */
    n2 = new Node((int)3.0, n1);

    n3 = new Node(5, n2);
    n4 = new Node(7, n3);
    n5 = new Node(9, n4);
    v = n5.getSum();
    print("Value of the list is "+(string)v);

    s = n5.toString();
    print(s);

    return 0;
}
```