DECAF

Final Report

May 10, 2017

Hidy Han, JiaYan Hu, Kimberly Tao, Kylie Wu

Table of Contents

1. Introduction

Overview

DECAF is a general-purpose, object-oriented language that compiles to LLVM and can be used in a variety of applications. It is a language that will be intuitive to use for programmers who have previously used other high-level programming languages, as the syntax and semantic structure of DECAF are rather similar to that of Java and C. DECAF extracts a core subset of features from Java and C and presents these features in a concise semantic model.

More specifically, DECAF supports object-oriented functionalities, such as inheritance, alongside the expected functions from high-level programming languages, such as arrays, static functions, and function calling.

Background

Object-oriented programming is a type of programming that centers around reusability. This is accomplished by organizing the code around the concept of an object: something with state and behavior. Before object-oriented programming came about, programs were simply viewed as logical procedures that took input, manipulated it, and then produced output. Object-oriented programming focuses on using objects that interact together to ultimately generate the output.

DECAF is implemented using OCaml, and its functionality is inspired by Java, C, and Python. The compiler accepts valid DECAF files and transforms them into LLVM code.

Goals

Safety - DECAF is a statically typed language, without any built-in automatic casting. This means that programmers must explicitly cast values by themselves when they use different types together.

Familiarity - DECAF's syntax is designed to resemble C and Java. New programmers will also be able to learn DECAF fairly quickly and apply their knowledge to other languages.

2. Language Tutorial

Configuration

DECAF requires the OCaml LLVM library, which can be installed using opam. This can be accomplished by doing the following:

```
sudo apt-get install -y ocaml m4 llvm opam
opam init
opam install llvm.3.6 ocamlfind
eval `opam config env`
```

Afterwards, running make in the root directory of DECAF will generate all the parts needed for DECAF to work. Running `./testall.sh` will make the compiler go through all the included test cases, printing out OK or FAIL for each test.

Using the Compiler

All DECAF programs are required to have a main() function. An example of a simple program is hello_world.dcf:

```
main() -> void {
    print_string("Hello World!\n");
}
```

The file ending for DECAF programs if *.dcf
To compile and run this program, type

```
./decaf.native -l < hello_world.dcf | tail -n+3 > hello_world.ll
llc hello_world.ll
gcc hello_world.s
./a.out
```

The output will subsequently be: `Hello World!`

3. Language Reference Manual

Data Types

i. Primitives

DECAF supports a variety of different basic primitive types, allowing its programs to be versatile.

Type	Description	Use Cases
`bool`	Boolean type; can either be true or false	Used to indicate a conditional statement that can be true or false.
`int`	Integer with 32-bit (4 byte) capacity	Used to store base ten integers ranging from -2^{31} to $2^{31} - 1$.
`float`	Floating point with 64-bit (8 byte) capacity	Can store decimals or integers greater than the range int can handle.
`char`	Represents a single ASCII character; size is 1 byte	Can be used to represent a character or text as a sequence of characters.

ii. Literals

Literals are syntactic representations of fixed values for primitive types. They are the actual representation of values in the program, as opposed to being hidden within an identifier.

Type	Syntax	Example
`bool`	Only two possible values: true, false	`true, false`
`int`	A sequence of digits, may be preceded by - to indicate negativity	`4115, -325409`
`float`	Typed as decimal fractions, may be preceded by - to denote negativity	`0.25, -534.2908`
`char`	A single character enclosed by single quotes (‘’)	`‘a’, ‘K’`

	Refer to the following table for escape sequences	
`string`	Sequence of characters enclosed by double quotes (“”)	`“Hello World”`
`array`	Mutable fixed-length sequence of expressions separated by commas enclosed by brackets ([])	`[int] arr = [int, 5]`

Escape Sequences:

Syntax	Meaning
`‘\\’`	backslash
`‘\’’`	single quote
`‘\”’`	double quote
`‘\n’`	newline character
`‘\t’`	tab character
`‘\r’`	carriage return

Example:

```
string str = “William says:\t\”Hello!\”\n”
print_string(str);

Expected Output:
William says:    “Hello!”
```

iii. Built-ins: Strings

Strings are immutable sequences of characters.

`string`	Represents a sequence of characters; size is variable	Used to represent text.

iv. Built-ins: Arrays

Arrays are mutable fixed-length sequences, whose contents can be changed after their initial instantiation. The items of an array are arbitrary objects of the same type (see object hierarchy) or primitives of the same type. Arrays are created by stating the type and length in square brackets as shown below.

Array Operations

Operation	Effect
[ ]	Index into the array.

Example:

```
[int] nums = [int,5];

nums[0] = -5;
nums[1] = nums[0];
print_string("nums[0] is now ");
print_int(nums[0]);
```

```
//int array of length 5

//nums[0] becomes -5
//nums[1] becomes -5
//Output: nums[0] is now -5
```

v. Objects

DECAF is an object-oriented programming language, so it supports the implementation of objects, which are entities that have state and behavior. While DECAF has the capability of being object-oriented, it does not have many built-in classes. Thus, it is left to the programmer's discretion to use this feature as they please.

The state and behavior of object classes are defined by listing out characteristic fields and methods in a class declaration. Object classes can extend other classes in a simple manner, by inheriting all of the fields and methods of the parent class. As a result, objects can be created in an inheritance hierarchy, such that in the case of B extends A and C extends B, this means that B contains all fields and methods of A, and C inherits all the fields and methods of B, which includes those from A.

Fields

Fields can consist of primitive data types or other object classes, and methods can return primitives or objects. Fields can be accessed and modified using dot (`.`) operator. A field can be accessed by the following format: `obj.field`

Methods

Object methods can be defined by the user within the class. They typically perform an operation on the instance it is invoked upon and can modify the state of the object. A method can be called by using the dot (`.`) operator in the following format: `obj.method()`. In DECAF, the programmer is given the ability to write whatever methods they may require, utilizing the various built-in data types and operators.

Methods are operations performed on an object that can accept input as parameters and produce output. These inputs and outputs can be primitives, objects, or built-ins. The method signature determines the formal parameters, including the number of arguments, their type, and their order. A calling method must match this signature in order to make a valid call to the method.

Return Types

In the method signature, the return type of a method is specified by the type following the `->` symbol. Returning from a method is indicated by the `return` keyword followed by the primitive or object to return. The type that is returned must match that of the method signature. If a method does not return anything, it returns a `void` type and the return can be omitted (although it can still be used to prematurely exit a method).

The format of a method is as follows:

```
method_name(param1_type param1_name, ... , paramn_type paramn_name) ->
return_type {
        ...
        return some_value;
}
```

Examples:

```
// Method that returns int
gcd(int x, int y) -> int {
    if (y === 0) {
        return x;
    }
    return gcd(y, x % y);
}
```

```
// Method that returns nothing
foo(string print_me) -> void {
    print_string(print_me);
}
```

```
class Square {
    int length;

    Square(int length) -> Square {
        self.length = length;
    }
}

// Method that returns Square object if x is true, returns dummy
otherwise
bar(bool x) -> Square {
    if (x) {
        return Square(5);
    }
    return Square(1);
}
```

Instantiation

Objects are instantiated using the object's constructor method. The name of a constructor has to match the class name. Constructors require no return values but implicitly return the object just instantiated to the calling method.

Objects can be assigned to variables once they have been instantiated.

Classes

A class is the template for an object which can be instantiated. A class is capable of extending the functionality of another class, or being extended to subclasses. Classes have state, represented by fields, and behavior, represented by the methods that can be invoked.

Object Hierarchy and Polymorphism

Objects have the concept of subclasses that extend superclasses. The core purpose of the object-oriented model is that subclasses will inherit the fields and methods of superclasses. A subclass will inherit all of the fields and methods of its superclass. Furthermore, it can override methods in superclasses to provide its own implementation as well as add new fields and methods. DECAF supports single inheritance rather than multiple inheritance, so a subclass can extend at most a single superclass.

Object Keywords

Keyword	Description
`class`	Beginning of a class definition.
`extends`	Indicates that a class is the subclass of another class.
`self`	Similar to this in Java, which refers to the object in the current context.

Object Operations

Operator	Description
`Object(args)`	Instantiates an object.
`o.f`	Accesses a field f in the object o.
`o.m()`	Invokes method m on the object o.

The class definition follows the following format:

```
class class_name [extends super_class]{
    ...
}
```

Examples:

```
// Basic class declaration
class Animal {
    ...
}
```

```
// Example of extending a class
class Mammal extends Animal {
    Mammal(string species) {
        ...    // Add in other fields/methods specific to Mammal
    }
    ...
}
```

```
// Example of implementing an interface
class TwoDShape {
    int num_sides;
    ...
}
```

```
class Rectangle extends TwoDShape {
    int length;
    int width;

    Rectangle(int length, int width) {
        self.length = length;    // self refers to its own fields
        self.width = width;
    }

    ...

    perimeter() -> int {
        return self.length * self.width;
    }
}
```

vi. Casting

Casting must be explicitly used by using the <type> operator. Automatic promotion is not supported. For instance, if a user want to perform arithmetic operations on a float and an integer, they must explicitly choose to cast one of the two operands so that they have matching types.

The following casts are supported:

- `int -> float`
- `int -> bool (0 is false, nonzero is true)`
- `int -> char (ASCII character equivalent)`

- `float -> int (fractional portion will be truncated)`
- `float -> bool (0.0 is false, else true)`
- `float -> char (rounded to nearest int and converted to ASCII character equivalent)`
- `bool -> int` (`true` becomes 1, `false` becomes 0)
- `bool -> float` (`true` becomes 1.0, `false becomes 0.0`)
- `bool -> char (true becomes character with ASCII code 1, false becomes character with ASCII code 0)`
- `char -> bool (if ASCII character number is 0, then false, else true)`
- `char -> int (equivalent ASCII character number)`
- `char -> float (equivalent ASCII character number)`

In addition, all types can be trivially cast to their own type (e.g. bool -> bool, int -> int, float -> float, char -> char), although it will essentially act as a no-op.

`int x = 3;`
`float y = 1.0;`
`int z =  x + y; //error`
	`int x = 3;`
	`float y = 1.0;`
	`int z =  x + <int>y; //ok`

Lexical Conventions

i. Identifiers

Identifiers are sequences of characters used for naming variables, new objects and methods. Identifiers can contain ASCII characters, digits and underscore '_'. The first character can only be a character or underscore. An identifier cannot have collisions with other keywords or names of built-in methods.

By convention, '_', rather than camelCase, is used for the readability of long identifiers.

ii. Comments

Comments in DECAF are written using Java-like syntax, and do not affect compilation. Comments cannot be nested.

Syntax	Description
`//  comment`	Used for single-line comments.

`/*`
`  comment`
`  comment`
`*/`
	Used for multiline comments.

iii. Keywords

The following keywords are reserved in DECAF and may not be used otherwise:

`main`	`int`	`float`	`string`
`bool`	`class`	`extends`	`self`
`and`	`or`	`not`	`return`
`for`	`while`	`if`	`elseif`
`else`	`continue`	`break`	

iv. Operators

DECAF can perform arithmetic, logical, and boolean operations. These different types of operators can be used in combination to create powerful methods and logic.

Arithmetic Operators

Operator	Description
`+`	Sums the operands together.
`-`	Subtracts the right operand from the left operand.
`*`	Multiplies the operands together.
`/`	Divides the left operand by the right operand.
`%`	Performs modulus division of the left operand using the

	right operand.

Boolean Operators

Operator	Description
`and`	Returns true when both operands evaluate to true; returns false otherwise.
`or`	Returns true if either or both of the operands evaluate to true; return false otherwise.
`not`	Negates the boolean value of the operand.

Comparison Operators

Operator	Description
`===`	Compares two operands' values; returns true when they are equal in terms of value, false otherwise.
`!==`	Compares two operands' values; returns true when they are not equal in value, false otherwise.
`<`	Returns true if the left operand is less than the right operand, false otherwise.
`>`	Returns true if the left operand is greater than the right operand, false otherwise.
`<=`	Returns true if the left operand is less than or equal to the right operand, false otherwise.
`>=`	Returns true if the left operand is greater than or equal to the right operand, false otherwise.

Example:

```
int a = 100;                                  // sample declarations
int z = 2;
float f = 50.3892;
bool valid = true;
bool not_valid = false;
string str1 = “sample program”;
string str2 = “this is a”;

int b = 5 + a * z;
if (<float> b > f) {                          // b has the value of 205
    print_string(“b is big\n”);               // evaluates to true
}                                             // prints “b is big\n”
if (valid and not_valid) {
    print_string(“both true\n”);              // and evaluates to false
}
print_string(str2);
print_string(“ “);                            /* prints “this is a sample
print_string(str1);                           program” */
```

v. Precedence and Associativity Rules

A majority of the operators in DECAF are left-associative. This means that consecutive operations with the same precedence will be evaluated from left-to-right. Operators at the top of the table have higher precedence than operators at the lower levels of the table.

Operator	Description	Associativity
. [] f(args…)	Access object member Access array element Invoke a method	Left
+ - not	Unary plus Unary minus Logical NOT	Right
<>	Casting	Right
* / %	Multiplicative	Left
+ -	Additive	Left
< <=	Relational type	Left

> >=		
=== !==	Value equality Value not equals	Left
and	Conditional AND	Left
or	Condition OR	Left
=	Assignment	Right

vi. Brackets

Brackets are symbols that occur in pairs. They are used to separate blocks of code from each other and to alter default precedence in expressions. The scope of a variable is determined by the outermost brackets in which it is enclosed.

Bracket Type	Usage	Example

{ }	Used to denote control flow, methods, and classes.	`main() -> int {`
		`  print("Hello world!\n");`
		`}`

( )	Used in a variety of situations; such as conditionals, grouping, for loops, tuple instantiation.	`if (x < 5) {`
		`  ...`
		`}`

[ ]	Array access operator (see Arrays).	`a = numbers[0];`
< >	Denotes type that operand should be cast to. The type is put between the angle brackets.	`num = <int>16.0`

vii. White Space

Aside from its function to separate tokens, white space (including ASCII SP, HT, FF, and new line characters) is ignored. Developers can use them freely for code readability.

Statements

i. Declarations

Each identifier must be associated with a type. A declaration is a statement which introduces an identifier into the program and specifies its type. If a variable is not initialized at its declaration time, it will be given a default value. The default value is 0 for a number, false for a boolean, and empty strings.

```
string str = "hello world";
```

ii. Expressions

Adding a semicolon to the end of an expression makes it a statement. When executed, the expression is evaluated.

iii. Control Flow

main

The `main` method is the start point for execution of the program. Main does not need to be defined within a class and can exist within the global scope (the same hold true for other global functions). It returns whatever type it is written for. However, it is convention to have main return either a `void` or an `int` representing the return code of the program. If using an `int`, the return value of 0 indicates successful termination whereas a nonzero return value indicates failure.
If a file contains multiple main methods, the compiler will alert programmers to this issue.

```
// Main method
main() -> int {
    ...
    return 0;
}
```

if/elseif/else

The most basic element of control flow structures is the `if-elseif-else` statement. In DECAF, at bare minimum an `if` statement is necessary. `else` is used to indicate what should be done if the conditional is not met. `elseif` denotes

other possible conditions that the scenario may fall into and executes the respective code.
The conditions for decision-making must be boolean values, i.e. `true` or `false`.

The format of an `if-elseif-else` statement is as follows:

```
if (condition) {
    ...
}
elseif (condition) {
    ...
}
else {
    ...
}
```

Example:

```
if (x < 150) {
    print_string("short");
}
elseif (x < 180) {
    print_string("medium");
}
else {
    print_string("tall");
}
```

for loop

The for loop is the most basic loop that enables code to run iteratively. It requires the initialization of some control variable, a range of values at which the control variable should allow the loop to execute, and an expression indicating how the control variable's value should change in between iterations. Failure to provide an expression that alters the control variable's value will result in an infinite loop.

The format for a `for` loop is as follows:

```
for (initialization; terminating condition; iteration) {
    ...
}
```

Example:

```
int n = 0;
int i = 0
for (i = 0; i < 10; i = i + 1) {
    n = n + i;
}
```

while loop

The while loop is another type of loop supporting iteration based on a conditional statement. It is controlled by a conditional statement that is dependent on the control variable's value. The control variable's value must be altered in the body of the loop in order to avoid getting stuck in an infinite loop.

The format of a while loop is as follows:

```
while (condition) {
    ...
}
```

Example:

```
int x = 5;
int y = 1;
while (x > 1) {
    y = y * x;
    x = x - 1;
}
```

break

break is used to indicate when a loop should stop and exit early. It is used within a "for" or "while" loop. If there are nested looping statements, it terminates the execution of the loop whose scope it is immediately nested within.

continue

continue is used to skip the remaining statements not yet executed in the current iteration of a for or while loop, and begins the next iteration of the loop. If there are nested looping statements, it affects the loop whose scope it is immediately nested within.

Example:

`for (i = 0; i < 6; i = i + 1) {`
`  if (i === 3) {`
`  continue;`
`  }`
`  elseif (i === 5) {`
`  print_string("break at 5\n");`
`  break;`
`  }`
`  print_string("i is ");`
`  print_int(i);`
`  print_string("\n");`
`}`
	Expected output:
	`i is 0`
	`i is 1`
	`i is 2`
	`i is 4`
	`break at 5`

return

return exits the method and returns the primitive, built-in, or object specified following it to the calling method. If a method does not return anything (in other words, it returns `void`), the return statement can be omitted. In all other cases, it is required.

4. Project Plan

Planning:

The team met weekly on Thursday evenings with our Teaching Assistant, Jacob Graff. During these meetings, we discussed what we had accomplished in the past week, and our plans for the next week. These worked as weekly sprint meetings, that allowed us the gauge how well we were doing in terms of time.

In addition to these weekly meetings with Jacob, the team met on Saturday afternoons during the beginning of the semester to discuss the idea behind the language and what features we wanted to include. After the Language Reference Manual was submitted, these weekly Saturday meetings turned into real-time discussions on our team's Slack channel to delegate different tasks to each team member.

Specification:

All of the features and functionalities detailed in the first version of the Language Reference Manual were what the team wanted the language to be able to do in an

idealized world where we would accomplish all that we intended to. DECAF is heavily based off of Java and C, so we knew we wanted to prioritize “C-like” basic functionality first before tackling Object-Oriented features. As time progressed, it became apparent that smaller, more niche features like exception handling and creating a standard library would probably not happen, so such features were deprioritized as the team focused on making the core features robust.

Development:

We started working on software development in the middle of March during Spring Break. We started by tackling the scanner and parser as thoroughly as possible first.

Afterward, for every incremental feature that we had on our list, the semantic checker for that feature would be written, and then the respective code generation block. Any push for a new feature also came with test cases that were added onto the test suite for DECAF. This way, we were able to ensure that each feature worked incrementally throughout the semester.

Testing:

The tests folder is filled with various test cases that have been added at various stages in the development process. Once the semantic checker and code generation were made for a specific feature (take if statements as an example), a number of test cases were made to ensure that the feature worked as expected.

Programing Style Guide:

The following style conventions were followed throughout the project:

- 1 level of indentation is 1 tab. Tab width is 4 characters long.
- All lines of code are under 80 characters long.
- Variables and functions are named similarly to those in C, with underscores as separators.
- Variable names and functions are named meaningfully for the understanding of future readers.
- Pattern matches are at the same level of indentation for the sake of readability.
- Comments should be used where necessary for documentation.

Project Timeline:

Date	Milestone
2/8/17	DECAF Language Proposal

2/22/17	DECAF Language Reference Manual
3/16/17	Basic AST Generation Complete
3/23/17	Basic Code Generation Complete
3/27/17	DECAF “Hello World” Demo
4/13/17	“C-like” Functionality Complete
4/30/17	OOP Functionality Complete
5/06/17	Array Functionality Complete
5/08/17	Updated LRM
5/10/17	Final Demonstration and Project Submission

Roles & Responsibilities:

Although each member held a different position, the responsibilities were divided rather evenly. However, based on the position held, one person may have had more of a say in some decisions than in others.

Team Member	Position	Responsibility
Hidy Han	Language Guru	LRM, Semantic Checker, SAST, Code Generation, Test Suite, Demo, Final Report
JiaYan Hu	Systems Architect	LRM, Semantic Checker, SAST, Code Generation, Test Suite, Final Report
Kim Tao	Tester	LRM, Scanner, Parser, Semantic Checker, AST, SAST, Code Generation, Test Suite, Demo, Final Report
Kylie Wu	Manager	LRM, Scanner, Parser, Semantic Checker, Code Generation, Test Suite, Final Report

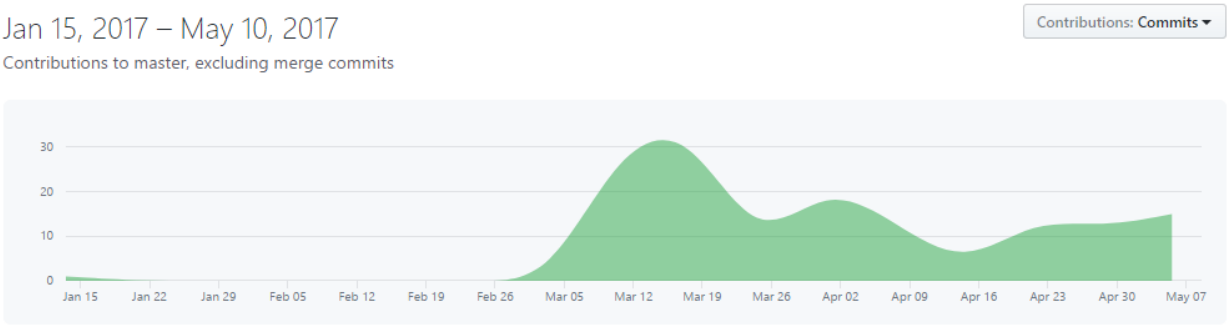
Software Development Environment:

Language: Ocaml 4.02.3
Operating System: Ubuntu 16.04 LTS
Text Editing: Vim
Version Control: Git, via GitHub

Typesetting: Google Documents

Project Log:

Below is a timeline of commits over the course of the semester.




```
commit 490f38bb5a5ad65f8871c30cfaf62b71ff318d4a
Merge: e6bc651 83456f4
Author: HidyHan <yh2635@columbia.edu>
Date:   Wed May 10 19:50:01 2017 -0400

    merge conflict resolved

commit e6bc6516804a2ee9c842606d9d7f456b283d3f48
Author: HidyHan <yh2635@columbia.edu>
Date:   Wed May 10 19:47:58 2017 -0400

    cleanup

commit 83456f46ace02a716ef4f40a4f580ef815bf4fe9
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Wed May 10 17:00:33 2017 -0400

    cleaning unneeded comments and debug print statements; fixed
spacing and lines

commit 868a611144f3ab832c652c82979bc3cb712bc736
Author: HidyHan <yh2635@columbia.edu>
Date:   Wed May 10 12:00:40 2017 -0400

    tree tests

commit 7abc32426e144a38c638dd614ecd1c636c2e5468
```

```
Merge: 5dd5e0b d321956
Author: HidyHan <yh2635@columbia.edu>
Date:   Wed May 10 11:57:11 2017 -0400

    Merge branch 'master' of https://github.com/JY-H/DECAF

commit 5dd5e0b5b64dba71b217b98f59915b8c68cc773d
Author: HidyHan <yh2635@columbia.edu>
Date:   Wed May 10 11:57:02 2017 -0400

    global class map

commit d321956b6de83441299889559e21e93b09b18ba8
Merge: c398de3 c115c2b
Author: Kim Tao <Kimchelly@users.noreply.github.com>
Date:   Wed May 10 11:08:34 2017 -0400

    Merge pull request #21 from JY-H/fire

    added mergesort integration test

commit c115c2b06a7783a679f57838f55afc4644b6f04d
Author: Kim Tao <kmtao88@gmail.com>
Date:   Wed May 10 11:07:26 2017 -0400

    added mergesort integration test

commit c398de334a1c1ef42c0accc9e34615652496d2e3
Merge: 71578d1 2be6fc2
Author: Kim Tao <Kimchelly@users.noreply.github.com>
Date:   Wed May 10 09:50:46 2017 -0400

    Merge pull request #20 from JY-H/fire

    Fire

commit 2be6fc238cd9c11f396425e7787e5cde6c5c2e02
Author: Kim Tao <kmtao88@gmail.com>
Date:   Wed May 10 09:27:10 2017 -0400

    fixed all test cases, added integration arraylist
```

```
commit 204ed3c3eddb40389776eaa6a4fb29fd947b722f
Author: HidyHan <yh2635@columbia.edu>
Date:   Tue May 9 15:31:05 2017 -0400

    cleanup

commit 7f80ee63f73152b200c85cda75a8b8243e71ccc4
Merge: 9217e32 9476f4b
Author: HidyHan <yh2635@columbia.edu>
Date:   Tue May 9 15:05:49 2017 -0400

    Merge branch 'fire' of https://github.com/JY-H/DECAF into fire

commit 9217e3219e5d4855de68c7c3c3a02c5189a64db4
Author: HidyHan <yh2635@columbia.edu>
Date:   Tue May 9 15:05:42 2017 -0400

    sizeof

commit 9476f4b5ed0d3649e8d4212e7968104e70828f90
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Tue May 9 14:53:51 2017 -0400

    bank_acct integration test

commit 3adc99463fefe692049c063d2007a8f2e7d4b3b7
Author: Kim Tao <kmtao88@gmail.com>
Date:   Tue May 9 13:20:56 2017 -0400

    revamped test cases; added forgotten print_char; updated casting
behavior for primitives

commit aac8b608dae216ac5f7ad95bb056e4f165d72f27
Author: HidyHan <yh2635@columbia.edu>
Date:   Tue May 9 12:53:09 2017 -0400

    array access works for 1d; object for array works

commit 6acc9dc682c6ccf24190fb756b331bff334360f1
Author: Kim Tao <kmtao88@gmail.com>
Date:   Mon May 8 00:41:17 2017 -0400
```

```
    semant checks fixed for recursive array types

commit a7c38d51c3f098553ad8e2bf2931b8f5995b6a3e
Author: Kim Tao <kmtao88@gmail.com>
Date:   Mon May 8 00:09:39 2017 -0400

    renaming lst -> array

commit ae03b0a05e2fbdf96b5a094c68215499f640ec12
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Mon May 8 00:02:47 2017 -0400

    new semantic checking for lst is complete

commit b60d0d2599100e82e50b75881378995af1b79807
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sun May 7 23:54:37 2017 -0400

    simplified list access

commit f603e533ee1bc191ae2c3c109a711e638cea9df3
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sun May 7 23:15:38 2017 -0400

    fixed dis cuz y da hull not

commit 33fe161ac118cfcc112105957ee245b72d309d44
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sun May 7 23:08:25 2017 -0400

    updated ast + parser

commit 79574dc6fa81d30a79b67bb98f155c1ccabb88f1
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Sun May 7 22:45:37 2017 -0400

    commenting out list functions; all list-related tests will fail

commit 71578d1f16f32475e028ca0f5b733c2ac5e98da5
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sat May 6 14:10:39 2017 -0400
```

```
    added list access and assign functionality as well as basic
semantic check (needs to be fixed for empty list handling)

commit ad17eec90d30dccf78090d2d1dfb37eb7a8c0368
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Sat May 6 13:44:12 2017 -0400

    warnings all cleaned out except for two unused match cases

commit de792600646b8e4a04e6cf6deaacc0fbf4929209
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Sat May 6 13:03:35 2017 -0400

    cleaned up some warnings, fixed merge conflicts

commit 48c2a88bea1b75dd35e382d704d1c71ea48fd8b5
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sat May 6 12:46:28 2017 -0400

    list access has semantic checks

commit 9f3cc7510866d12a7305fdbc8add0a8e7dc17009
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sat May 6 12:17:04 2017 -0400

    added forgotten if expr check for boolean; started SeqAccess

commit 762ee40c73f3b4474221dafcd4a34b1ff78746ed
Author: HidyHan <yh2635@columbia.edu>
Date:   Fri May 5 17:38:01 2017 -0400

    list creation tentatively works; void* issue unresolved

commit c774835c45ba57f1942eb9b65769fa904be09d45
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Thu May 4 13:19:11 2017 -0400

    beginning list codegen; type mismatch happenign in *.ll files

commit 4a69405d7074fce08e06e5a0cf5ac8e0e7e8d60f
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Tue May 2 23:28:10 2017 -0400
```

```
    enable prints for various types; updated test cases accordingly

commit c20c20be8ffa2694a2484b539631f291d1847d0d
Author: Kim Tao <kmtao88@gmail.com>
Date:   Tue May 2 18:58:48 2017 -0400

    added typed list semantic checking

commit 869b9454827620c82b26a9d406b804a960adf8ef
Merge: ee3d479 6ff2dc1
Author: HidyHan <yh2635@columbia.edu>
Date:   Tue May 2 15:44:26 2017 -0400

    Merge pull request #19 from JY-H/OOP

    Oop

commit 6ff2dc10baf0f4619e233e003d47fa81db7e284c
Merge: ee3d479 13c8540
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Tue May 2 15:40:32 2017 -0400

    removed debug prints, OOP is complete for now

commit 13c85408775741981d223fda705d6c57ec53302a
Author: HidyHan <yh2635@columbia.edu>
Date:   Tue May 2 15:14:59 2017 -0400

    function inheritance supported in a memory-inefficient way; never
have I spent so much time writing so little doing idk how much; ocaml
needs more automatically linked libraries

commit 6609c317c88919ab0390c7b79c3edc9333515f2b
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sun Apr 30 22:31:48 2017 -0400

    Connie is not cute; object field inheritance should work for
multilevel hierarchy

commit aab659e372472a12f41d44eb1c70fddec8870990
Author: Kylie Wu <kcw2141@columbia.edu>
```

```
Date:   Sun Apr 30 20:58:29 2017 -0400

    fixed ordering issue for classes, explicit return statements, still
need to fix actual inheritance of parent fields and methods

commit c12b15280ff75dff2e2fb43392acd70c3381a1d6
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Sat Apr 29 23:59:02 2017 -0400

    reworking code such that semant maps all classes that are not child
classes first, before mapping the child/dependent classes ... having
type mismatch at line 223

commit 17cd313b82d926a15926fb7104f88db112775380
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Sat Apr 29 23:58:06 2017 -0400

    new test case for super classes; fails due to the way order
processes/maps the classes

commit 15046e5c1662e40f3c3154059da7352dc66eba78
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Sat Apr 29 17:25:08 2017 -0400

    classes inherit superclass's fields and methods

commit ee3d4794dabe2ee14d1b60b8ee0ad55fa4058e20
Author: HidyHan <yh2635@columbia.edu>
Date:   Sat Apr 29 16:05:25 2017 -0400

    Update README.md

commit 399912211b1bc724684a9ff2677d3b70ce5a0e99
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Sat Apr 29 11:25:26 2017 -0400

    added test case for object field access and function calls, edited
codegen to fix binop comparison bug

commit f5c11b5b725e4a09d7581ee73b301a949d3580bf
Author: HidyHan <yh2635@columbia.edu>
Date:   Fri Apr 28 18:25:29 2017 -0400
```

```
    updated test case

commit 05bff7703a277b7beafb36df1635b4742634c47e
Author: HidyHan <yh2635@columbia.edu>
Date:   Fri Apr 28 18:24:57 2017 -0400

    cleanup

commit 8cbf1e935dece29b602a98420c73c18b85ca2552
Author: HidyHan <yh2635@columbia.edu>
Date:   Fri Apr 28 18:10:38 2017 -0400

    added methodcall in sast; done with (single layer) method call

commit 9bb2bf50d91c6218909e4fa95c55334e8e3555d3
Author: HidyHan <yh2635@columbia.edu>
Date:   Fri Apr 28 16:00:41 2017 -0400

    rip, I am gonna change sast

commit bebf7f5f6bed0b9c4339beccc935451e96ad1b78
Author: HidyHan <yh2635@columbia.edu>
Date:   Fri Apr 28 14:49:41 2017 -0400

    constructor and field access tentatively work; working on method
call

commit 3572976ebac83192009f312e7f8d033ff823fce4
Author: HidyHan <yh2635@columbia.edu>
Date:   Thu Apr 27 23:19:35 2017 -0400

    malloc and malloc_cast

commit 3add8cd9b87495adcc1a1ccb5149725a12db2ca8
Author: HidyHan <yh2635@columbia.edu>
Date:   Thu Apr 27 20:23:40 2017 -0400

    hello world in class passed

commit 781dade563a908e960c19510a4cd971c67fbdb98
Author: HidyHan <yh2635@columbia.edu>
```

```
Date:   Thu Apr 27 11:34:33 2017 -0400

    class: assignment, objcreate

commit 39de0984e2c63f467b99b0bf59baedc8802e065f
Author: HidyHan <yh2635@columbia.edu>
Date:   Wed Apr 26 10:50:02 2017 -0400

    class in semant

commit 401eae5260fe32f9decfb50cd09e960d97d04d53
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Wed Apr 19 22:56:17 2017 -0400

    fixed get_sast to account for classes, still has type mismatch
error

commit b06251ecfd5c6aefe4f6b144e25bc4536e3dbf23
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Wed Apr 19 21:19:35 2017 -0400

    Started code gen for classes

commit 218e053123c379bed3a46fbe33148d86d019443e
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Sun Apr 16 19:10:54 2017 -0400

    modified OOP semant a bit, but currently encountering 'Invalid
class name' error that isn't from semant

commit 4d7fa18f01e7148b77fc2b9881c189d153a0fa44
Author: Kim Tao <kmtao88@gmail.com>
Date:   Thu Apr 13 17:16:37 2017 -0400

    fixed minor bug, types should be correct now

commit 7c3bdcacd0788d2a8fb4dbcab6965e51d60a0cc0
Author: Kim Tao <kmtao88@gmail.com>
Date:   Thu Apr 13 17:00:20 2017 -0400

    convinced check_field_access to pass semant, likely works now;
renamed Obj to more descriptive ClassTyp to avoid confusion
```

```
commit b679ac9b9fb4196efb2711610db8eafcb038e3b5
Author: Kim Tao <kmtao88@gmail.com>
Date:   Thu Apr 13 13:37:52 2017 -0400

    added various test cases for if-elseif-else guaranteed return

commit da18e29654a021a32cea87ecbe083df684f0a042
Author: Kim Tao <kmtao88@gmail.com>
Date:   Thu Apr 13 13:30:47 2017 -0400

    updated semant to more thoroughly check for guaranteed return from
function

commit 56e1c44d870537c418f011160a7dbb83f715feb3
Author: Kim Tao <kmtao88@gmail.com>
Date:   Thu Apr 13 11:25:19 2017 -0400

    minor ast/parser updates to make it work with semant better

commit 04db73a8c2a50e93ccb0f03d4934081b2138216a
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Wed Apr 12 01:06:08 2017 -0400

    started OOP implementation

commit 1366f6863b8e6cfa049ebb44dcc5e85435f816b1
Merge: 3015e33 abcb9f0
Author: Kim Tao <Kimchelly@users.noreply.github.com>
Date:   Tue Apr 11 18:18:10 2017 -0400

    Merge pull request #18 from JY-H/hidy-codegen

    Hidy codegen

commit abcb9f065258784fcdb7bed5a20cb8d1da6fb32c
Author: Kim Tao <kmtao88@gmail.com>
Date:   Tue Apr 11 18:17:24 2017 -0400

    removed 1 duplicate check on function return value

commit f3a7fac5310a0ac46348f29803f1e34e2bb16378
```

```
Author: Kim Tao <kmtao88@gmail.com>
Date:   Tue Apr 11 17:00:49 2017 -0400

    fixed unreachable issue for statements after return

commit 945185613f8815d2817795f92b22d31306d872aa
Author: Kim Tao <kmtao88@gmail.com>
Date:   Tue Apr 11 16:21:33 2017 -0400

    added fix for if-elseif-else missing return on merge block; added
test cases

commit e0451402031b171e7c1b6b627089ae83dc30811d
Author: Kim Tao <kmtao88@gmail.com>
Date:   Tue Apr 11 13:51:40 2017 -0400

    reworked checking function returns properly in semant

commit 39c6a944612ee7a1fd806fc840fa672501932154
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Tue Apr 11 00:03:57 2017 -0400

    new test cannot recognize return in else statement

commit 02399c874c4ef51e91ccf8d0ffbdfdec660b3619
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Mon Apr 10 23:35:53 2017 -0400

    more testing for function calls, return type checking, nested
control flow

commit 3015e333ada231ea2f149e8d77a2b823ccfa7f4c
Merge: f2cdbfe 617ac4e
Author: HidyHan <yh2635@columbia.edu>
Date:   Sun Apr 9 09:42:34 2017 -0400

    Merge pull request #17 from JY-H/hidy-codegen

    (partial) function call implementation

commit 617ac4e8f6cb401882dbb76ea29a62347e6d693d
Author: HidyHan <yh2635@columbia.edu>
```

```
Date:   Sun Apr 9 09:42:17 2017 -0400

    function call and return tentatively function; updated README;

commit 6a82e0104628ddfa60d46bad93d09c9f97f1dbd6
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sat Apr 8 20:11:47 2017 -0400

    fixed test cases to work; reworded some code to make it less jank;
fixed spacing

commit 03a316458a1a3a8304b5fe787f47c24fdc1dfd86
Author: HidyHan <yh2635@columbia.edu>
Date:   Sat Apr 8 11:30:14 2017 -0400

    added test cases for function call

commit ad0f932b42da1ce70dd43a3bd6c5a8b63c8c2d3a
Author: HidyHan <yh2635@columbia.edu>
Date:   Sat Apr 8 11:25:49 2017 -0400

    removed build_load for func params

commit b0ea7b33460b4608c8f8022a7d19fb7b83a83f42
Author: HidyHan <yh2635@columbia.edu>
Date:   Sat Apr 8 10:09:56 2017 -0400

    reverted print back to printf so that it works

commit 41d5735b8e8af08e2e4a2e7323387709eda11f36
Author: HidyHan <yh2635@columbia.edu>
Date:   Fri Apr 7 22:25:56 2017 -0400

    made global_func_map global pointer; fixed a parsing error for
formals

commit 4ca31b3d87d79d2e847b1e580ce28794d77c6f9d
Merge: 49da01d f2cdbfe
Author: HidyHan <yh2635@columbia.edu>
Date:   Fri Apr 7 18:02:38 2017 -0400

    resolved merge conflict
```

```
commit 49da01d75b1295d7e1b06a2e2c7e42ca882dbb2c
Author: HidyHan <yh2635@columbia.edu>
Date:   Fri Apr 7 17:55:41 2017 -0400

    got rid of commented-out code

commit 025f44dd06a801f3706f3b3d25473305857aff82
Author: HidyHan <yh2635@columbia.edu>
Date:   Fri Apr 7 17:09:26 2017 -0400

    partial implementation of function call and return

commit f2cdbfe5800ad3410401f0d2962548e3d799f2f4
Merge: 91f13ba e029f2f
Author: HidyHan <yh2635@columbia.edu>
Date:   Fri Apr 7 11:20:20 2017 -0400

    Merge pull request #16 from JY-H/casting

    added final changes, added a forgotten type check for local
constants…

commit e029f2fe60bd1a1844cc4255a3852583be44b9cc
Author: Kim Tao <kmtao88@gmail.com>
Date:   Thu Apr 6 18:03:34 2017 -0400

    fixed some bugs, added const support; fixed casting

commit 33b18f740f75a962f7ebdf5bcd68c45fc7d651e2
Author: Kim Tao <kmtao88@gmail.com>
Date:   Wed Apr 5 20:59:07 2017 -0400

    added final changes, added a forgotten type check for local
constants in semant; added more test cases; updated README

commit 91f13bab7801343ca4ebe83df67024addfcb0a59
Merge: 2e543ac 7dec3f4
Author: HidyHan <yh2635@columbia.edu>
Date:   Wed Apr 5 20:33:53 2017 -0400

    Merge pull request #15 from JY-H/casting
```

```
    Casting

commit 7dec3f48d0660f3e58a0f1116d74317653ad2268
Author: Kim Tao <kmtao88@gmail.com>
Date:   Wed Apr 5 19:16:21 2017 -0400

    casting -tentatively- works for realz

commit 287f026edd43161ec2d795c18b2385bea456ec8b
Author: Kim Tao <kmtao88@gmail.com>
Date:   Wed Apr 5 16:10:59 2017 -0400

    added check_cast and basic casting, no tests lol

commit 246e1521721ece66908df463041a64f93ddaaad7
Author: Kim Tao <kmtao88@gmail.com>
Date:   Wed Apr 5 14:09:50 2017 -0400

    fixed semant check_casting; removed a bunch of warnings from
codegen due to unused (and unnecessary) variables; added KILL_ME
statements for statements to kill at the end of the project if they're
useless

commit 2e543ac321eae54dfd69bfb5295ad1f31d251de0
Merge: 777c3ab 0bfa537
Author: Kim Tao <Kimchelly@users.noreply.github.com>
Date:   Wed Apr 5 13:22:29 2017 -0400

    Merge pull request #13 from JY-H/casting

    Casting

commit 0bfa537391f8a871989343c4681d263935fddeec
Author: Kim Tao <kmtao88@gmail.com>
Date:   Wed Apr 5 13:18:05 2017 -0400

    killed prints; removed ref because it was unnecessary

commit a189cbfc5e114c7356d2d19a774810dfb6ae8056
Author: Kim Tao <kmtao88@gmail.com>
Date:   Tue Apr 4 20:46:02 2017 -0400
```

```
    removed prints

commit 59b7c01e339c2e64f376771e7fe01f0f766ec7a4
Author: Kim Tao <kmtao88@gmail.com>
Date:   Tue Apr 4 20:40:45 2017 -0400

    Continue and break -should- work, they work in nested loops

commit 5ce01dc662c2999f54f2c6a72e2677bf59416a2f
Author: Kim Tao <kmtao88@gmail.com>
Date:   Tue Apr 4 19:53:02 2017 -0400

    added for/while loops (tentatively working, but stacking blocks
probably doesn't), added test cases

commit ee02db4a940b49832ade8b1461bb1adf78f739c2
Author: Kim Tao <kmtao88@gmail.com>
Date:   Tue Apr 4 18:36:22 2017 -0400

    fixed minor binop problem typing for equality and relational ops;
removed incorrect list reversal in parser; fixed bug in if codegen;
added more test cases for boolean expressions

commit 6049f86a23b1ee2cc4ab1c799598e4f7754880dc
Author: Kim Tao <kmtao88@gmail.com>
Date:   Tue Apr 4 11:18:24 2017 -0400

    updated gitignore to ignore log files; added more scoping test
cases

commit 86a00851c15b64bf1a24019e45d04984ec9aa290
Author: Kim Tao <kmtao88@gmail.com>
Date:   Mon Apr 3 23:02:31 2017 -0400

    fixed if-elseif-else codegen branch logic; added basic test cases;
updated gitignore

commit 7f32f482951533b61df735a843e702baede088ba
Author: Kim Tao <kmtao88@gmail.com>
Date:   Mon Apr 3 22:11:46 2017 -0400
```

```
    tentatively added if-elseif-else codegen

commit 1cf559c15311e2391105332305092abb16bf43d6
Author: Kim Tao <kmtao88@gmail.com>
Date:   Mon Apr 3 12:23:14 2017 -0400

    updated semant to find types of IDs in control flow blocks

commit 481a9b348b413a5c2dc599e026d4c19e95a5bf0b
Author: Kim Tao <kmtao88@gmail.com>
Date:   Mon Apr 3 00:56:23 2017 -0400

    tentatively added block and control flow handling

commit 2a324ea9099bd70cea4bccd4e0844dcb2a7330ff
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Sat Apr 1 16:04:40 2017 -0400

    casting is now supported, but with limited functionality

commit 777c3ab47e48534665715cc722daf8e5c509e229
Merge: d9b90ba 9333eca
Author: Kylie Wu <kycwuwu@users.noreply.github.com>
Date:   Wed Mar 29 18:42:51 2017 -0400

    Merge pull request #10 from JY-H/env

    Env

commit 9333eca245b9de568b3a98a2b071425cbb95fda2
Author: Kim Tao <kmtao88@gmail.com>
Date:   Wed Mar 29 18:25:05 2017 -0400

    removed print, fixed assign_gen for SLocalVar, fixed some
formatting, added test cases

commit 2b0ea88ba3797b1f8ebcf919e2c8d78385c1f978
Author: Kim Tao <kmtao88@gmail.com>
Date:   Wed Mar 29 14:09:36 2017 -0400

    fixed id_gen, added more test cases
```

```
commit 3a87f109f37bc9699e22015033645c1807ca962b
Author: Kim Tao <kmtao88@gmail.com>
Date:   Wed Mar 29 01:06:37 2017 -0400

    Added basic binop and unop, added test cases

commit 55a65bafdadc268076bfa82371b7bba5b83cb76f
Author: Kim Tao <kmtao88@gmail.com>
Date:   Tue Mar 28 23:35:04 2017 -0400

    basic assignment works, variable declaration works

commit aac6213b5a9f08096aeb2e18389e17f304b57c1e
Author: Kim Tao <kmtao88@gmail.com>
Date:   Tue Mar 28 20:19:59 2017 -0400

    semant compiles, hello world still works

commit a4167f1b49716b1fcad39ddf65f60affe16d2b4b
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Tue Mar 28 03:38:04 2017 -0400

    adding env tracking functionality into semant, see TODOs and
private slack msg for further instructions

commit d9b90ba7840f91ca0869393f150beb69d5c43d98
Merge: 243346d 4829d1e
Author: Kylie Wu <kycwuwu@users.noreply.github.com>
Date:   Sat Mar 25 22:06:47 2017 -0400

    Merge pull request #8 from JY-H/parser_fixes

    simplified if because elseif is weird

commit 243346ddf99474bf2d24a347a9aa81f56dc2e05d
Merge: 3a845da 51780cd
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sat Mar 25 17:39:57 2017 -0400

    Merge branch 'master' of https://github.com/JY-H/DECAF

commit 3a845dae3b909128f40ad7ec2f87cde95df4631a
```

```
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sat Mar 25 17:39:20 2017 -0400

    added testall for test_*.dcf files, added test case README

commit 51780cd4bb97a222e26716259f0e2357ea8220b7
Merge: daeaedb 6412a8a
Author: JiaYan Hu <jh3541@columbia.edu>
Date:   Sat Mar 25 16:39:37 2017 -0400

    Merge pull request #7 from JY-H/hidy-codegen

    updated semant.ml so that hello_world can use sast

commit 6412a8a236cc2ef1a77749283f43343d1a4a8f89
Author: HidyHan <yh2635@columbia.edu>
Date:   Sat Mar 25 16:36:47 2017 -0400

    formatted README

commit 5069862dc2cab652f7da1f8d57588895bdff893c
Author: HidyHan <yh2635@columbia.edu>
Date:   Sat Mar 25 16:32:50 2017 -0400

    updated README for week Apr.1st

commit 5d8b65cefb2c9d8dc865750920c17df28b6c0abb
Author: HidyHan <yh2635@columbia.edu>
Date:   Sat Mar 25 14:56:40 2017 -0400

    renamed for disambiguation

commit bc21a41f926992a42b0f2704d9ae58d79c9cb34e
Author: HidyHan <yh2635@columbia.edu>
Date:   Sat Mar 25 14:21:32 2017 -0400

    wrote expr->sexpr and stmt->sstmt translations

commit 4b39cde2369dc4666fd388a2c06f54b6a1b793bd
Author: HidyHan <yh2635@columbia.edu>
Date:   Fri Mar 24 15:16:10 2017 -0400
```

```
    fixed double quote in string literal

commit daeaedb4fd4ecef4a5e1dc4140983983d3322e1b
Merge: c4723bf a82393d
Author: HidyHan <yh2635@columbia.edu>
Date:   Thu Mar 23 09:07:28 2017 -0400

    Merge pull request #6 from JY-H/codegen

    Hello World is Functional

commit a82393d0707f7ec8f39adbd5870e1d8834730e1e
Author: JiaYan Hu <jh3541@columbia.edu>
Date:   Thu Mar 23 01:17:58 2017 -0400

    Add LRM Changelist section

    * Friends pls add if anything comes up.

commit a984c1b17bddd1f43ac109e08c0b99390486bbda
Author: JiaYan Hu <jh3541@columbia.edu>
Date:   Thu Mar 23 01:11:19 2017 -0400

    Update TODOs now that hello-world is functional.

commit 51605f729b93127b0655de8554bda3d02b8826f1
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Thu Mar 23 01:02:54 2017 -0400

    codegen can now successfully execute HELLO_WORLD.

commit 6ee7afe04791f00d10930a072d7e500dd9767737
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Thu Mar 23 00:50:03 2017 -0400

    Initial stages of stmt->sstmt, expr->sexpr translation.

commit fa1f786e47d1aae21b3adcb65476cf97cdf69813
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Wed Mar 22 22:11:47 2017 -0400

    Wrap up semant.ml for hello-world purposes.
```

```
commit f3caedff62f04cccb03076bf37cf57d0f9dd157e
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Wed Mar 22 22:08:32 2017 -0400

    semant.ml now correctly detects invalid main declarations (both
global and class decs) as well as conflicts with built-in functions.

commit b87ccad9985b2f253add8598db90d56e05f93cc1
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Wed Mar 22 20:01:45 2017 -0400

    Change semant.ml and sast.ml to include built-in of print.

commit bb353c6e2de4e91c1fa8219e99a7f77f8f16b767
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Wed Mar 22 06:22:37 2017 -0400

    Stablize codegen and update TODO.

commit ed493eec138a935db538e3dedfe3a894b6c9461e
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Wed Mar 22 06:19:44 2017 -0400

    semant and sast should now be stable for purposes of generating
hello_world.

commit 62f7cec017fb18740111147af1f1d9119bb403bc
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Wed Mar 22 05:03:18 2017 -0400

    Fix all type match errors. AST changes were unnecessary and thus
were reverted.

commit d23f8d8c6ba2d45056ea34a07f8dc7bca2aec3b0
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Wed Mar 22 02:31:49 2017 -0400

    refactored structure of program, solved more compilation errors,
but now stuck on string vs Sast.sprogram type mismatch

commit 107a0d382148d7b0e475a09d8bcd9ee85c223033
```

```
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Wed Mar 22 01:51:21 2017 -0400

    added func_stub_gen and func_body_gen functionality, other
hashtables to codegen, however semant is having type mismatch issues

commit c4723bfb8ad23aef2de7b304ac14a47e3ef84a47
Merge: a64832a 70ad7a2
Author: Kylie Wu <kycwuwu@users.noreply.github.com>
Date:   Tue Mar 21 20:47:59 2017 -0400

    Merge pull request #5 from JY-H/codegen

    Initial Pass is Complete

commit 70ad7a28ddeb3600a29add11b35fce033de16e98
Author: JiaYan Hu <jh3541@columbia.edu>
Date:   Tue Mar 21 20:36:43 2017 -0400

    Update TODO-list

commit 5eecd2fb66646b08d1522b0b5461ad5814e86b3e
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Tue Mar 21 20:31:46 2017 -0400

    Add top-level decaf.ml to link components. Currently codegen only
generates IR for print function, but first pass through the compiler is
done.

commit de2da29c185647b96dfdcd775d9268e601274c29
Author: JiaYan Hu <jh3541@columbia.edu>
Date:   Tue Mar 21 20:27:48 2017 -0400

    Update usage instructions

commit 427e5a59f4412efc7acde2a567886ac60241a009
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Tue Mar 21 20:03:50 2017 -0400

    Update Makefile to generate decaf.native.

commit 98eafd421b13c55f74860bd885fc7d1083c35859
```

```
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Tue Mar 21 19:58:55 2017 -0400

    Update README.

commit a1a66dea5d89fbd774f4b1e67201dd282cd1a77e
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Tue Mar 21 19:57:38 2017 -0400

    Add executable generation instructions to README

commit 81f03aeb17e8984e01cf1ef1919ce6529278f20d
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Tue Mar 21 17:48:07 2017 -0400

    Start codegen. Main template in place.

commit 4829d1eb63acaf9c9e2ecf5ec6c504afc73a674f
Author: Kim Tao <kmtao88@gmail.com>
Date:   Mon Mar 20 12:12:31 2017 -0400

    simplified if because elseif is weird

commit a64832aa475052083c4e1a9602001d60e3ec28a0
Merge: a0be547 d280048
Author: JiaYan Hu <jh3541@columbia.edu>
Date:   Mon Mar 20 08:51:49 2017 -0400

    Merge pull request #4 from JY-H/global_decls

    Global decls

commit d2800481ca754fdcd60629443352169780ecf849
Author: Kim Tao <kmtao88@gmail.com>
Date:   Mon Mar 20 00:17:46 2017 -0400

    Re-added handling of global function calls

commit f6059bfde07e84a6cabe7415efc1b61d9ec0d1bc
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sun Mar 19 23:45:14 2017 -0400
```

```
    Minor bug fix with Id(), fixed all style

commit 1cbe827933d9777ed07cdcde2238a94159774c3f
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sun Mar 19 22:25:27 2017 -0400

    added throw keyword

commit 0e5b72c8e5fae0ebc1bdc24b234bfc2d593f6cdd
Merge: dd7281b 7b46690
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sun Mar 19 22:08:45 2017 -0400

    Merge branch 'global_decls' of https://github.com/JY-H/DECAF into
global_decls

commit dd7281bfe63082a1b6efd3812eff541e00787d1d
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sun Mar 19 22:05:22 2017 -0400

    program now allows global function declarations along with class
declarations; updated README

commit 7b4669062404f8b0ae891d5cdc2333eb7c244a6d
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sun Mar 19 22:05:22 2017 -0400

    program now allows global function declarations along with class
declarations

commit 12e583b76a38b065b7e9ac50d41a873ad1547a81
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sun Mar 19 21:25:34 2017 -0400

    Updated README since menhir tests are done; added try-catch-
finally; updated tests to include try-catch-finally

commit a0be547a95ec6bc6eb0fe3724328abfa085b7193
Merge: 680104f 7893445
Author: HidyHan <yh2635@columbia.edu>
Date:   Sun Mar 19 18:23:40 2017 -0400
```

```
    Merge pull request #2 from JY-H/id_fix

    Id fix

commit 7893445103ad42f6efbe7da3418a79d98e65058e
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sun Mar 19 17:49:11 2017 -0400

    Added self, super; updated tests

commit 36a361065b57e8838e59418e84b7b0e43307594a
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sun Mar 19 17:19:33 2017 -0400

    all parser functionality done

commit 467a50eefabc0e4182c3648a4a4804a4d82e1c78
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sun Mar 19 12:19:56 2017 -0400

    added parser tests for list and tuple

commit 19865f5689131d29e152ac6acf52fe6c86ef9394
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sat Mar 18 22:37:18 2017 -0400

    added object creation, changed list and tuple declarations to
prevent conflicts

commit df1d804441af31b407c297296919393bdfdbc325
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sat Mar 18 22:17:21 2017 -0400

    fixed test script to correctly show all failed cases

commit a68e148ea2c89b680fbeee120f13dca5c5044948
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sat Mar 18 21:15:34 2017 -0400

    Proposed fix to parser ID issue between classes and variables

commit 8235d1bfe373af7f477e8e43bbbe280d4fc44251
```

```
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sat Mar 18 21:02:28 2017 -0400

    Fixed small bug in while; added all functionality except for Object
due to reduce/reduce issue

commit 922b81097f398d406b0a534038faba7f2933b809
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sat Mar 18 18:21:44 2017 -0400

    split types into primitives and return types; slight renaming

commit e463e8851200fff3deafa3364ca145eade51aee7
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sat Mar 18 18:10:59 2017 -0400

    fixed list and tuple declarations; removed in keyword; changed List
-> Lst to avoid confusion; updated README

commit cf60b6837c82628d4fb9f25ece02c11e03f7744e
Author: Kim Tao <kmtao88@gmail.com>
Date:   Sat Mar 18 17:39:56 2017 -0400

    Added lists and tuples, access, and in operator to parser and ast

commit 157b653db9e3ea4a176f5356bedd394c5d3a58b7
Author: Kim Tao <kmtao88@gmail.com>
Date:   Fri Mar 17 17:14:24 2017 -0400

    Updated menhir tests and Makefile

commit 14a3c386a732eca9a194e120a36c7fa261518124
Author: Kim Tao <kmtao88@gmail.com>
Date:   Fri Mar 17 16:16:44 2017 -0400

    Added script to add menhir tests and quickly test parser

commit 87533901d0f43ce7abbdf1bfff0c686c3600c4bb
Author: Kim Tao <kmtao88@gmail.com>
Date:   Fri Mar 17 13:56:12 2017 -0400

    Updated README, changed _LIT -> LIT
```

```
commit a82dff6a4e2cdd619ac8de8ee676849ec1b07704
Author: Kim Tao <kmtao88@gmail.com>
Date:   Fri Mar 17 13:48:10 2017 -0400

    Revert "KYLO REN CHANGE YOUR SPACES TO TABS; changed var declaration
to an expr instead of stmt (will need a better fix later); renamed _LIT
to LIT"

    This reverts commit 632fb545d4798e99bcc40cbee9444e2969d1c02c.

commit 632fb545d4798e99bcc40cbee9444e2969d1c02c
Author: Kim Tao <kmtao88@gmail.com>
Date:   Fri Mar 17 13:35:58 2017 -0400

    KYLO REN CHANGE YOUR SPACES TO TABS; changed var declaration to an
expr instead of stmt (will need a better fix later); renamed _LIT to
LIT

commit d000bf4e5166eb222de879bb5bde312e7953ff9e
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Fri Mar 17 00:11:54 2017 -0400

    added CHAR to scanner/parser/ast, fixed tabbing in semant, added to
codegen to differentiate int/float operations, fixed spacing in codegen

commit ac34464b8f9fab8859d6812490a1f6d7774f1c84
Author: JiaYan Hu <jh3541@columbia.edu>
Date:   Thu Mar 16 19:26:13 2017 -0400

    Update README.md

commit f29f49a30a6312d191a262b9c705985ef467ccf8
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Thu Mar 16 19:19:23 2017 -0400

    edited parser to enable object field instantiation outside of class
functions, fixed ripple-down effects corresponding to the change,
removed MAIN from scanner so that the semantic checker can handle it,
added TODO to allow counter variable declaration in for loops

commit 9fa09200042b945f3d8e57c772d2c21d274efe5b
```

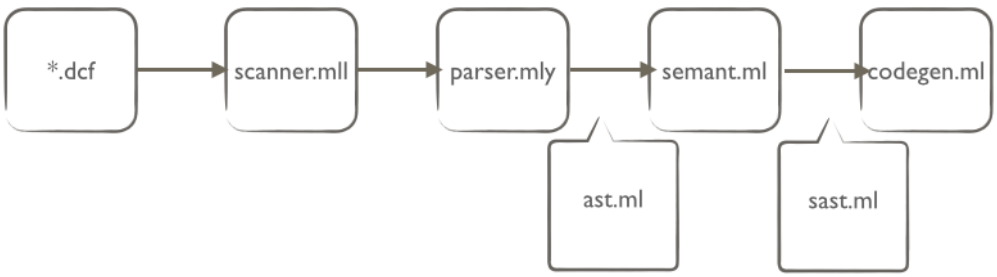
```
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Thu Mar 16 18:56:17 2017 -0400

    Add initial functionality for semantic checker -- extract
information from ast, main-checks, as well as the sast definitions.
Updated Makefile to reflect changes.

commit 5bbb98aa53012d77f0cc7cafa75c0131e7023ee9
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Thu Mar 16 15:08:13 2017 -0400

    Fix type mismatches for elseif logic. Ast and Parser now compile
down to object code without any errors.

commit e6311e7057381a6d6458029e6fbcd683cfa3f35b
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Thu Mar 16 14:26:32 2017 -0400

    Fix type mismatches and logic errors in parser and ast.

commit c7146c9b40c79e5dc144a60c8b9bf1cb08993d5a
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Thu Mar 16 14:25:10 2017 -0400

    Update Makefile to compile all the way down to object files.

commit 3d7a5e0b557cf618f766428673a0d52b373fc67b
Author: JiaYan Hu <jiayan.a.hu@gmail.com>
Date:   Thu Mar 16 12:15:22 2017 -0400

    Minor renaming and typo fixes.

commit 089c5d526d4386d6078700db283d1bbd5a020a70
Author: Kim Tao <kmtao88@gmail.com>
Date:   Thu Mar 16 09:45:31 2017 -0400

    added basic tests

commit 4c372da2e200c70d2efcbdfd64ecf8f0d9e48f9f
Author: Kim Tao <kmtao88@gmail.com>
Date:   Wed Mar 15 22:58:04 2017 -0400
```

```
    added ast to Makefile

commit d742fffcfcdf201efff475140fc78ebf2a932bf4
Author: Kim Tao <kmtao88@gmail.com>
Date:   Wed Mar 15 22:57:15 2017 -0400

    Fixed minor elseif logic issue in parser; added ast (which probably
doesn't work LOL); updated Makefile

commit b887ebd4ca68cddb50e11eef73e907c6a8782513
Author: Kim Tao <kmtao88@gmail.com>
Date:   Wed Mar 15 21:34:35 2017 -0400

    fixed scanner string literal tokens, scanner compiles; added basic
parser functionality, parser compiles; added Makefile

commit 5cec6aedfda5055f0c1cba91f2802c5c2cfb54ea
Author: Kim Tao <kmtao88@gmail.com>
Date:   Wed Mar 15 19:06:30 2017 -0400

    fixed some naming problems, started fixing multiple elseif issue

commit 2a2635ea054bee802e060c9d2aa93d2fdf588b5e
Author: Kim Tao <kmtao88@gmail.com>
Date:   Wed Mar 15 18:24:48 2017 -0400

    added class declarations to parser, added a lot of missing
functionality, fixed some unusable parts of MicroC; fixed spacing

commit c4f75548c63b5863787754bf946b103e835928ea
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Mon Mar 13 19:02:31 2017 -0400

    fixing EQ names, adding literals, editing the parser to fit DECAF

commit 0955d529362ab8bd50563c985f62674b5a837f6b
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Mon Mar 13 17:36:04 2017 -0400

    changed DECIMAL to DOT, added in decaf_string ignoring logic

commit ee6dcd6ae6951ea082d1f3d48bbe6886b8ad4532
```

```
Merge: f47d3d1 680104f
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Mon Mar 13 15:49:34 2017 -0400

    added [] and other reserved keywords

commit 680104f8f3e5ceb3dccf247b0e49ab8e0b7950c4
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Mon Mar 13 15:46:59 2017 -0400

    raw import from microc

commit f47d3d1e8a7616bce0757dc1ab798aa11612ab2b
Author: Kylie Wu <kcw2141@columbia.edu>
Date:   Mon Mar 13 15:34:51 2017 -0400

    imported microc scanner, added more tokens for DECAF

commit 7b895686d64f5c5c3da091b1ce9da3ed3022cf96
Author: JiaYan Hu <jh3541@columbia.edu>
Date:   Sat Jan 21 14:18:56 2017 -0500

    Initial commit
```

5. Architectural Design

Overview



The figure above illustrates DECAF's compiler pipeline, which follows a traditional compiler architecture design. The compilation process is divided into the following phases -- lexical analysis, syntax analysis, semantic and type checking, and code generation. Specifically, DECAF contains a total of five compilation modules. A lexical scanner (scanner.mll) is first used to read and tokenize DECAF source code files. These tokens are then fed through a parser (parser.mly), which generates an abstract syntax tree, or AST (ast.ml). The AST is then fed to a semantic checker (semant.ml) in order to generate a semantically checked and typed abstract syntax tree, or SAST (sast.ml). Finally, the code generation module (codegen.ml) traverses the SAST in order to generate LLVM as DECAF's choice of intermediate code representation. In the below sections, we describe each module in detail.

Lexical Analysis: scanner.mll

The lexical scanner is written with Ocamllex, a lexer generation utility that is a part of the OCaml distribution. This is the very first module in the compilation pipeline. The scanner is responsible for reading program texts and dividing the texts into tokens, where each token corresponds to a symbol in the programming language such as a variable name or a keyword. The scanner is also responsible for ignoring whitespace and comments, as they do not bear any significance to those outside of the program creator. Note that if a program contains illegal characters, an error will be reported.

Syntax Analysis: parser.mly, ast.ml

This phase of the compilation pipeline takes as input the list of tokens generated by the lexical analysis phase, and arranges these tokens in a tree-like structure in order to reflect the structure of the source program; such tree-like structure is called an abstract syntax tree, or AST, and is represented by the ast.ml file. Note that the parser interprets the tokens according to the operator precedence rules of the DECAF language as outlined above in the Language Reference Manual. The parser is built with OCamlyacc, a parser generation utility that is a part of the OCaml distribution.

As the goal of the parser is to generate an AST which resembles the structure of the program, the top-level abstraction of the AST is essentially a global declaration record, which includes both class declaration records and function declaration records. The class and function declaration records each then contain their own relevant fields. For example, a function declaration record contains fields regarding the function's return type, function names, formal parameters, etc. The abstractions of the abstract syntax tree thus resemble that of an acceptable DECAF source program.

Semantic and Type Checking: semant.ml, sast.ml

Though the AST generated by the syntax analysis phase may resemble the structure of a DECAF program, it may still contain semantic inconsistencies, such as type mismatches or usage of variable prior to declaration. Thus it is the job of the semantic and type checking phase to ensure that the program is semantically correct. This is accomplished through the semant.ml module, which is written in OCaml. It takes as input the generated AST from the syntax analysis phase, and produces a semantically and type checked AST, or SAST, and is represented by the sast.ml file.

The semantic checker module recursively traverses the AST, examining each node and generating the corresponding SAST node. Below are the main functionalities of the semantic checker:

- Check for valid operations, including binop, arithmetic, relational, equality, etc. This also involves operations on arrays and class objects.
- Check for duplicate fields or methods within the same class.
- Checking for type mismatches.
- Check for exactly one main declaration within the program.

Intermediate Code Generation: codegen.ml

This is the last phase of the compilation pipeline. The code generation module takes as input the SAST from the semantic checking phase, and produces corresponding LLVM IR. This module is written using the LLVM OCaml library, which is comprehensively documented.

6. Test Plan

Overview

Our testing can be divided into the following stages:

- Unit Testing: Following the implementation of every functionality, small test *.dcf test programs are written for that functionality in order to ensure that the functionality is working as intended. *.cmp files contained the expected output so that our test script could compare the actual output and determine if the test worked as expected.
- Integration Testing: Following the completion of core language features, larger test programs are written to test and demonstrate overall functionalities of the language.

Example Programs

test_integration_arraylist.dcf

```
class ArrayList {
	// Buffer
	[int] data;
	// Current size
	int size;
	// Capacity
	int capacity;

	/* Creates the list */
	ArrayList(int capacity) -> ArrayList {
		self.size = 0;
		if (capacity === 0) {
			self.capacity = 10;
		}
		else {
			self.capacity = capacity;
		}
		self.data = [int, self.capacity];

		int i;
		for (i = 0; i < self.capacity; i = i + 1) {
			self.data[i] = 0;
		}
	}

	/* Fills the list up to current size with dummy elements */
	fill(int newSize) -> void {
		int i;
		for (i = 0; i < newSize; i = i + 1) {
			self.data[i] = i;
		}
		self.size = newSize;
	}

	/* Gets the element at the index */
	get(int index) -> int {
		if (not (0 <= index and index <= self.size)) {
			print_string("Error: index out of bounds\n");
			return -1;
```

```
	}
	return self.data[index];
}

/* Set 1 element in list */
set(int index, int val) -> void {
	if (not (0 <= index and index <= self.size)) {
		print_string("Error: index out of bounds\n");
		return;
	}
	self.data[index] = val;
}

/* Add 1 element to end of list */
add(int val) -> void {
	if (self.size >= self.capacity - 1) {
		self.capacity = self.capacity * 2;
		[int] newData = [int, self.capacity];
		int i;
		for (i = 0; i < self.capacity; i = i + 1) {
			newData[i] = self.data[i];
		}
		self.data = newData;
	}

	self.data[self.size] = val;
	self.size = self.size + 1;
}

/* Removes at index */
remove(int index) -> void {
	if (not (0 <= index and index <= self.size)) {
		print_string("Error: index out of bounds\n");
		return;
	}

	// Shift over all elements
	int i;
	for (i = index; i < self.size - 1; i = i + 1) {
		self.data[i] = self.data[i + 1];
	}
```

```
            self.size = self.size - 1;
        }

        print() -> void {
            int i;
            for (i = 0; i < self.size; i = i + 1) {
                print_int(self.data[i]);

                if (i !== self.size - 1) {
                    print_string(" ");
                }
            }
            print_string("\n");
        }
}

main() -> int {
        int len = 4;
        ArrayList lst1 = ArrayList(len);
        // Test fill()
        lst1.fill(lst1.capacity);

        // Test set
        int i;
        for (i = 0; i < lst1.size; i = i + 1) {
            lst1.set(i, i * 100);
        }

        // Test print()
        // 0 100 200 300
        lst1.print();

        ArrayList lst2 = ArrayList(len);

        // Test set()
        for (i = 0; i < lst1.size; i = i + 1) {
            if (i >= lst2.size) {
                lst2.add(0);
            }
            lst2.set(i, lst1.get(i) / 100);
        }
```

```
        // 0 1 2 3
        lst2.print();

        // Test add()
        for (i = 0; i < 10; i = i + 1) {
              lst2.add(i * 100);
        }

        // 0 1 2 3 0 100 200 300 400 500 600 700 800 900
        lst2.print();

        ArrayList lst3 = ArrayList(lst2.size / 2);
        int end = lst3.capacity;
        for (i = 0; i < end; i = i + 1) {
              lst3.add(i * 2);
        }

        // 0 2 4 6 8 10 12
        lst3.print();

        // Test remove()
        for (i = lst3.size - 1; i >= 0; i = i - 1) {
              int index = lst3.get(i);
              lst2.remove(index);
        }

        // 1 3 100 300 500 700 900
        lst2.print();

        return 0;
   }
```

test_integration_arraylist.ll

```
%ArrayList = type <{ i32, i32*, i32 }>

   @tmp = private unnamed_addr constant [28 x i8] c"Error: index out of
   bounds\0A\00"
   @tmp1 = private unnamed_addr constant [3 x i8] c"%s\00"
   @tmp2 = private unnamed_addr constant [3 x i8] c"%d\00"
   @tmp3 = private unnamed_addr constant [2 x i8] c" \00"
   @tmp4 = private unnamed_addr constant [3 x i8] c"%s\00"
   @tmp5 = private unnamed_addr constant [2 x i8] c"\0A\00"
```

```
@tmp6 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp7 = private unnamed_addr constant [28 x i8] c"Error: index out of
bounds\0A\00"
@tmp8 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp9 = private unnamed_addr constant [28 x i8] c"Error: index out of
bounds\0A\00"
@tmp10 = private unnamed_addr constant [3 x i8] c"%s\00"

declare i32 @printf(i8*, ...)

declare noalias i8* @malloc(i32)

define %ArrayList* @ArrayList(i32 %capacity) {
entry:
  %self = alloca %ArrayList
  %tmp = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1** null,
i32 1) to i32))
  %tmp1 = bitcast i8* %tmp to %ArrayList*
  %tmp2 = load %ArrayList* %tmp1
  store %ArrayList %tmp2, %ArrayList* %self
  %size = getelementptr inbounds %ArrayList* %self, i32 0, i32 0
  store i32 0, i32* %size
  br label %if

if:                                               ; preds = %entry
  %int_eqtmp = icmp eq i32 %capacity, 0
  br i1 %int_eqtmp, label %if_body, label %else_body

if_body:                                          ; preds = %if
  %capacity3 = getelementptr inbounds %ArrayList* %self, i32 0, i32 2
  store i32 10, i32* %capacity3
  br label %merge

else_body:                                        ; preds = %if
  %capacity4 = getelementptr inbounds %ArrayList* %self, i32 0, i32 2
  store i32 %capacity, i32* %capacity4
  br label %merge

merge:                                            ; preds = %else_body,
%if_body
  %data = getelementptr inbounds %ArrayList* %self, i32 0, i32 1
  %capacity5 = getelementptr inbounds %ArrayList* %self, i32 0, i32 2
```

```
  %capacity6 = load i32* %capacity5
  %tmp7 = mul i32 ptrtoint (i1** getelementptr (i1** null, i32 1) to
i32), %capacity6
  %tmp8 = add i32 %tmp7, 1
  %mallocsize = mul i32 %tmp8, ptrtoint (i1** getelementptr (i1** null,
i32 1) to i32)
  %malloccall = tail call i8* @malloc(i32 %mallocsize)
  %tmp9 = bitcast i8* %malloccall to i32**
  %tmp10 = bitcast i32** %tmp9 to i32*
  store i32 %tmp8, i32* %tmp10
  store i32* %tmp10, i32** %data
  %i = alloca i32
  store i32 0, i32* %i
  br label %loop_cond
  br label %loop_step

loop_step:                                        ; preds = %loop_body,
%merge
  %i11 = load i32* %i
  %int_addtmp = add i32 %i11, 1
  store i32 %int_addtmp, i32* %i
  br label %loop_cond

loop_cond:                                        ; preds = %loop_step,
%merge
  %i12 = load i32* %i
  %capacity13 = getelementptr inbounds %ArrayList* %self, i32 0, i32 2
  %capacity14 = load i32* %capacity13
  %int_lesstmp = icmp slt i32 %i12, %capacity14
  br i1 %int_lesstmp, label %loop_body, label %loop_exit

loop_exit:                                        ; preds = %loop_cond
  ret %ArrayList* %self
  unreachable

loop_body:                                        ; preds = %loop_cond
  %data15 = getelementptr inbounds %ArrayList* %self, i32 0, i32 1
  %data16 = load i32** %data15
  %i17 = load i32* %i
  %list_index = add i32 %i17, 1
  %list_access = getelementptr i32* %data16, i32 %list_index
  store i32 0, i32* %list_access
```

```
  br label %loop_step
}

define void @ArrayList.add(%ArrayList* %self, i32 %val) {
entry:
  br label %if

if:                                               ; preds = %entry
  %size = getelementptr inbounds %ArrayList* %self, i32 0, i32 0
  %size20 = load i32* %size
  %capacity21 = getelementptr inbounds %ArrayList* %self, i32 0, i32 2
  %capacity22 = load i32* %capacity21
  %int_subtmp = sub i32 %capacity22, 1
  %int_geqtmp = icmp sge i32 %size20, %int_subtmp
  br i1 %int_geqtmp, label %if_body, label %else_body

if_body:                                          ; preds = %if
  %capacity = getelementptr inbounds %ArrayList* %self, i32 0, i32 2
  %capacity1 = getelementptr inbounds %ArrayList* %self, i32 0, i32 2
  %capacity2 = load i32* %capacity1
  %int_multop = mul i32 %capacity2, 2
  store i32 %int_multop, i32* %capacity
  %newData = alloca i32*
  %capacity3 = getelementptr inbounds %ArrayList* %self, i32 0, i32 2
  %capacity4 = load i32* %capacity3
  %tmp = mul i32 ptrtoint (i1** getelementptr (i1** null, i32 1) to
i32), %capacity4
  %tmp5 = add i32 %tmp, 1
  %mallocsize = mul i32 %tmp5, ptrtoint (i1** getelementptr (i1** null,
i32 1) to i32)
  %malloccall = tail call i8* @malloc(i32 %mallocsize)
  %tmp6 = bitcast i8* %malloccall to i32**
  %tmp7 = bitcast i32** %tmp6 to i32*
  store i32 %tmp5, i32* %tmp7
  store i32* %tmp7, i32** %newData
  %i = alloca i32
  store i32 0, i32* %i
  br label %loop_cond
  br label %loop_step

loop_step:                                        ; preds = %loop_body,
%if_body
```

```
  %i8 = load i32* %i
  %int_addtmp = add i32 %i8, 1
  store i32 %int_addtmp, i32* %i
  br label %loop_cond

loop_cond:                                        ; preds = %loop_step,
%if_body
  %i9 = load i32* %i
  %capacity10 = getelementptr inbounds %ArrayList* %self, i32 0, i32 2
  %capacity11 = load i32* %capacity10
  %int_lesstmp = icmp slt i32 %i9, %capacity11
  br i1 %int_lesstmp, label %loop_body, label %loop_exit

loop_exit:                                        ; preds = %loop_cond
  %data18 = getelementptr inbounds %ArrayList* %self, i32 0, i32 1
  %newData19 = load i32** %newData
  store i32* %newData19, i32** %data18
  br label %merge

loop_body:                                        ; preds = %loop_cond
  %newData12 = load i32** %newData
  %i13 = load i32* %i
  %list_index = add i32 %i13, 1
  %list_access = getelementptr i32* %newData12, i32 %list_index
  %data = getelementptr inbounds %ArrayList* %self, i32 0, i32 1
  %data14 = load i32** %data
  %i15 = load i32* %i
  %list_index16 = add i32 %i15, 1
  %list_access17 = getelementptr i32* %data14, i32 %list_index16
  %list_access_val = load i32* %list_access17
  store i32 %list_access_val, i32* %list_access
  br label %loop_step

else_body:                                        ; preds = %if
  br label %merge

merge:                                            ; preds = %else_body,
%loop_exit
  %data23 = getelementptr inbounds %ArrayList* %self, i32 0, i32 1
  %data24 = load i32** %data23
  %size25 = getelementptr inbounds %ArrayList* %self, i32 0, i32 0
  %size26 = load i32* %size25
```

```
  %list_index27 = add i32 %size26, 1
  %list_access28 = getelementptr i32* %data24, i32 %list_index27
  store i32 %val, i32* %list_access28
  %size29 = getelementptr inbounds %ArrayList* %self, i32 0, i32 0
  %size30 = getelementptr inbounds %ArrayList* %self, i32 0, i32 0
  %size31 = load i32* %size30
  %int_addtmp32 = add i32 %size31, 1
  store i32 %int_addtmp32, i32* %size29
  ret void
  unreachable
}

define void @ArrayList.fill(%ArrayList* %self, i32 %newSize) {
entry:
  %i = alloca i32
  store i32 0, i32* %i
  br label %loop_cond
  br label %loop_step

loop_step:                                        ; preds = %loop_body,
%entry
  %i1 = load i32* %i
  %int_addtmp = add i32 %i1, 1
  store i32 %int_addtmp, i32* %i
  br label %loop_cond

loop_cond:                                        ; preds = %loop_step,
%entry
  %i2 = load i32* %i
  %int_lesstmp = icmp slt i32 %i2, %newSize
  br i1 %int_lesstmp, label %loop_body, label %loop_exit

loop_exit:                                        ; preds = %loop_cond
  %size = getelementptr inbounds %ArrayList* %self, i32 0, i32 0
  store i32 %newSize, i32* %size
  ret void
  unreachable

loop_body:                                        ; preds = %loop_cond
  %data = getelementptr inbounds %ArrayList* %self, i32 0, i32 1
  %data3 = load i32** %data
  %i4 = load i32* %i
```

```
  %list_index = add i32 %i4, 1
  %list_access = getelementptr i32* %data3, i32 %list_index
  %i5 = load i32* %i
  store i32 %i5, i32* %list_access
  br label %loop_step
}

define i32 @ArrayList.get(%ArrayList* %self, i32 %index) {
entry:
  br label %if

if:                                               ; preds = %entry
  %int_leqtmp = icmp sle i32 0, %index
  %size = getelementptr inbounds %ArrayList* %self, i32 0, i32 0
  %size1 = load i32* %size
  %int_leqtmp2 = icmp sle i32 %index, %size1
  %int_andtmp = and i1 %int_leqtmp, %int_leqtmp2
  %not_bool_tmp = xor i1 %int_andtmp, true
  br i1 %not_bool_tmp, label %if_body, label %else_body

if_body:                                          ; preds = %if
  %print = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([3
x i8]* @tmp1, i32 0, i32 0), i8* getelementptr inbounds ([28 x i8]*
@tmp, i32 0, i32 0))
  ret i32 -1
  br label %merge

else_body:                                        ; preds = %if
  br label %merge

merge:                                            ; preds = %else_body,
%if_body
  %data = getelementptr inbounds %ArrayList* %self, i32 0, i32 1
  %data3 = load i32** %data
  %list_index = add i32 %index, 1
  %list_access = getelementptr i32* %data3, i32 %list_index
  %list_access_val = load i32* %list_access
  ret i32 %list_access_val
  unreachable
}

define void @ArrayList.print(%ArrayList* %self) {
```

```
entry:
  %i = alloca i32
  store i32 0, i32* %i
  br label %loop_cond
  br label %loop_step

loop_step:                                        ; preds = %merge,
%entry
  %i1 = load i32* %i
  %int_addtmp = add i32 %i1, 1
  store i32 %int_addtmp, i32* %i
  br label %loop_cond

loop_cond:                                        ; preds = %loop_step,
%entry
  %i2 = load i32* %i
  %size = getelementptr inbounds %ArrayList* %self, i32 0, i32 0
  %size3 = load i32* %size
  %int_lesstmp = icmp slt i32 %i2, %size3
  br i1 %int_lesstmp, label %loop_body, label %loop_exit

loop_exit:                                        ; preds = %loop_cond
  %print10 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds
([3 x i8]* @tmp6, i32 0, i32 0), i8* getelementptr inbounds ([2 x i8]*
@tmp5, i32 0, i32 0))
  ret void
  unreachable

loop_body:                                        ; preds = %loop_cond
  %data = getelementptr inbounds %ArrayList* %self, i32 0, i32 1
  %data4 = load i32** %data
  %i5 = load i32* %i
  %list_index = add i32 %i5, 1
  %list_access = getelementptr i32* %data4, i32 %list_index
  %list_access_val = load i32* %list_access
  %print = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([3
x i8]* @tmp2, i32 0, i32 0), i32 %list_access_val)
  br label %if

if:                                               ; preds = %loop_body
  %i7 = load i32* %i
  %size8 = getelementptr inbounds %ArrayList* %self, i32 0, i32 0
```

```
  %size9 = load i32* %size8
  %int_subtmp = sub i32 %size9, 1
  %int_neqtmp = icmp ne i32 %i7, %int_subtmp
  br i1 %int_neqtmp, label %if_body, label %else_body

if_body:                                          ; preds = %if
  %print6 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([3
x i8]* @tmp4, i32 0, i32 0), i8* getelementptr inbounds ([2 x i8]*
@tmp3, i32 0, i32 0))
  br label %merge

else_body:                                        ; preds = %if
  br label %merge

merge:                                            ; preds = %else_body,
%if_body
  br label %loop_step
}

define void @ArrayList.remove(%ArrayList* %self, i32 %index) {
entry:
  br label %if

if:                                               ; preds = %entry
  %int_leqtmp = icmp sle i32 0, %index
  %size = getelementptr inbounds %ArrayList* %self, i32 0, i32 0
  %size1 = load i32* %size
  %int_leqtmp2 = icmp sle i32 %index, %size1
  %int_andtmp = and i1 %int_leqtmp, %int_leqtmp2
  %not_bool_tmp = xor i1 %int_andtmp, true
  br i1 %not_bool_tmp, label %if_body, label %else_body

if_body:                                          ; preds = %if
  %print = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([3
x i8]* @tmp8, i32 0, i32 0), i8* getelementptr inbounds ([28 x i8]*
@tmp7, i32 0, i32 0))
  ret void
  br label %merge

else_body:                                        ; preds = %if
  br label %merge
```

```
merge:                                            ; preds = %else_body,
%if_body
  %i = alloca i32
  store i32 %index, i32* %i
  br label %loop_cond
  br label %loop_step

loop_step:                                        ; preds = %loop_body,
%merge
  %i3 = load i32* %i
  %int_addtmp = add i32 %i3, 1
  store i32 %int_addtmp, i32* %i
  br label %loop_cond

loop_cond:                                        ; preds = %loop_step,
%merge
  %i4 = load i32* %i
  %size5 = getelementptr inbounds %ArrayList* %self, i32 0, i32 0
  %size6 = load i32* %size5
  %int_subtmp = sub i32 %size6, 1
  %int_lesstmp = icmp slt i32 %i4, %int_subtmp
  br i1 %int_lesstmp, label %loop_body, label %loop_exit

loop_exit:                                        ; preds = %loop_cond
  %size15 = getelementptr inbounds %ArrayList* %self, i32 0, i32 0
  %size16 = getelementptr inbounds %ArrayList* %self, i32 0, i32 0
  %size17 = load i32* %size16
  %int_subtmp18 = sub i32 %size17, 1
  store i32 %int_subtmp18, i32* %size15
  ret void
  unreachable

loop_body:                                        ; preds = %loop_cond
  %data = getelementptr inbounds %ArrayList* %self, i32 0, i32 1
  %data7 = load i32** %data
  %i8 = load i32* %i
  %list_index = add i32 %i8, 1
  %list_access = getelementptr i32* %data7, i32 %list_index
  %data9 = getelementptr inbounds %ArrayList* %self, i32 0, i32 1
  %data10 = load i32** %data9
  %i11 = load i32* %i
  %int_addtmp12 = add i32 %i11, 1
```

```
  %list_index13 = add i32 %int_addtmp12, 1
  %list_access14 = getelementptr i32* %data10, i32 %list_index13
  %list_access_val = load i32* %list_access14
  store i32 %list_access_val, i32* %list_access
  br label %loop_step
}
define void @ArrayList.set(%ArrayList* %self, i32 %index, i32 %val) {
entry:
  br label %if

if:                                               ; preds = %entry
  %int_leqtmp = icmp sle i32 0, %index
  %size = getelementptr inbounds %ArrayList* %self, i32 0, i32 0
  %size1 = load i32* %size
  %int_leqtmp2 = icmp sle i32 %index, %size1
  %int_andtmp = and i1 %int_leqtmp, %int_leqtmp2
  %not_bool_tmp = xor i1 %int_andtmp, true
  br i1 %not_bool_tmp, label %if_body, label %else_body

if_body:                                          ; preds = %if
  %print = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([3
x i8]* @tmp10, i32 0, i32 0), i8* getelementptr inbounds ([28 x i8]*
@tmp9, i32 0, i32 0))
  ret void
  br label %merge

else_body:                                        ; preds = %if
  br label %merge

merge:                                            ; preds = %else_body,
%if_body
  %data = getelementptr inbounds %ArrayList* %self, i32 0, i32 1
  %data3 = load i32** %data
  %list_index = add i32 %index, 1
  %list_access = getelementptr i32* %data3, i32 %list_index
  store i32 %val, i32* %list_access
  ret void
  unreachable
}

define i32 @main() {
entry:
```

```
  %len = alloca i32
  store i32 4, i32* %len
  %lst1 = alloca %ArrayList
  %len1 = load i32* %len
  %tmp = call %ArrayList* @ArrayList(i32 %len1)
  %tmp2 = load %ArrayList* %tmp
  store %ArrayList %tmp2, %ArrayList* %lst1
  %capacity = getelementptr inbounds %ArrayList* %lst1, i32 0, i32 2
  %capacity3 = load i32* %capacity
  call void @ArrayList.fill(%ArrayList* %lst1, i32 %capacity3)
  %i = alloca i32
  store i32 0, i32* %i
  br label %loop_cond
  br label %loop_step

loop_step:                                        ; preds = %loop_body,
%entry
  %i4 = load i32* %i
  %int_addtmp = add i32 %i4, 1
  store i32 %int_addtmp, i32* %i
  br label %loop_cond

loop_cond:                                        ; preds = %loop_step,
%entry
  %i5 = load i32* %i
  %size = getelementptr inbounds %ArrayList* %lst1, i32 0, i32 0
  %size6 = load i32* %size
  %int_lesstmp = icmp slt i32 %i5, %size6
  br i1 %int_lesstmp, label %loop_body, label %loop_exit

loop_exit:                                        ; preds = %loop_cond
  call void @ArrayList.print(%ArrayList* %lst1)
  %lst2 = alloca %ArrayList
  %len9 = load i32* %len
  %tmp10 = call %ArrayList* @ArrayList(i32 %len9)
  %tmp11 = load %ArrayList* %tmp10
  store %ArrayList %tmp11, %ArrayList* %lst2
  store i32 0, i32* %i
  br label %loop_cond14
  br label %loop_step13

loop_step13:                                      ; preds = %merge,
```

```
%loop_exit
  %i16 = load i32* %i
  %int_addtmp17 = add i32 %i16, 1
  store i32 %int_addtmp17, i32* %i
  br label %loop_cond14

loop_cond14:                                      ; preds =
%loop_step13, %loop_exit
  %i18 = load i32* %i
  %size19 = getelementptr inbounds %ArrayList* %lst1, i32 0, i32 0
  %size20 = load i32* %size19
  %int_lesstmp21 = icmp slt i32 %i18, %size20
  br i1 %int_lesstmp21, label %loop_body12, label %loop_exit15

loop_exit15:                                      ; preds = %loop_cond14
  call void @ArrayList.print(%ArrayList* %lst2)
  store i32 0, i32* %i
  br label %loop_cond30
  br label %loop_step29

loop_step29:                                      ; preds =
%loop_body28, %loop_exit15
  %i32 = load i32* %i
  %int_addtmp33 = add i32 %i32, 1
  store i32 %int_addtmp33, i32* %i
  br label %loop_cond30

loop_cond30:                                      ; preds =
%loop_step29, %loop_exit15
  %i34 = load i32* %i
  %int_lesstmp35 = icmp slt i32 %i34, 10
  br i1 %int_lesstmp35, label %loop_body28, label %loop_exit31

loop_exit31:                                      ; preds = %loop_cond30
  call void @ArrayList.print(%ArrayList* %lst2)
  %lst3 = alloca %ArrayList
  %size38 = getelementptr inbounds %ArrayList* %lst2, i32 0, i32 0
  %size39 = load i32* %size38
  %int_divop40 = sdiv i32 %size39, 2
  %tmp41 = call %ArrayList* @ArrayList(i32 %int_divop40)
  %tmp42 = load %ArrayList* %tmp41
  store %ArrayList %tmp42, %ArrayList* %lst3
```

```
  %end = alloca i32
  %capacity43 = getelementptr inbounds %ArrayList* %lst3, i32 0, i32 2
  %capacity44 = load i32* %capacity43
  store i32 %capacity44, i32* %end
  store i32 0, i32* %i
  br label %loop_cond47
  br label %loop_step46

loop_step46:                                      ; preds =
%loop_body45, %loop_exit31
  %i49 = load i32* %i
  %int_addtmp50 = add i32 %i49, 1
  store i32 %int_addtmp50, i32* %i
  br label %loop_cond47

loop_cond47:                                      ; preds =
%loop_step46, %loop_exit31
  %i51 = load i32* %i
  %end52 = load i32* %end
  %int_lesstmp53 = icmp slt i32 %i51, %end52
  br i1 %int_lesstmp53, label %loop_body45, label %loop_exit48

loop_exit48:                                      ; preds = %loop_cond47
  call void @ArrayList.print(%ArrayList* %lst3)
  %size56 = getelementptr inbounds %ArrayList* %lst3, i32 0, i32 0
  %size57 = load i32* %size56
  %int_subtmp = sub i32 %size57, 1
  store i32 %int_subtmp, i32* %i
  br label %loop_cond60
  br label %loop_step59

loop_step59:                                      ; preds =
%loop_body58, %loop_exit48
  %i62 = load i32* %i
  %int_subtmp63 = sub i32 %i62, 1
  store i32 %int_subtmp63, i32* %i
  br label %loop_cond60

loop_cond60:                                      ; preds =
%loop_step59, %loop_exit48
  %i64 = load i32* %i
  %int_geqtmp65 = icmp sge i32 %i64, 0
```

```
  br i1 %int_geqtmp65, label %loop_body58, label %loop_exit61

loop_exit61:                                      ; preds = %loop_cond60
  call void @ArrayList.print(%ArrayList* %lst2)
  ret i32 0
  unreachable

loop_body:                                        ; preds = %loop_cond
  %i7 = load i32* %i
  %i8 = load i32* %i
  %int_multop = mul i32 %i8, 100
  call void @ArrayList.set(%ArrayList* %lst1, i32 %i7, i32 %int_multop)
  br label %loop_step

loop_body12:                                      ; preds = %loop_cond14
  br label %if

if:                                               ; preds =
%loop_body12
  %i22 = load i32* %i
  %size23 = getelementptr inbounds %ArrayList* %lst2, i32 0, i32 0
  %size24 = load i32* %size23
  %int_geqtmp = icmp sge i32 %i22, %size24
  br i1 %int_geqtmp, label %if_body, label %else_body

if_body:                                          ; preds = %if
  call void @ArrayList.add(%ArrayList* %lst2, i32 0)
  br label %merge

else_body:                                        ; preds = %if
  br label %merge

merge:                                            ; preds = %else_body,
%if_body
  %i25 = load i32* %i
  %i26 = load i32* %i
  %tmp27 = call i32 @ArrayList.get(%ArrayList* %lst1, i32 %i26)
  %int_divop = sdiv i32 %tmp27, 100
  call void @ArrayList.set(%ArrayList* %lst2, i32 %i25, i32 %int_divop)
  br label %loop_step13

loop_body28:                                      ; preds = %loop_cond30
```

```
  %i36 = load i32* %i
  %int_multop37 = mul i32 %i36, 100
  call void @ArrayList.add(%ArrayList* %lst2, i32 %int_multop37)
  br label %loop_step29

loop_body45:                                      ; preds = %loop_cond47
  %i54 = load i32* %i
  %int_multop55 = mul i32 %i54, 2
  call void @ArrayList.add(%ArrayList* %lst3, i32 %int_multop55)
  br label %loop_step46

loop_body58:                                      ; preds = %loop_cond60
  %index = alloca i32
  %i66 = load i32* %i
  %tmp67 = call i32 @ArrayList.get(%ArrayList* %lst3, i32 %i66)
  store i32 %tmp67, i32* %index
  %index68 = load i32* %index
  call void @ArrayList.remove(%ArrayList* %lst2, i32 %index68)
  br label %loop_step59
}
```

test_integration_bankacct.dcf

```
class Bank_Acct {
   string owner;
   int balance;
   float interest;

   Bank_Acct(string o, int b, float i) -> Bank_Acct {
         self.owner = o;
         self.balance = b;
         self.interest = i;
   }

   acct_summary() -> void {
         print_string(self.owner);
         print_string("\'s account earns ");
         print_float(self.interest);
         print_string(" interest and currently has a balance of $");
         print_int(<int>((<float> self.balance)/100.0));
         print_string(".\n");
   }
```

```
    deposit(int money) -> void {
          if (money < 0) {
                print_string("Value cannot be negative.\n");
          } else {
                int new_balance = self.balance + money;
                self.balance = new_balance;
                print_string(self.owner);
                print_string("\'s new balance is now: $");
                print_int(<int>((<float> self.balance)/100.0));
                print_string(".\n");
          }
    }

    withdraw(int money) -> void {
          if (money < 0) {
                print_string("Value cannot be negative.\n");
          } else {
                int new_balance = self.balance - money;
                if (new_balance < 0) {
                      print_string("Cannot withdraw this much.\n");
                } else {
                      self.balance = new_balance;
                      print_string(self.owner);
                      print_string("\'s new balance is now: $");
                      print_int(<int>((<float> self.balance)/100.0));
                      print_string(".\n");
                }
          }
    }

    calc_interest() -> void {
          float balance = <float> self.balance;
          self.balance = <int> (balance * (1.0 + self.interest));
          print_string("After a year's cumulative interest, ");
          print_string(self.owner);
          print_string("\'s new balance is now: $");
          print_int(<int>((<float> self.balance)/100.0));
          print_string(".\n");
    }
}

// when balance on card is negative, indicates amount owed
```

```
class Credit_Card extends Bank_Acct {
   int minimum_payment;

   Credit_Card (string o, int b, float i, int mp) -> Credit_Card {
         self.owner = o;
         self.balance = b;
         self.interest = i;
         self.minimum_payment = mp;
   }

   pay(int money) -> void {
         if (money < self.minimum_payment) {
               print_string("Payment must be greater than $");
               print_int(self.minimum_payment);
               print_string(".\n");
         } else {
               int new_balance = self.balance + money;
               self.balance = new_balance;
         }
   }

   acct_summary() -> void {
         print_string(self.owner);
         print_string("\'s account pays ");
         print_float(self.interest);
         print_string(" interest and currently has a balance of $");
         print_int(<int>((<float> self.balance)/100.0));
         print_string(". ");
         if (self.balance < 0) {
               print_string("Account overdue!\n");
         } else {
               print_string("\n");
         }
   }

}

main() -> int {
   Bank_Acct acct0 = Bank_Acct("Terrence", 50000, 0.01); // 500 dollars,
   0.01 interest rate
   Bank_Acct acct1 = Bank_Acct("Aly", 50000, 0.025);
   Bank_Acct acct2 = Bank_Acct("Clara", 50000, 0.05);
```

```
    Credit_Card card0 = Credit_Card("Holly", -20089, 0.32, 24);
    Credit_Card card1 = Credit_Card("Darren", -100034, 0.27, 42);

    [Bank_Acct] accounts = [Bank_Acct, 3];
    accounts[0] = acct0;
    accounts[1] = acct1;
    accounts[2] = acct2;
    [Credit_Card] cards = [Credit_Card, 2];
    cards[0] = card0;
    cards[1] = card1;

    int i = 0;
    for (i = 0; i < 3; i = i + 1) {
          Bank_Acct temp = accounts[i];
          temp.acct_summary();
    }
    for (i = 0; i < 3; i = i + 1) {
          Bank_Acct temp = accounts[i];
          temp.calc_interest();
    }
    for (i = 0; i < 2; i = i + 1) {
          Credit_Card temp = cards[i];
          temp.pay(252);
          temp.acct_summary();
    }

    return 0;
}
```

test_integration_bankacct.ll

```
%Bank_Acct = type <{ i8*, double, i32 }>
%Credit_Card = type <{ i8*, i32, double, i32 }>

@tmp = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.1 = private unnamed_addr constant [18 x i8] c"'s account earns \00"
@tmp.2 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.3 = private unnamed_addr constant [3 x i8] c"%f\00"
@tmp.4 = private unnamed_addr constant [43 x i8] c" interest and currently has
a balance of $\00"
@tmp.5 = private unnamed_addr constant [3 x i8] c"%s\00"
```

```
@tmp.6 = private unnamed_addr constant [3 x i8] c"%d\00"
@tmp.7 = private unnamed_addr constant [3 x i8] c".\0A\00"
@tmp.8 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.9 = private unnamed_addr constant [37 x i8] c"After a year's cumulative
interest, \00"
@tmp.10 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.11 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.12 = private unnamed_addr constant [25 x i8] c"'s new balance is now:
$\00"
@tmp.13 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.14 = private unnamed_addr constant [3 x i8] c"%d\00"
@tmp.15 = private unnamed_addr constant [3 x i8] c".\0A\00"
@tmp.16 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.17 = private unnamed_addr constant [27 x i8] c"Value cannot be
negative.\0A\00"
@tmp.18 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.19 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.20 = private unnamed_addr constant [25 x i8] c"'s new balance is now:
$\00"
@tmp.21 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.22 = private unnamed_addr constant [3 x i8] c"%d\00"
@tmp.23 = private unnamed_addr constant [3 x i8] c".\0A\00"
@tmp.24 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.25 = private unnamed_addr constant [27 x i8] c"Value cannot be
negative.\0A\00"
@tmp.26 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.27 = private unnamed_addr constant [28 x i8] c"Cannot withdraw this
much.\0A\00"
@tmp.28 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.29 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.30 = private unnamed_addr constant [25 x i8] c"'s new balance is now:
$\00"
@tmp.31 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.32 = private unnamed_addr constant [3 x i8] c"%d\00"
@tmp.33 = private unnamed_addr constant [3 x i8] c".\0A\00"
@tmp.34 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.35 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.36 = private unnamed_addr constant [17 x i8] c"'s account pays \00"
@tmp.37 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.38 = private unnamed_addr constant [3 x i8] c"%f\00"
@tmp.39 = private unnamed_addr constant [43 x i8] c" interest and currently
has a balance of $\00"
```

```
@tmp.40 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.41 = private unnamed_addr constant [3 x i8] c"%d\00"
@tmp.42 = private unnamed_addr constant [3 x i8] c". \00"
@tmp.43 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.44 = private unnamed_addr constant [18 x i8] c"Account overdue!\0A\00"
@tmp.45 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.46 = private unnamed_addr constant [2 x i8] c"\0A\00"
@tmp.47 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.48 = private unnamed_addr constant [37 x i8] c"After a year's cumulative
interest, \00"
@tmp.49 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.50 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.51 = private unnamed_addr constant [25 x i8] c"'s new balance is now:
$\00"
@tmp.52 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.53 = private unnamed_addr constant [3 x i8] c"%d\00"
@tmp.54 = private unnamed_addr constant [3 x i8] c".\0A\00"
@tmp.55 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.56 = private unnamed_addr constant [27 x i8] c"Value cannot be
negative.\0A\00"
@tmp.57 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.58 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.59 = private unnamed_addr constant [25 x i8] c"'s new balance is now:
$\00"
@tmp.60 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.61 = private unnamed_addr constant [3 x i8] c"%d\00"
@tmp.62 = private unnamed_addr constant [3 x i8] c".\0A\00"
@tmp.63 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.64 = private unnamed_addr constant [31 x i8] c"Payment must be greater
than $\00"
@tmp.65 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.66 = private unnamed_addr constant [3 x i8] c"%d\00"
@tmp.67 = private unnamed_addr constant [3 x i8] c".\0A\00"
@tmp.68 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.69 = private unnamed_addr constant [27 x i8] c"Value cannot be
negative.\0A\00"
@tmp.70 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.71 = private unnamed_addr constant [28 x i8] c"Cannot withdraw this
much.\0A\00"
@tmp.72 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.73 = private unnamed_addr constant [3 x i8] c"%s\00"
```

```
@tmp.74 = private unnamed_addr constant [25 x i8] c"'s new balance is now:
$\00"
@tmp.75 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.76 = private unnamed_addr constant [3 x i8] c"%d\00"
@tmp.77 = private unnamed_addr constant [3 x i8] c".\0A\00"
@tmp.78 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp.79 = private unnamed_addr constant [9 x i8] c"Terrence\00"
@tmp.80 = private unnamed_addr constant [4 x i8] c"Aly\00"
@tmp.81 = private unnamed_addr constant [6 x i8] c"Clara\00"
@tmp.82 = private unnamed_addr constant [6 x i8] c"Holly\00"
@tmp.83 = private unnamed_addr constant [7 x i8] c"Darren\00"

declare i32 @printf(i8*, ...)

declare noalias i8* @malloc(i32)

define %Bank_Acct* @Bank_Acct(i8* %o, i32 %b, double %i) {
entry:
  %self = alloca %Bank_Acct
  %tmp = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null,
i32 1) to i32))
  %tmp1 = bitcast i8* %tmp to %Bank_Acct*
  %tmp2 = load %Bank_Acct, %Bank_Acct* %tmp1
  store %Bank_Acct %tmp2, %Bank_Acct* %self
  %owner = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0, i32 0
  store i8* %o, i8** %owner
  %balance = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0, i32
2
  store i32 %b, i32* %balance
  %interest = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0,
i32 1
  store double %i, double* %interest
  ret %Bank_Acct* %self
  unreachable
}

define void @Bank_Acct.acct_summary(%Bank_Acct* %self) {
entry:
  %owner = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0, i32 0
  %owner1 = load i8*, i8** %owner
  %print = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp, i32 0, i32 0), i8* %owner1)
```

```
  %print2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.2, i32 0, i32 0), i8* getelementptr inbounds ([18 x i8], [18 x
i8]* @tmp.1, i32 0, i32 0))
  %interest = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0,
i32 1
  %interest3 = load double, double* %interest
  %print4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.3, i32 0, i32 0), double %interest3)
  %print5 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.5, i32 0, i32 0), i8* getelementptr inbounds ([43 x i8], [43 x
i8]* @tmp.4, i32 0, i32 0))
  %balance = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0, i32
2
  %balance6 = load i32, i32* %balance
  %int_float_cast = sitofp i32 %balance6 to double
  %flt_divop = fdiv double %int_float_cast, 1.000000e+02
  %float_int_cast = fptosi double %flt_divop to i32
  %print7 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.6, i32 0, i32 0), i32 %float_int_cast)
  %print8 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.8, i32 0, i32 0), i8* getelementptr inbounds ([3 x i8], [3 x
i8]* @tmp.7, i32 0, i32 0))
  ret void
  unreachable
}

define void @Bank_Acct.calc_interest(%Bank_Acct* %self) {
entry:
  %balance = alloca double
  %balance1 = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0,
i32 2
  %balance2 = load i32, i32* %balance1
  %int_float_cast = sitofp i32 %balance2 to double
  store double %int_float_cast, double* %balance
  %balance3 = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0,
i32 2
  %balance4 = load double, double* %balance
  %interest = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0,
i32 1
  %interest5 = load double, double* %interest
  %flt_addtmp = fadd double 1.000000e+00, %interest5
  %flt_multop = fmul double %balance4, %flt_addtmp
```

```
  %float_int_cast = fptosi double %flt_multop to i32
  store i32 %float_int_cast, i32* %balance3
  %print = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.10, i32 0, i32 0), i8* getelementptr inbounds ([37 x i8], [37 x
i8]* @tmp.9, i32 0, i32 0))
  %owner = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0, i32 0
  %owner6 = load i8*, i8** %owner
  %print7 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.11, i32 0, i32 0), i8* %owner6)
  %print8 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.13, i32 0, i32 0), i8* getelementptr inbounds ([25 x i8], [25 x
i8]* @tmp.12, i32 0, i32 0))
  %balance9 = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0,
i32 2
  %balance10 = load i32, i32* %balance9
  %int_float_cast11 = sitofp i32 %balance10 to double
  %flt_divop = fdiv double %int_float_cast11, 1.000000e+02
  %float_int_cast12 = fptosi double %flt_divop to i32
  %print13 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x
i8], [3 x i8]* @tmp.14, i32 0, i32 0), i32 %float_int_cast12)
  %print14 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x
i8], [3 x i8]* @tmp.16, i32 0, i32 0), i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.15, i32 0, i32 0))
  ret void
  unreachable
}

define void @Bank_Acct.deposit(%Bank_Acct* %self, i32 %money) {
entry:
  br label %if

if:                                               ; preds = %entry
  %int_lesstmp = icmp slt i32 %money, 0
  br i1 %int_lesstmp, label %if_body, label %else_body

if_body:                                          ; preds = %if
  %print = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.18, i32 0, i32 0), i8* getelementptr inbounds ([27 x i8], [27 x
i8]* @tmp.17, i32 0, i32 0))
  br label %merge

else_body:                                        ; preds = %if
```

```
  %new_balance = alloca i32
  %balance = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0, i32
2
  %balance1 = load i32, i32* %balance
  %int_addtmp = add i32 %balance1, %money
  store i32 %int_addtmp, i32* %new_balance
  %balance2 = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0,
i32 2
  %new_balance3 = load i32, i32* %new_balance
  store i32 %new_balance3, i32* %balance2
  %owner = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0, i32 0
  %owner4 = load i8*, i8** %owner
  %print5 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.19, i32 0, i32 0), i8* %owner4)
  %print6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.21, i32 0, i32 0), i8* getelementptr inbounds ([25 x i8], [25 x
i8]* @tmp.20, i32 0, i32 0))
  %balance7 = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0,
i32 2
  %balance8 = load i32, i32* %balance7
  %int_float_cast = sitofp i32 %balance8 to double
  %flt_divop = fdiv double %int_float_cast, 1.000000e+02
  %float_int_cast = fptosi double %flt_divop to i32
  %print9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.22, i32 0, i32 0), i32 %float_int_cast)
  %print10 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x
i8], [3 x i8]* @tmp.24, i32 0, i32 0), i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.23, i32 0, i32 0))
  br label %merge

merge:                                            ; preds = %else_body,
%if_body
  ret void
  unreachable
}

define void @Bank_Acct.withdraw(%Bank_Acct* %self, i32 %money) {
entry:
  br label %if

if:                                               ; preds = %entry
  %int_lesstmp17 = icmp slt i32 %money, 0
```

```
  br i1 %int_lesstmp17, label %if_body, label %else_body

if_body:                                          ; preds = %if
  %print = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.26, i32 0, i32 0), i8* getelementptr inbounds ([27 x i8], [27 x
i8]* @tmp.25, i32 0, i32 0))
  br label %merge16

else_body:                                        ; preds = %if
  %new_balance = alloca i32
  %balance = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0, i32
2
  %balance1 = load i32, i32* %balance
  %int_subtmp = sub i32 %balance1, %money
  store i32 %int_subtmp, i32* %new_balance
  br label %if2

if2:                                              ; preds = %else_body
  %new_balance15 = load i32, i32* %new_balance
  %int_lesstmp = icmp slt i32 %new_balance15, 0
  br i1 %int_lesstmp, label %if_body3, label %else_body5

if_body3:                                         ; preds = %if2
  %print4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.28, i32 0, i32 0), i8* getelementptr inbounds ([28 x i8], [28 x
i8]* @tmp.27, i32 0, i32 0))
  br label %merge

else_body5:                                       ; preds = %if2
  %balance6 = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0,
i32 2
  %new_balance7 = load i32, i32* %new_balance
  store i32 %new_balance7, i32* %balance6
  %owner = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0, i32 0
  %owner8 = load i8*, i8** %owner
  %print9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.29, i32 0, i32 0), i8* %owner8)
  %print10 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x
i8], [3 x i8]* @tmp.31, i32 0, i32 0), i8* getelementptr inbounds ([25 x i8],
[25 x i8]* @tmp.30, i32 0, i32 0))
  %balance11 = getelementptr inbounds %Bank_Acct, %Bank_Acct* %self, i32 0,
i32 2
```

```
  %balance12 = load i32, i32* %balance11
  %int_float_cast = sitofp i32 %balance12 to double
  %flt_divop = fdiv double %int_float_cast, 1.000000e+02
  %float_int_cast = fptosi double %flt_divop to i32
  %print13 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x
i8], [3 x i8]* @tmp.32, i32 0, i32 0), i32 %float_int_cast)
  %print14 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x
i8], [3 x i8]* @tmp.34, i32 0, i32 0), i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.33, i32 0, i32 0))
  br label %merge

merge:                                            ; preds = %else_body5,
%if_body3
  br label %merge16

merge16:                                          ; preds = %merge, %if_body
  ret void
  unreachable
}

define %Credit_Card* @Credit_Card(i8* %o, i32 %b, double %i, i32 %mp) {
entry:
  %self = alloca %Credit_Card
  %tmp = call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null,
i32 1) to i32))
  %tmp1 = bitcast i8* %tmp to %Credit_Card*
  %tmp2 = load %Credit_Card, %Credit_Card* %tmp1
  store %Credit_Card %tmp2, %Credit_Card* %self
  %owner = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32 0,
i32 0
  store i8* %o, i8** %owner
  %balance = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32 0,
i32 3
  store i32 %b, i32* %balance
  %interest = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32
0, i32 2
  store double %i, double* %interest
  %minimum_payment = getelementptr inbounds %Credit_Card, %Credit_Card*
%self, i32 0, i32 1
  store i32 %mp, i32* %minimum_payment
  ret %Credit_Card* %self
  unreachable
```

```
}

define void @Credit_Card.acct_summary(%Credit_Card* %self) {
entry:
  %owner = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32 0,
i32 0
  %owner1 = load i8*, i8** %owner
  %print = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.35, i32 0, i32 0), i8* %owner1)
  %print2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.37, i32 0, i32 0), i8* getelementptr inbounds ([17 x i8], [17 x
i8]* @tmp.36, i32 0, i32 0))
  %interest = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32
0, i32 2
  %interest3 = load double, double* %interest
  %print4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.38, i32 0, i32 0), double %interest3)
  %print5 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.40, i32 0, i32 0), i8* getelementptr inbounds ([43 x i8], [43 x
i8]* @tmp.39, i32 0, i32 0))
  %balance = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32 0,
i32 3
  %balance6 = load i32, i32* %balance
  %int_float_cast = sitofp i32 %balance6 to double
  %flt_divop = fdiv double %int_float_cast, 1.000000e+02
  %float_int_cast = fptosi double %flt_divop to i32
  %print7 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.41, i32 0, i32 0), i32 %float_int_cast)
  %print8 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.43, i32 0, i32 0), i8* getelementptr inbounds ([3 x i8], [3 x
i8]* @tmp.42, i32 0, i32 0))
  br label %if

if:                                               ; preds = %entry
  %balance11 = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32
0, i32 3
  %balance12 = load i32, i32* %balance11
  %int_lesstmp = icmp slt i32 %balance12, 0
  br i1 %int_lesstmp, label %if_body, label %else_body

if_body:                                          ; preds = %if
```

```
  %print9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.45, i32 0, i32 0), i8* getelementptr inbounds ([18 x i8], [18 x
i8]* @tmp.44, i32 0, i32 0))
  br label %merge

else_body:                                        ; preds = %if
  %print10 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x
i8], [3 x i8]* @tmp.47, i32 0, i32 0), i8* getelementptr inbounds ([2 x i8],
[2 x i8]* @tmp.46, i32 0, i32 0))
  br label %merge

merge:                                            ; preds = %else_body,
%if_body
  ret void
  unreachable
}

define void @Credit_Card.calc_interest(%Credit_Card* %self) {
entry:
  %balance = alloca double
  %balance1 = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32
0, i32 3
  %balance2 = load i32, i32* %balance1
  %int_float_cast = sitofp i32 %balance2 to double
  store double %int_float_cast, double* %balance
  %balance3 = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32
0, i32 3
  %balance4 = load double, double* %balance
  %interest = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32
0, i32 2
  %interest5 = load double, double* %interest
  %flt_addtmp = fadd double 1.000000e+00, %interest5
  %flt_multop = fmul double %balance4, %flt_addtmp
  %float_int_cast = fptosi double %flt_multop to i32
  store i32 %float_int_cast, i32* %balance3
  %print = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.49, i32 0, i32 0), i8* getelementptr inbounds ([37 x i8], [37 x
i8]* @tmp.48, i32 0, i32 0))
  %owner = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32 0,
i32 0
  %owner6 = load i8*, i8** %owner
```

```
  %print7 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.50, i32 0, i32 0), i8* %owner6)
  %print8 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.52, i32 0, i32 0), i8* getelementptr inbounds ([25 x i8], [25 x
i8]* @tmp.51, i32 0, i32 0))
  %balance9 = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32
0, i32 3
  %balance10 = load i32, i32* %balance9
  %int_float_cast11 = sitofp i32 %balance10 to double
  %flt_divop = fdiv double %int_float_cast11, 1.000000e+02
  %float_int_cast12 = fptosi double %flt_divop to i32
  %print13 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x
i8], [3 x i8]* @tmp.53, i32 0, i32 0), i32 %float_int_cast12)
  %print14 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x
i8], [3 x i8]* @tmp.55, i32 0, i32 0), i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.54, i32 0, i32 0))
  ret void
  unreachable
}

define void @Credit_Card.deposit(%Credit_Card* %self, i32 %money) {
entry:
  br label %if

if:                                               ; preds = %entry
  %int_lesstmp = icmp slt i32 %money, 0
  br i1 %int_lesstmp, label %if_body, label %else_body

if_body:                                          ; preds = %if
  %print = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.57, i32 0, i32 0), i8* getelementptr inbounds ([27 x i8], [27 x
i8]* @tmp.56, i32 0, i32 0))
  br label %merge

else_body:                                        ; preds = %if
  %new_balance = alloca i32
  %balance = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32 0,
i32 3
  %balance1 = load i32, i32* %balance
  %int_addtmp = add i32 %balance1, %money
  store i32 %int_addtmp, i32* %new_balance
```

```
  %balance2 = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32
0, i32 3
  %new_balance3 = load i32, i32* %new_balance
  store i32 %new_balance3, i32* %balance2
  %owner = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32 0,
i32 0
  %owner4 = load i8*, i8** %owner
  %print5 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.58, i32 0, i32 0), i8* %owner4)
  %print6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.60, i32 0, i32 0), i8* getelementptr inbounds ([25 x i8], [25 x
i8]* @tmp.59, i32 0, i32 0))
  %balance7 = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32
0, i32 3
  %balance8 = load i32, i32* %balance7
  %int_float_cast = sitofp i32 %balance8 to double
  %flt_divop = fdiv double %int_float_cast, 1.000000e+02
  %float_int_cast = fptosi double %flt_divop to i32
  %print9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.61, i32 0, i32 0), i32 %float_int_cast)
  %print10 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x
i8], [3 x i8]* @tmp.63, i32 0, i32 0), i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.62, i32 0, i32 0))
  br label %merge

merge:                                            ; preds = %else_body,
%if_body
  ret void
  unreachable
}

define void @Credit_Card.pay(%Credit_Card* %self, i32 %money) {
entry:
  br label %if

if:                                               ; preds = %entry
  %minimum_payment7 = getelementptr inbounds %Credit_Card, %Credit_Card*
%self, i32 0, i32 1
  %minimum_payment8 = load i32, i32* %minimum_payment7
  %int_lesstmp = icmp slt i32 %money, %minimum_payment8
  br i1 %int_lesstmp, label %if_body, label %else_body
```

```
if_body:                                          ; preds = %if
  %print = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.65, i32 0, i32 0), i8* getelementptr inbounds ([31 x i8], [31 x
i8]* @tmp.64, i32 0, i32 0))
  %minimum_payment = getelementptr inbounds %Credit_Card, %Credit_Card*
%self, i32 0, i32 1
  %minimum_payment1 = load i32, i32* %minimum_payment
  %print2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.66, i32 0, i32 0), i32 %minimum_payment1)
  %print3 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.68, i32 0, i32 0), i8* getelementptr inbounds ([3 x i8], [3 x
i8]* @tmp.67, i32 0, i32 0))
  br label %merge

else_body:                                        ; preds = %if
  %new_balance = alloca i32
  %balance = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32 0,
i32 3
  %balance4 = load i32, i32* %balance
  %int_addtmp = add i32 %balance4, %money
  store i32 %int_addtmp, i32* %new_balance
  %balance5 = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32
0, i32 3
  %new_balance6 = load i32, i32* %new_balance
  store i32 %new_balance6, i32* %balance5
  br label %merge

merge:                                            ; preds = %else_body,
%if_body
  ret void
  unreachable
}

define void @Credit_Card.withdraw(%Credit_Card* %self, i32 %money) {
entry:
  br label %if

if:                                               ; preds = %entry
  %int_lesstmp17 = icmp slt i32 %money, 0
  br i1 %int_lesstmp17, label %if_body, label %else_body

if_body:                                          ; preds = %if
```

```
  %print = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.70, i32 0, i32 0), i8* getelementptr inbounds ([27 x i8], [27 x
i8]* @tmp.69, i32 0, i32 0))
  br label %merge16

else_body:                                        ; preds = %if
  %new_balance = alloca i32
  %balance = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32 0,
i32 3
  %balance1 = load i32, i32* %balance
  %int_subtmp = sub i32 %balance1, %money
  store i32 %int_subtmp, i32* %new_balance
  br label %if2

if2:                                              ; preds = %else_body
  %new_balance15 = load i32, i32* %new_balance
  %int_lesstmp = icmp slt i32 %new_balance15, 0
  br i1 %int_lesstmp, label %if_body3, label %else_body5

if_body3:                                         ; preds = %if2
  %print4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.72, i32 0, i32 0), i8* getelementptr inbounds ([28 x i8], [28 x
i8]* @tmp.71, i32 0, i32 0))
  br label %merge

else_body5:                                       ; preds = %if2
  %balance6 = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32
0, i32 3
  %new_balance7 = load i32, i32* %new_balance
  store i32 %new_balance7, i32* %balance6
  %owner = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32 0,
i32 0
  %owner8 = load i8*, i8** %owner
  %print9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.73, i32 0, i32 0), i8* %owner8)
  %print10 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x
i8], [3 x i8]* @tmp.75, i32 0, i32 0), i8* getelementptr inbounds ([25 x i8],
[25 x i8]* @tmp.74, i32 0, i32 0))
  %balance11 = getelementptr inbounds %Credit_Card, %Credit_Card* %self, i32
0, i32 3
  %balance12 = load i32, i32* %balance11
  %int_float_cast = sitofp i32 %balance12 to double
```

```
  %flt_divop = fdiv double %int_float_cast, 1.000000e+02
  %float_int_cast = fptosi double %flt_divop to i32
  %print13 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x
i8], [3 x i8]* @tmp.76, i32 0, i32 0), i32 %float_int_cast)
  %print14 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x
i8], [3 x i8]* @tmp.78, i32 0, i32 0), i8* getelementptr inbounds ([3 x i8],
[3 x i8]* @tmp.77, i32 0, i32 0))
  br label %merge

merge:                                            ; preds = %else_body5,
%if_body3
  br label %merge16

merge16:                                          ; preds = %merge, %if_body
  ret void
  unreachable
}

define i32 @main() {
entry:
  %acct0 = alloca %Bank_Acct
  %tmp = call %Bank_Acct* @Bank_Acct(i8* getelementptr inbounds ([9 x i8], [9
x i8]* @tmp.79, i32 0, i32 0), i32 50000, double 1.000000e-02)
  %tmp1 = load %Bank_Acct, %Bank_Acct* %tmp
  store %Bank_Acct %tmp1, %Bank_Acct* %acct0
  %acct1 = alloca %Bank_Acct
  %tmp2 = call %Bank_Acct* @Bank_Acct(i8* getelementptr inbounds ([4 x i8], [4
x i8]* @tmp.80, i32 0, i32 0), i32 50000, double 2.500000e-02)
  %tmp3 = load %Bank_Acct, %Bank_Acct* %tmp2
  store %Bank_Acct %tmp3, %Bank_Acct* %acct1
  %acct2 = alloca %Bank_Acct
  %tmp4 = call %Bank_Acct* @Bank_Acct(i8* getelementptr inbounds ([6 x i8], [6
x i8]* @tmp.81, i32 0, i32 0), i32 50000, double 5.000000e-02)
  %tmp5 = load %Bank_Acct, %Bank_Acct* %tmp4
  store %Bank_Acct %tmp5, %Bank_Acct* %acct2
  %card0 = alloca %Credit_Card
  %tmp6 = call %Credit_Card* @Credit_Card(i8* getelementptr inbounds ([6 x
i8], [6 x i8]* @tmp.82, i32 0, i32 0), i32 -20089, double 3.200000e-01, i32
24)
  %tmp7 = load %Credit_Card, %Credit_Card* %tmp6
  store %Credit_Card %tmp7, %Credit_Card* %card0
  %card1 = alloca %Credit_Card
```

```
  %tmp8 = call %Credit_Card* @Credit_Card(i8* getelementptr inbounds ([7 x
i8], [7 x i8]* @tmp.83, i32 0, i32 0), i32 -100034, double 2.700000e-01, i32
42)
  %tmp9 = load %Credit_Card, %Credit_Card* %tmp8
  store %Credit_Card %tmp9, %Credit_Card* %card1
  %accounts = alloca %Bank_Acct**
  %malloccall = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32 ptrtoint
(i1** getelementptr (i1*, i1** null, i32 1) to i32), i32 3), i32 1), i32
ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32)))
  %tmp10 = bitcast i8* %malloccall to %Bank_Acct***
  %tmp11 = bitcast %Bank_Acct*** %tmp10 to %Bank_Acct**
  %tmp12 = bitcast %Bank_Acct** %tmp11 to i32*
  store i32 add (i32 mul (i32 ptrtoint (i1** getelementptr (i1*, i1** null,
i32 1) to i32), i32 3), i32 1), i32* %tmp12
  store %Bank_Acct** %tmp11, %Bank_Acct*** %accounts
  %accounts13 = load %Bank_Acct**, %Bank_Acct*** %accounts
  %list_access = getelementptr %Bank_Acct*, %Bank_Acct** %accounts13, i32 1
  store %Bank_Acct* %acct0, %Bank_Acct** %list_access
  %accounts14 = load %Bank_Acct**, %Bank_Acct*** %accounts
  %list_access15 = getelementptr %Bank_Acct*, %Bank_Acct** %accounts14, i32 2
  store %Bank_Acct* %acct1, %Bank_Acct** %list_access15
  %accounts16 = load %Bank_Acct**, %Bank_Acct*** %accounts
  %list_access17 = getelementptr %Bank_Acct*, %Bank_Acct** %accounts16, i32 3
  store %Bank_Acct* %acct2, %Bank_Acct** %list_access17
  %cards = alloca %Credit_Card**
  %malloccall18 = tail call i8* @malloc(i32 mul (i32 add (i32 mul (i32
ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32), i32 2), i32 1),
i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32)))
  %tmp19 = bitcast i8* %malloccall18 to %Credit_Card***
  %tmp20 = bitcast %Credit_Card*** %tmp19 to %Credit_Card**
  %tmp21 = bitcast %Credit_Card** %tmp20 to i32*
  store i32 add (i32 mul (i32 ptrtoint (i1** getelementptr (i1*, i1** null,
i32 1) to i32), i32 2), i32 1), i32* %tmp21
  store %Credit_Card** %tmp20, %Credit_Card*** %cards
  %cards22 = load %Credit_Card**, %Credit_Card*** %cards
  %list_access23 = getelementptr %Credit_Card*, %Credit_Card** %cards22, i32 1
  store %Credit_Card* %card0, %Credit_Card** %list_access23
  %cards24 = load %Credit_Card**, %Credit_Card*** %cards
  %list_access25 = getelementptr %Credit_Card*, %Credit_Card** %cards24, i32 2
  store %Credit_Card* %card1, %Credit_Card** %list_access25
  %i = alloca i32
  store i32 0, i32* %i
```

```
  store i32 0, i32* %i
  br label %loop_cond
  br label %loop_step

loop_step:                                        ; preds = %loop_body, %entry
  %i26 = load i32, i32* %i
  %int_addtmp = add i32 %i26, 1
  store i32 %int_addtmp, i32* %i
  br label %loop_cond

loop_cond:                                        ; preds = %loop_step, %entry
  %i27 = load i32, i32* %i
  %int_lesstmp = icmp slt i32 %i27, 3
  br i1 %int_lesstmp, label %loop_body, label %loop_exit

loop_exit:                                        ; preds = %loop_cond
  store i32 0, i32* %i
  br label %loop_cond34
  br label %loop_step33

loop_step33:                                      ; preds = %loop_body32,
%loop_exit
  %i36 = load i32, i32* %i
  %int_addtmp37 = add i32 %i36, 1
  store i32 %int_addtmp37, i32* %i
  br label %loop_cond34

loop_cond34:                                      ; preds = %loop_step33,
%loop_exit
  %i38 = load i32, i32* %i
  %int_lesstmp39 = icmp slt i32 %i38, 3
  br i1 %int_lesstmp39, label %loop_body32, label %loop_exit35

loop_exit35:                                      ; preds = %loop_cond34
  store i32 0, i32* %i
  br label %loop_cond49
  br label %loop_step48

loop_step48:                                      ; preds = %loop_body47,
%loop_exit35
  %i51 = load i32, i32* %i
  %int_addtmp52 = add i32 %i51, 1
```

```
  store i32 %int_addtmp52, i32* %i
  br label %loop_cond49

loop_cond49:                                      ; preds = %loop_step48,
%loop_exit35
  %i53 = load i32, i32* %i
  %int_lesstmp54 = icmp slt i32 %i53, 2
  br i1 %int_lesstmp54, label %loop_body47, label %loop_exit50

loop_exit50:                                      ; preds = %loop_cond49
  ret i32 0
  unreachable

loop_body:                                        ; preds = %loop_cond
  %temp = alloca %Bank_Acct
  %accounts28 = load %Bank_Acct**, %Bank_Acct*** %accounts
  %i29 = load i32, i32* %i
  %list_index = add i32 %i29, 1
  %list_access30 = getelementptr %Bank_Acct*, %Bank_Acct** %accounts28, i32
%list_index
  %list_access_val = load %Bank_Acct*, %Bank_Acct** %list_access30
  %tmp31 = load %Bank_Acct, %Bank_Acct* %list_access_val
  store %Bank_Acct %tmp31, %Bank_Acct* %temp
  call void @Bank_Acct.acct_summary(%Bank_Acct* %temp)
  br label %loop_step

loop_body32:                                      ; preds = %loop_cond34
  %temp40 = alloca %Bank_Acct
  %accounts41 = load %Bank_Acct**, %Bank_Acct*** %accounts
  %i42 = load i32, i32* %i
  %list_index43 = add i32 %i42, 1
  %list_access44 = getelementptr %Bank_Acct*, %Bank_Acct** %accounts41, i32
%list_index43
  %list_access_val45 = load %Bank_Acct*, %Bank_Acct** %list_access44
  %tmp46 = load %Bank_Acct, %Bank_Acct* %list_access_val45
  store %Bank_Acct %tmp46, %Bank_Acct* %temp40
  call void @Bank_Acct.calc_interest(%Bank_Acct* %temp40)
  br label %loop_step33

loop_body47:                                      ; preds = %loop_cond49
  %temp55 = alloca %Credit_Card
  %cards56 = load %Credit_Card**, %Credit_Card*** %cards
```

```
  %i57 = load i32, i32* %i
  %list_index58 = add i32 %i57, 1
  %list_access59 = getelementptr %Credit_Card*, %Credit_Card** %cards56, i32
%list_index58
  %list_access_val60 = load %Credit_Card*, %Credit_Card** %list_access59
  %tmp61 = load %Credit_Card, %Credit_Card* %list_access_val60
  store %Credit_Card %tmp61, %Credit_Card* %temp55
  call void @Credit_Card.pay(%Credit_Card* %temp55, i32 252)
  call void @Credit_Card.acct_summary(%Credit_Card* %temp55)
  br label %loop_step48
}
```

test_integration_mergesort.dcf

```
doMergesort([int] a, int start, int end, [int] buffer) -> void {
      if (start < end) {
            int mid = start + (start - end) / 2;
            mid = start + (end - start) / 2;
            doMergesort(a, start, mid, buffer);
            doMergesort(a, mid + 1, end, buffer);
            merge(a, start, mid, end, buffer);
      }
}

merge([int] a, int start, int mid, int end, [int] buffer) -> void {
      int i = 0;
      int j = mid + 1;
      int k = start;

      for (i = start; i <= end; i = i + 1) {
            buffer[i] = a[i];
      }
      i = start;

      while (i <= mid and j <= end) {
            if (buffer[i] <= buffer[j]) {
                  a[k] = buffer[i];
                  i = i + 1;
            }
            else {
                  a[k] = buffer[j];
                  j = j + 1;
```

```
            }

            k = k + 1;
      }

      while (i <= mid) {
            a[k] = buffer[i];
            k = k + 1;
            i = i + 1;
      }

}

mergesort([int] a, int length) -> void {
      if (length < 2) {
            return;
      }

      [int] buffer = [int, length];
      doMergesort(a, 0, length, buffer);
}

print_array([int] a, int length) -> void {
      int i;
      for (i = 0; i < length; i = i + 1) {
            print_int(a[i]);

            if (i < length - 1) {
                  print_string(" ");
            }
      }

      print_string("\n");
}


main() -> int {
      int length = 20;
      [int] a = [int, length];

      int i;
      for (i = 0; i < length; i = i + 1) {
```

```
            a[i] = i % 5;
        }


        print_string("Unsorted: \n");
        print_array(a, length);

        print_string("Sorted: \n");
        mergesort(a, length);
        print_array(a, length);

        a[0] = 50;
        a[1] = 3;
        a[2] = -1;
        a[3] = -100;
        a[4] = 20;
        a[5] = 90;
        a[6] = 5;
        a[7] = 3;
        a[8] = 10;
        a[9] = 20;
        a[10] = 0;
        a[11] = -20;
        a[12] = -20;
        a[13] = 60;
        a[14] = 60;
        a[15] = 5;
        a[16] = 80;
        a[17] = 70;
        a[18] = 10;
        a[19] = 100;

        print_string("Unsorted: \n");
        print_array(a, length);

        print_string("Sorted: \n");
        mergesort(a, length);
        print_array(a, length);

        return 0;
}
```

test_integration_mergesort.ll

```
@tmp = private unnamed_addr constant [12 x i8] c"Unsorted: \0A\00"
@tmp1 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp2 = private unnamed_addr constant [10 x i8] c"Sorted: \0A\00"
@tmp3 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp4 = private unnamed_addr constant [12 x i8] c"Unsorted: \0A\00"
@tmp5 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp6 = private unnamed_addr constant [10 x i8] c"Sorted: \0A\00"
@tmp7 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp8 = private unnamed_addr constant [3 x i8] c"%d\00"
@tmp9 = private unnamed_addr constant [2 x i8] c" \00"
@tmp10 = private unnamed_addr constant [3 x i8] c"%s\00"
@tmp11 = private unnamed_addr constant [2 x i8] c"\0A\00"
@tmp12 = private unnamed_addr constant [3 x i8] c"%s\00"

declare i32 @printf(i8*, ...)

declare noalias i8* @malloc(i32)

define i32 @main() {
entry:
  %length = alloca i32
  store i32 20, i32* %length
  %a = alloca i32*
  %length1 = load i32* %length
  %tmp = mul i32 ptrtoint (i1** getelementptr (i1** null, i32 1) to
i32), %length1
  %tmp2 = add i32 %tmp, 1
  %mallocsize = mul i32 %tmp2, ptrtoint (i1** getelementptr (i1** null,
i32 1) to i32)
  %malloccall = tail call i8* @malloc(i32 %mallocsize)
  %tmp3 = bitcast i8* %malloccall to i32**
  %tmp4 = bitcast i32** %tmp3 to i32*
  store i32 %tmp2, i32* %tmp4
  store i32* %tmp4, i32** %a
  %i = alloca i32
  store i32 0, i32* %i
  br label %loop_cond
  br label %loop_step

loop_step:                                        ; preds = %loop_body,
%entry
```

```
  %i5 = load i32* %i
  %int_addtmp = add i32 %i5, 1
  store i32 %int_addtmp, i32* %i
  br label %loop_cond

loop_cond:                                        ; preds = %loop_step,
%entry
  %i6 = load i32* %i
  %length7 = load i32* %length
  %int_lesstmp = icmp slt i32 %i6, %length7
  br i1 %int_lesstmp, label %loop_body, label %loop_exit

loop_exit:                                        ; preds = %loop_cond
  %print = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([3
x i8]* @tmp1, i32 0, i32 0), i8* getelementptr inbounds ([12 x i8]*
@tmp, i32 0, i32 0))
  %a11 = load i32** %a
  %length12 = load i32* %length
  call void @print_array(i32* %a11, i32 %length12)
  %print13 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds
([3 x i8]* @tmp3, i32 0, i32 0), i8* getelementptr inbounds ([10 x i8]*
@tmp2, i32 0, i32 0))
  %a14 = load i32** %a
  %length15 = load i32* %length
  call void @mergesort(i32* %a14, i32 %length15)
  %a16 = load i32** %a
  %length17 = load i32* %length
  call void @print_array(i32* %a16, i32 %length17)
  %a18 = load i32** %a
  %list_access19 = getelementptr i32* %a18, i32 1
  store i32 50, i32* %list_access19
  %a20 = load i32** %a
  %list_access21 = getelementptr i32* %a20, i32 2
  store i32 3, i32* %list_access21
  %a22 = load i32** %a
  %list_access23 = getelementptr i32* %a22, i32 3
  store i32 -1, i32* %list_access23
  %a24 = load i32** %a
  %list_access25 = getelementptr i32* %a24, i32 4
  store i32 -100, i32* %list_access25
  %a26 = load i32** %a
  %list_access27 = getelementptr i32* %a26, i32 5
```

```
store i32 20, i32* %list_access27
%a28 = load i32** %a
%list_access29 = getelementptr i32* %a28, i32 6
store i32 90, i32* %list_access29
%a30 = load i32** %a
%list_access31 = getelementptr i32* %a30, i32 7
store i32 5, i32* %list_access31
%a32 = load i32** %a
%list_access33 = getelementptr i32* %a32, i32 8
store i32 3, i32* %list_access33
%a34 = load i32** %a
%list_access35 = getelementptr i32* %a34, i32 9
store i32 10, i32* %list_access35
%a36 = load i32** %a
%list_access37 = getelementptr i32* %a36, i32 10
store i32 20, i32* %list_access37
%a38 = load i32** %a
%list_access39 = getelementptr i32* %a38, i32 11
store i32 0, i32* %list_access39
%a40 = load i32** %a
%list_access41 = getelementptr i32* %a40, i32 12
store i32 -20, i32* %list_access41
%a42 = load i32** %a
%list_access43 = getelementptr i32* %a42, i32 13
store i32 -20, i32* %list_access43
%a44 = load i32** %a
%list_access45 = getelementptr i32* %a44, i32 14
store i32 60, i32* %list_access45
%a46 = load i32** %a
%list_access47 = getelementptr i32* %a46, i32 15
store i32 60, i32* %list_access47
%a48 = load i32** %a
%list_access49 = getelementptr i32* %a48, i32 16
store i32 5, i32* %list_access49
%a50 = load i32** %a
%list_access51 = getelementptr i32* %a50, i32 17
store i32 80, i32* %list_access51
%a52 = load i32** %a
%list_access53 = getelementptr i32* %a52, i32 18
store i32 70, i32* %list_access53
%a54 = load i32** %a
%list_access55 = getelementptr i32* %a54, i32 19
```

```
  store i32 10, i32* %list_access55
  %a56 = load i32** %a
  %list_access57 = getelementptr i32* %a56, i32 20
  store i32 100, i32* %list_access57
  %print58 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds
([3 x i8]* @tmp5, i32 0, i32 0), i8* getelementptr inbounds ([12 x i8]*
@tmp4, i32 0, i32 0))
  %a59 = load i32** %a
  %length60 = load i32* %length
  call void @print_array(i32* %a59, i32 %length60)
  %print61 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds
([3 x i8]* @tmp7, i32 0, i32 0), i8* getelementptr inbounds ([10 x i8]*
@tmp6, i32 0, i32 0))
  %a62 = load i32** %a
  %length63 = load i32* %length
  call void @mergesort(i32* %a62, i32 %length63)
  %a64 = load i32** %a
  %length65 = load i32* %length
  call void @print_array(i32* %a64, i32 %length65)
  ret i32 0
  unreachable

loop_body:                                        ; preds = %loop_cond
  %a8 = load i32** %a
  %i9 = load i32* %i
  %list_index = add i32 %i9, 1
  %list_access = getelementptr i32* %a8, i32 %list_index
  %i10 = load i32* %i
  %int_modop = srem i32 %i10, 5
  store i32 %int_modop, i32* %list_access
  br label %loop_step
}
define void @print_array(i32* %a, i32 %length) {
entry:
  %i = alloca i32
  store i32 0, i32* %i
  br label %loop_cond
  br label %loop_step

loop_step:                                        ; preds = %merge,
%entry
  %i1 = load i32* %i
```

```
  %int_addtmp = add i32 %i1, 1
  store i32 %int_addtmp, i32* %i
  br label %loop_cond

loop_cond:                                        ; preds = %loop_step,
%entry
  %i2 = load i32* %i
  %int_lesstmp = icmp slt i32 %i2, %length
  br i1 %int_lesstmp, label %loop_body, label %loop_exit

loop_exit:                                        ; preds = %loop_cond
  %print7 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([3
x i8]* @tmp12, i32 0, i32 0), i8* getelementptr inbounds ([2 x i8]*
@tmp11, i32 0, i32 0))
  ret void
  unreachable

loop_body:                                        ; preds = %loop_cond
  %i3 = load i32* %i
  %list_index = add i32 %i3, 1
  %list_access = getelementptr i32* %a, i32 %list_index
  %list_access_val = load i32* %list_access
  %print = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([3
x i8]* @tmp8, i32 0, i32 0), i32 %list_access_val)
  br label %if

if:                                               ; preds = %loop_body
  %i5 = load i32* %i
  %int_subtmp = sub i32 %length, 1
  %int_lesstmp6 = icmp slt i32 %i5, %int_subtmp
  br i1 %int_lesstmp6, label %if_body, label %else_body

if_body:                                          ; preds = %if
  %print4 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([3
x i8]* @tmp10, i32 0, i32 0), i8* getelementptr inbounds ([2 x i8]*
@tmp9, i32 0, i32 0))
  br label %merge

else_body:                                        ; preds = %if
  br label %merge

merge:                                            ; preds = %else_body,
```

```
%if_body
  br label %loop_step
}

define void @mergesort(i32* %a, i32 %length) {
entry:
  br label %if

if:                                               ; preds = %entry
  %int_lesstmp = icmp slt i32 %length, 2
  br i1 %int_lesstmp, label %if_body, label %else_body

if_body:                                          ; preds = %if
  ret void
  br label %merge

else_body:                                        ; preds = %if
  br label %merge

merge:                                            ; preds = %else_body,
%if_body
  %buffer = alloca i32*
  %tmp = mul i32 ptrtoint (i1** getelementptr (i1** null, i32 1) to
i32), %length
  %tmp1 = add i32 %tmp, 1
  %mallocsize = mul i32 %tmp1, ptrtoint (i1** getelementptr (i1** null,
i32 1) to i32)
  %malloccall = tail call i8* @malloc(i32 %mallocsize)
  %tmp2 = bitcast i8* %malloccall to i32**
  %tmp3 = bitcast i32** %tmp2 to i32*
  store i32 %tmp1, i32* %tmp3
  store i32* %tmp3, i32** %buffer
  %buffer4 = load i32** %buffer
  call void @doMergesort(i32* %a, i32 0, i32 %length, i32* %buffer4)
  ret void
  unreachable
}

define void @merge(i32* %a, i32 %start, i32 %mid, i32 %end, i32*
%buffer) {
entry:
  %i = alloca i32
```

```
  store i32 0, i32* %i
  %j = alloca i32
  %int_addtmp = add i32 %mid, 1
  store i32 %int_addtmp, i32* %j
  %k = alloca i32
  store i32 %start, i32* %k
  store i32 %start, i32* %i
  br label %loop_cond
  br label %loop_step

loop_step:                                        ; preds = %loop_body,
%entry
  %i1 = load i32* %i
  %int_addtmp2 = add i32 %i1, 1
  store i32 %int_addtmp2, i32* %i
  br label %loop_cond

loop_cond:                                        ; preds = %loop_step,
%entry
  %i3 = load i32* %i
  %int_leqtmp = icmp sle i32 %i3, %end
  br i1 %int_leqtmp, label %loop_body, label %loop_exit

loop_exit:                                        ; preds = %loop_cond
  store i32 %start, i32* %i
  br label %loop_cond10
  br label %loop_step9

loop_step9:                                       ; preds = %merge,
%loop_exit
  br label %loop_cond10

loop_cond10:                                      ; preds = %loop_step9,
%loop_exit
  %i12 = load i32* %i
  %int_leqtmp13 = icmp sle i32 %i12, %mid
  %j14 = load i32* %j
  %int_leqtmp15 = icmp sle i32 %j14, %end
  %int_andtmp = and i1 %int_leqtmp13, %int_leqtmp15
  br i1 %int_andtmp, label %loop_body8, label %loop_exit11

loop_exit11:                                      ; preds = %loop_cond10
```

```
  br label %loop_cond47
  br label %loop_step46

loop_step46:                                      ; preds =
%loop_body45, %loop_exit11
  br label %loop_cond47

loop_cond47:                                      ; preds =
%loop_step46, %loop_exit11
  %i49 = load i32* %i
  %int_leqtmp50 = icmp sle i32 %i49, %mid
  br i1 %int_leqtmp50, label %loop_body45, label %loop_exit48

loop_exit48:                                      ; preds = %loop_cond47
  ret void
  unreachable

loop_body:                                        ; preds = %loop_cond
  %i4 = load i32* %i
  %list_index = add i32 %i4, 1
  %list_access = getelementptr i32* %buffer, i32 %list_index
  %i5 = load i32* %i
  %list_index6 = add i32 %i5, 1
  %list_access7 = getelementptr i32* %a, i32 %list_index6
  %list_access_val = load i32* %list_access7
  store i32 %list_access_val, i32* %list_access
  br label %loop_step

loop_body8:                                       ; preds = %loop_cond10
  br label %if

if:                                               ; preds = %loop_body8
  %i34 = load i32* %i
  %list_index35 = add i32 %i34, 1
  %list_access36 = getelementptr i32* %buffer, i32 %list_index35
  %list_access_val37 = load i32* %list_access36
  %j38 = load i32* %j
  %list_index39 = add i32 %j38, 1
  %list_access40 = getelementptr i32* %buffer, i32 %list_index39
  %list_access_val41 = load i32* %list_access40
  %int_leqtmp42 = icmp sle i32 %list_access_val37, %list_access_val41
  br i1 %int_leqtmp42, label %if_body, label %else_body
```

```
if_body:                                          ; preds = %if
  %k16 = load i32* %k
  %list_index17 = add i32 %k16, 1
  %list_access18 = getelementptr i32* %a, i32 %list_index17
  %i19 = load i32* %i
  %list_index20 = add i32 %i19, 1
  %list_access21 = getelementptr i32* %buffer, i32 %list_index20
  %list_access_val22 = load i32* %list_access21
  store i32 %list_access_val22, i32* %list_access18
  %i23 = load i32* %i
  %int_addtmp24 = add i32 %i23, 1
  store i32 %int_addtmp24, i32* %i
  br label %merge

else_body:                                        ; preds = %if
  %k25 = load i32* %k
  %list_index26 = add i32 %k25, 1
  %list_access27 = getelementptr i32* %a, i32 %list_index26
  %j28 = load i32* %j
  %list_index29 = add i32 %j28, 1
  %list_access30 = getelementptr i32* %buffer, i32 %list_index29
  %list_access_val31 = load i32* %list_access30
  store i32 %list_access_val31, i32* %list_access27
  %j32 = load i32* %j
  %int_addtmp33 = add i32 %j32, 1
  store i32 %int_addtmp33, i32* %j
  br label %merge

merge:                                            ; preds = %else_body,
%if_body
  %k43 = load i32* %k
  %int_addtmp44 = add i32 %k43, 1
  store i32 %int_addtmp44, i32* %k
  br label %loop_step9

loop_body45:                                      ; preds = %loop_cond47
  %k51 = load i32* %k
  %list_index52 = add i32 %k51, 1
  %list_access53 = getelementptr i32* %a, i32 %list_index52
  %i54 = load i32* %i
  %list_index55 = add i32 %i54, 1
```

```
  %list_access56 = getelementptr i32* %buffer, i32 %list_index55
  %list_access_val57 = load i32* %list_access56
  store i32 %list_access_val57, i32* %list_access53
  %k58 = load i32* %k
  %int_addtmp59 = add i32 %k58, 1
  store i32 %int_addtmp59, i32* %k
  %i60 = load i32* %i
  %int_addtmp61 = add i32 %i60, 1
  store i32 %int_addtmp61, i32* %i
  br label %loop_step46
}

define void @doMergesort(i32* %a, i32 %start, i32 %end, i32* %buffer) {
entry:
  br label %if

if:                                               ; preds = %entry
  %int_lesstmp = icmp slt i32 %start, %end
  br i1 %int_lesstmp, label %if_body, label %else_body

if_body:                                          ; preds = %if
  %mid = alloca i32
  %int_subtmp = sub i32 %start, %end
  %int_divop = sdiv i32 %int_subtmp, 2
  %int_addtmp = add i32 %start, %int_divop
  store i32 %int_addtmp, i32* %mid
  %int_subtmp1 = sub i32 %end, %start
  %int_divop2 = sdiv i32 %int_subtmp1, 2
  %int_addtmp3 = add i32 %start, %int_divop2
  store i32 %int_addtmp3, i32* %mid
  %mid4 = load i32* %mid
  call void @doMergesort(i32* %a, i32 %start, i32 %mid4, i32* %buffer)
  %mid5 = load i32* %mid
  %int_addtmp6 = add i32 %mid5, 1
  call void @doMergesort(i32* %a, i32 %int_addtmp6, i32 %end, i32*
%buffer)
  %mid7 = load i32* %mid
  call void @merge(i32* %a, i32 %start, i32 %mid7, i32 %end, i32*
%buffer)
  br label %merge

else_body:                                        ; preds = %if
```

```
    br label %merge

  merge:                                            ; preds = %else_body,
  %if_body
    ret void
    unreachable
  }
```

Automation

Automated testing was done via the bash script `testall.sh`, given below.

```
#!/bin/sh

# Regression testing script for DECAF
# 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 LLVM compiler
LLC="llc"

# Path to the C compiler
CC="gcc"

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

file_ext="dcf"

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
```

```
error=0
globalerror=0

keep=0

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

SignalError() {
    if [ $error -eq 0 ] ; then
     echo "FAILED"
     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"
     echo "FAILED $1 differs from $2" 1>&2
    }
}

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

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

Check() {
    error=0
    basename=$(echo $1 | sed 's/.${file_ext}//' | cut -f 1 -d '.')
    reffile=$(echo $1 | sed 's/.${file_ext}$//')
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "${basename}..."

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

    generatedfiles=""

    generatedfiles="${generatedfiles} ${basename}.ll ${basename}.s
${basename} ${basename}.out ${basename}.diff" &&
    Run "${DECAF} -l" "<" $1 "| tail -n+3" ">" "${basename}.ll" &&
    Run "${LLC}" "${basename}.ll" ">" "${basename}.s" &&
    Run "${CC}" "-o" "${basename}" "${basename}.s" &&
    Run "./${basename}" ">" "${basename}.out" &&
    Compare ${basename}.cmp ${basename}.out ${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 "OK"
     echo "###### SUCCESS" 1>&2
```

```
    else
      echo "###### FAILED" 1>&2
      globalerror=${error}
    fi
}

CheckFail() {
    error=0
    basename=$(echo $1 | sed 's/.${file_ext}//' | cut -f 1 -d '.')
    reffile=`echo $1 | sed 's/.${file_ext}$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "${basename}..."

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

    generatedfiles=""

    generatedfiles="${generatedfiles} ${basename}.out ${basename}.diff"
&&
    RunFail "${DECAF}" "<" $1 "2>" "${basename}.out" ">>" ${globallog} &&
    Compare ${basename}.cmp ${basename}.out ${basename}.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
      if [ $keep -eq 0 ] ; then
          rm -f $generatedfiles
      fi
      echo "OK"
      echo "###### SUCCESS" 1>&2
    else
      echo "###### FAILED" 1>&2
      globalerror=$error
    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 [ $# -ge 1 ]
then
    files=$@
else
    files="tests/test_*.${file_ext} tests/fail_*.${file_ext}"
fi

for file in $files
do
    case $file in
     *test_*)
          Check $file 2>> $globallog
          ;;
     *fail_*)
          CheckFail $file 2>> $globallog
          ;;
     *)
          echo "unknown file type $file"
          globalerror=1
          ;;
    esac
done
```

```
exit $globalerror
```

Test Suite

The following programs were used to test particular features of the language. Each unit test is intended to test a single part of the language, usually at the time when such functionality is implemented. The unit tests are listed below.

test_assign_1.dcf

```
main() -> int {
        int x;
        x = 1;
        print_int(x);
        return 0;
}
```

test_assign_2.dcf

```
main() -> int {
        int x;
        x = 3 + 30 / (5 * 3);
        x = -1;
        x = 1 - -1;
        int y;
        y = 3 * 5;
        x = y - 2 * 5;
        print_int(x);
        print_string("\n");
        print_int(y);
        return 0;
}
```

test_assign_3.dcf

```
main() -> int {
        int x;
        x = 5;
        int y;
        y = 3;
        x = x * y;
        print_int(x);
        print_string("\n");
```

```
        print_int(y);
        return 0;
}
```

test_assign_4.dcf

```
main() -> int {
        int x = 3;
        int y = x = 5;
        print_int(x);
        print_string("\n");
        print_int(y);
        return 0;
}
```

test_cast_1.dcf

```
main() -> int {
        // Test bool conversions
        bool b = <bool> true;
        int i = <int> true;
        float f = <float> true;
        char c = <char> ((<int> true) + 64);

        print_int(i);
        print_string("\n");
        print_float(f);
        print_string("\n");
        print_char(c);

        return 0;
}
```

test_cast_2.dcf

```
main() -> int {
        float f = <float> 65.0;
        bool b = <bool> 65.0;
        int i = <int> 65.0;
        char c = <char> 65.0;

        print_int(i);
        print_string("\n");
        print_char(c);
        return 0;
```

```
}
```

test_cast_3.dcf

```
main() -> int {
        int x = <int> 5;
        bool b = <bool> 5;
        float f = <float> 5;

        print_int(x);
        print_string("\n");
        print_float(f);

        return 0;
}
```

test_cast_4.dcf

```
/* More extensive testing of int -> bool conversion */
main() -> int {
        int i;
        if (not <bool>i) {
                print_string("yes\n");
        }

        for (i = 1; i < 10; i = i + 1) {
                if (not <bool>i) {
                        print_string("no\n");
                }
                else {
                        print_string("yes\n");
                }
        }

        for (i = -1; i > -10; i = i - 1) {
                if (not <bool>i) {
                        print_string("no\n");
                }
                else {
                        print_string("yes\n");
                }
        }
        return 0;
}
```

test_cast_5.dcf

```
main() -> int {
        // Test char conversion
        char c = <char> 65;
        int i = <int> c;
        float f = <float> c;

        print_char(c);
        print_string("\n");
        print_int(i);
        print_string("\n");
        print_float(f);

        return 0;
}
```

test_class_1.dcf

```
class Animal {
        string name;
        int legs;
        bool tail;

        Animal(string n, int l, bool t) -> Animal {
                self.name = n;
                self.legs = l;
                self.tail = t;
        }

        talk() -> void {
                print_string("hi");
        }
}

main() -> int {
        Animal cat = Animal("Amy", 3, true);
        string old_name = cat.name;
        print_string(old_name);
        cat.name = "Bob";
        print_string(cat.name);
        cat.talk();
        return 0;
}
```

test_class_2.dcf

```
class Game {
        string title;
        float price;

        Game(string t, float p) -> Game {
                self.title = t;
                self.price = p;
        }

        compare(Game other) -> void {
                //fix once concat works
                print_string("Between ");
                print_string(self.title);
                print_string(" and ");
                print_string(other.title);
                print_string(",\n");
                if (self.price > other.price) {
                        print_string(self.title);
                        print_string(" is worth more at $");
                        print_float(self.price);
                        print_string(".\n");
                } elseif (self.price < other.price) {
                        print_string(other.title);
                        print_string(" is worth more at $");
                        print_float(other.price);
                        print_string(".\n");
                } else {
                        print_string("No game is worth more.\n");
                }
        }
}

main() -> int {
        Game chess = Game("Chess", 14.00);
        Game checkers = Game("Checkers", 14.00);
        Game sorry = Game("Sorry", 10.50);
        Game pandemic = Game("Pandemic: Legacy", 40.75);
        sorry.compare(pandemic);
        chess.compare(checkers);
        return 0;
}
```

test_class_3.dcf

```
class Person {
        string name;
        int age;

        Person(string n, int a) -> Person {
                self.name = n;
                self.age = a;
        }

        talk() -> void {
                print_string(self.name);
                print_string(" says hello!\n");
                return;
        }
}

class Student extends Person {
        float gpa;
        int id;

        Student(string n, int a, float g, int i) -> Student {
                self.name = n;
                self.age = a;
                self.gpa = g;
                self.id = i;
        }

        talk() -> void {
                print_string(self.name);
                print_string(" (Student #");
                print_int(self.id);
                print_string(") says hello!\n");
                return;
        }
}

main() -> int {
        Person a = Person("Andy", 34);
        Student b = Student("Grace", 20, 3.3, 54321);
        a.talk();
        b.talk();
```

```
        return 0;
}
```

test_class_4.dcf

```
class Animal {
        int num_feet;
        string name;

        Animal(int ft, string n) -> Animal {
                self.num_feet = ft;
                self.name = n;
        }

        summary() -> void {
                print_string(self.name);
                print_string(" has ");
                print_int(self.num_feet);
                print_string(" feet.\n");
        }
}

class Mammal extends Animal{
        int litter_size;

        Mammal(int ft, string n, int l) -> Mammal {
                self.num_feet = ft;
                self.name = n;
                self.litter_size = l;
        }

}

class Capybara extends Mammal {
        bool cute;

        Capybara(int ft, string n, int l, bool c) -> Capybara {
                self.num_feet = ft;
                self.name = n;
                self.litter_size = l;
                self.cute = c;
        }
```

```
        is_cute() -> void {
                print_string(self.name);
                print_string(" has ");
                print_int(self.num_feet);
                print_string(" feet and is ");
                if (not self.cute) {
                        print_string("not ");
                }
                print_string("cute.\n");
        }

        compare(Capybara c2) -> int {
                if (self.num_feet === c2.num_feet) {
                        return 0;
                } elseif (self.num_feet > c2.num_feet) {
                        return 1;
                } else {
                        return -1;
                }
        }
}

main() -> int {
        Capybara connie = Capybara(4, "Connie", 50, false);
        Capybara connie2 = Capybara(4, "Connie2", 50, false);
        connie.is_cute();
        int res = connie.compare(connie2);
        if (res === 0) {
                print_string("Connie and connie2 are equally (not)
cute.\n");
        }
        connie.summary();
        return 0;
}
```

test_continue_1.dcf

```
main() -> int {
        int x;
        while (x < 5) {
                x = x + 1;
                if (true) {
                        print_string("foo");
```

```
                        continue;
                }
                print_string("bar");
        }
        return 0;
}
```

test_continue_2.dcf

```
main() -> int {
        int i;
        for (i = 0; i < 5; i = i + 1) {
                int j;
                for (j = 0; j < 5; j = j + 1) {
                        print_string("foo");
                        break;
                }
                print_string("bar");
        }
        return 0;
}
```

test_declare_1.dcf

```
main() -> int {
        int x;
        return 0;
}
```

test_declare_2.dcf

```
main() -> int {
        int x = 5;
        return 0;
}
```

test_for_1.dcf

```
main() -> int {
        int x;
        for (x = 0; x < 10; x = x + 1) {
                print_string("hi");
        }
        print_string("done");
        return 0;
}
```

test_for_2.dcf

```
main() -> int {
        int i;
        for (i = 0; i < 5; i = i + 1) {
                int j;
                for (j = 0; j < 2; j = j + 1) {
                        print_string("hi");
                }
        }
        return 0;
}
```

test_for_while.dcf

```
main() -> int {
        int i;
        int j;
        for (i = 0; i < 3; i = i + 1) {
                j = 0;
                while (j < i) {
                        print_string("ij\t");
                        j = j + 1;
                }
        }
        return 0;
}
```

test_func_call_1.dcf

```
foo(int i) -> int {
        int x = i + 1;
        return x;
}

main() -> int {
        int y = foo(1);
        if (y === 2) {
                print_string("I can do math");
        }
        return 0;
}
```

test_func_call_2.dcf

```
main() -> int {
```

```
        int x = 12;
        while (x > 9) {
                x = minus(x);
                print_string("minus\n");
        }
        while (x < 10) {
                x = plus(x);
                print_string("plus\n");
        }
        return 0;
}

minus(int i) -> int {
        return i - 1;
}

plus(int i) -> int {
        return i + 1;
}
```

test_func_call_3.dcf

```
// recursive exponent method
exp(int base, int power) -> int {
        if (power === 0) {
                return 1;
        }
        elseif (power === 1) {
                return base;
        }
        else {
                return base * exp(base, power - 1);
        }
}

main() -> int {
        int a;
        a = exp(3, 2);
        int b = exp(2, 4);
        print_string("a: ");
        print_int(a);
        print_string("\tb: ");
        print_int(b);
```

```
        return 0;
}
```

test_gcd.dcf

```
gcd(int a, int b) -> int {
        if (b === 0) {
                return a;
        }
        return gcd(b, a%b);
}

main() -> int {
        int a = 100;
        int b = 20;
        int c = 49;
        int e = 29;
        if (gcd(a, b) === 20 and gcd(c, e) === 1) {
                print_string("I can do gcd -w-\n");
        }
        return 0;
}
```

test_global_hello_world.dcf

```
main() -> int {
    print_string("Hello world!");
    return 0;
}
```

test_hello_world.dcf

```
class Main {
        main() -> int {
                print_string("Hello world!");
                return 0;
        }
}
```

test_if_1.dcf

```
main() -> int {
        // Damn I'm so good at writing these test cases
        int x;
```

```
        if (false) {
                print_string("foo");
                x = 1;
        }
        elseif (true) {
                print_string("bar");
        }
        elseif (false) {
                print_string("bat");
        }
        else {
                print_string("baz");
        }
        return 0;
}
```

test_if_2.dcf

```
main() -> int {
        if (false) {
                print_string("foo");
        }
        else {
                print_string("bar");
        }

        if (true) {
                print_string("bat");
        }
        return 0;
}
```

test_if_3.dcf

```
main() -> int {
        if (true) {
                if (false) {
                        print_string("foo");
                }
                elseif (true) {
                        print_string("bar");
                }
                print_string("bat");
        }
```

```
        elseif (true) {
                print_string("baz");
        }
        return 0;
}
```

test_if_4.dcf

```
main() -> int {
        int x = 5;

        if (x < 5) {
                print_string("foo");
        }
        elseif (x === 5) {
                print_string("bar");
        }
        elseif (x > 3) {
                print_string("bat");
        }
        return 0;
}
```

test_integration_arraylist.dcf

```
class ArrayList {
   // Buffer
   [int] data;
   // Current size
   int size;
   // Capacity
   int capacity;

   /* Creates the list */
   ArrayList(int capacity) -> ArrayList {
         self.size = 0;
         if (capacity === 0) {
               self.capacity = 10;
         }
         else {
               self.capacity = capacity;
         }
         self.data = [int, self.capacity];
```

```
	int i;
	for (i = 0; i < self.capacity; i = i + 1) {
		self.data[i] = 0;
	}
}

/* Fills the list up to current size with dummy elements */
fill(int newSize) -> void {
	int i;
	for (i = 0; i < newSize; i = i + 1) {
		self.data[i] = i;
	}
	self.size = newSize;
}

/* Gets the element at the index */
get(int index) -> int {
	if (not (0 <= index and index <= self.size)) {
		print_string("Error: index out of bounds\n");
		return -1;
	}
	return self.data[index];
}

/* Set 1 element in list */
set(int index, int val) -> void {
	if (not (0 <= index and index <= self.size)) {
		print_string("Error: index out of bounds\n");
		return;
	}
	self.data[index] = val;
}

/* Add 1 element to end of list */
add(int val) -> void {
	if (self.size >= self.capacity - 1) {
		self.capacity = self.capacity * 2;
		[int] newData = [int, self.capacity];
		int i;
		for (i = 0; i < self.capacity; i = i + 1) {
			newData[i] = self.data[i];
		}
```

```
            self.data = newData;
        }

        self.data[self.size] = val;
        self.size = self.size + 1;
    }

    /* Removes at index */
    remove(int index) -> void {
        if (not (0 <= index and index <= self.size)) {
            print_string("Error: index out of bounds\n");
            return;
        }

        // Shift over all elements
        int i;
        for (i = index; i < self.size - 1; i = i + 1) {
            self.data[i] = self.data[i + 1];
        }

        self.size = self.size - 1;
    }

    print() -> void {
        int i;
        for (i = 0; i < self.size; i = i + 1) {
            print_int(self.data[i]);

            if (i !== self.size - 1) {
                print_string(" ");
            }
        }
        print_string("\n");
    }
}

main() -> int {
    int len = 4;
    ArrayList lst1 = ArrayList(len);
    // Test fill()
    lst1.fill(lst1.capacity);
```

```
// Test set
int i;
for (i = 0; i < lst1.size; i = i + 1) {
      lst1.set(i, i * 100);
}

// Test print()
// 0 100 200 300
lst1.print();

ArrayList lst2 = ArrayList(len);

// Test set()
for (i = 0; i < lst1.size; i = i + 1) {
      if (i >= lst2.size) {
            lst2.add(0);
      }
      lst2.set(i, lst1.get(i) / 100);
}

// 0 1 2 3
lst2.print();

// Test add()
for (i = 0; i < 10; i = i + 1) {
      lst2.add(i * 100);
}

// 0 1 2 3 0 100 200 300 400 500 600 700 800 900
lst2.print();

ArrayList lst3 = ArrayList(lst2.size / 2);
int end = lst3.capacity;
for (i = 0; i < end; i = i + 1) {
      lst3.add(i * 2);
}

// 0 2 4 6 8 10 12
lst3.print();

// Test remove()
for (i = lst3.size - 1; i >= 0; i = i - 1) {
```

```
            int index = lst3.get(i);
            lst2.remove(index);
        }

        // 1 3 100 300 500 700 900
        lst2.print();

        return 0;
    }
```

tests/test_integration_bankacct.dcf

```
class Bank_Acct {
   string owner;
   float balance;
   float interest;

   Bank_Acct(string o, float b, float i) -> Bank_Acct {
         self.owner = o;
         self.balance = b;
         self.interest = i;
   }

   acct_summary() -> void {
         print_string(self.owner);
         print_string("\'s account earns ");
         print_float(self.interest);
         print_string(" interest and currently has a balance of $");
         print_float(self.balance);
         print_string(".\n");
   }

   deposit(float money) -> void {
         if (money < 0.0) {
               print_string("Value cannot be negative.\n");
         } else {
               float new_balance = self.balance + money;
               self.balance = new_balance;
               print_string(self.owner);
               print_string("\'s new balance is now: $");
               print_float(self.balance);
               print_string(".\n");
         }
   }

   withdraw(float money) -> void {
         if (money < 0.0) {
               print_string("Value cannot be negative.\n");
         } else {
               float new_balance = self.balance - money;
               if (new_balance < 0.0) {
                     print_string("Cannot withdraw this much.\n");
```

```
                } else {
                        self.balance = new_balance;
                        print_string(self.owner);
                        print_string("\'s new balance is now: $");
                        print_float(self.balance);
                        print_string(".\n");
                }
        }
    }

    calc_interest() -> void {
        self.balance = self.balance * (1.0 + self.interest);
        print_string("After interest, ");
        print_string(self.owner);
        print_string("\'s new balance is now: $");
        print_float(self.balance);
        print_string(".\n");
    }
}

// when balance on card is negative, indicates amount owed
class Credit_Card extends Bank_Acct {
    float minimum_payment;

    Credit_Card (string o, float b, float i, float mp) -> Credit_Card
{
        self.owner = o;
        self.balance = b;
        self.interest = i;
        self.minimum_payment = mp;
    }

    pay(float money) -> void {
        if (money < self.minimum_payment) {
            print_string("Payment must be greater than $");
            print_float(self.minimum_payment);
            print_string(".\n");
        } else {
            float new_balance = self.balance + money;
            self.balance = new_balance;
            print_string(self.owner);
            print_string("\'s new balance is now: $");
```

```
            print_float(self.balance);
            print_string(".\n");
        }
    }
}

main() -> int {
    Bank_Acct acct0 = Bank_Acct("Terrence", 500.00, 0.01);
    Bank_Acct acct1 = Bank_Acct("Aly", 500.00, 0.025);
    Bank_Acct acct2 = Bank_Acct("Clara", 500.00, 0.05);

    Credit_Card card0 = Credit_Card("Holly", -200.89, 0.12, 24.32);
    Credit_Card card1 = Credit_Card("Darren", -1000.34, 0.10, 40.13);

    [Bank_Acct] accounts = [Bank_Acct, 3];
    accounts[0] = acct0;
    accounts[1] = acct1;
    accounts[2] = acct2;
    [Credit_Card] cards = [Credit_Card, 2];
    cards[0] = card0;
    cards[1] = card1;

    int i = 0;
    for (i = 0; i < 3; i = i + 1) {
        Bank_Acct temp = accounts[i];
        temp.acct_summary();
    }
    for (i = 0; i < 3; i = i + 1) {
        Bank_Acct temp = accounts[i];
        temp.calc_interest();
    }
    for (i = 0; i < 2; i = i + 1) {
        Credit_Card temp = cards[i];
        temp.pay(25.22);
    }
    return 0;
}
```

tests/test_integration_mergesort.dcf

```
doMergesort([int] a, int start, int end, [int] buffer) -> void {
    if (start < end) {
        int mid = start + (start - end) / 2;
```

```
        mid = start + (end - start) / 2;
        doMergesort(a, start, mid, buffer);
        doMergesort(a, mid + 1, end, buffer);
        merge(a, start, mid, end, buffer);
    }
}

merge([int] a, int start, int mid, int end, [int] buffer) -> void {
    int i = 0;
    int j = mid + 1;
    int k = start;

    for (i = start; i <= end; i = i + 1) {
        buffer[i] = a[i];
    }
    i = start;

    while (i <= mid and j <= end) {
        if (buffer[i] <= buffer[j]) {
            a[k] = buffer[i];
            i = i + 1;
        }
        else {
            a[k] = buffer[j];
            j = j + 1;
        }

        k = k + 1;
    }

    while (i <= mid) {
        a[k] = buffer[i];
        k = k + 1;
        i = i + 1;
    }

}

mergesort([int] a, int length) -> void {
    if (length < 2) {
        return;
    }
```

```
   [int] buffer = [int, length];
   doMergesort(a, 0, length, buffer);
}

print_array([int] a, int length) -> void {
   int i;
   for (i = 0; i < length; i = i + 1) {
         print_int(a[i]);

         if (i < length - 1) {
               print_string(" ");
         }
   }

   print_string("\n");
}


main() -> int {
   int length = 20;
   [int] a = [int, length];

   int i;
   for (i = 0; i < length; i = i + 1) {
         a[i] = i % 5;
   }


   print_string("Unsorted: \n");
   print_array(a, length);

   print_string("Sorted: \n");
   mergesort(a, length);
   print_array(a, length);

   a[0] = 50;
   a[1] = 3;
   a[2] = -1;
   a[3] = -100;
   a[4] = 20;
   a[5] = 90;
```

```
    a[6] = 5;
    a[7] = 3;
    a[8] = 10;
    a[9] = 20;
    a[10] = 0;
    a[11] = -20;
    a[12] = -20;
    a[13] = 60;
    a[14] = 60;
    a[15] = 5;
    a[16] = 80;
    a[17] = 70;
    a[18] = 10;
    a[19] = 100;

    print_string("Unsorted: \n");
    print_array(a, length);

    print_string("Sorted: \n");
    mergesort(a, length);
    print_array(a, length);

    return 0;
}
```

test_lits_1.dcf

```
main() -> int {
        print_int(5 + 3);
        print_string("\n");
        print_int(5 * 3);
        print_string("\n");
        print_float(5.0 / 3.0);
        print_string("\n");
        print_int(5 % 3);
        print_string("\n");
        print_char('c');
        print_string("\n");
        print_string("s");
        return 0;
}
```

test_return_1.dcf

```
main() -> int {
        // legal
        print_string("print me!");
        return 0;
        print_string("don't print me!");
        int i = 5;
}
```

test_return_2.dcf

```
main() -> int {
        // Check nested ifs that are guaranteed to return
        if (true) {
                return 0;
        }
        elseif (false) {
                if (true) {
                        return 0;
                }
                else {
                        return 0;
                }
        }
        else {
                return 0;
        }
}
```

test_return_3.dcf

```
main() -> int {
        // Guaranteed to return
        if (true) {
                return 0;
        }
        else {
                return 0;
        }

        int x = 5;
        int y = 3;
}
```

test_scope_1.dcf

```
main() -> int {
        if (true) {
                int i;
        }

        int i;
        return 0;
}
```

test_scope_2.dcf

```
main() -> int {
        int i = 0;

        while (i < 5) {
                string s = "hi\n";
                print_string(s);
                i = i + 1;
        }

        while (i < 10) {
                string s = "bye\n";
                print_string(s);
                i = i + 1;
        }
        return 0;
}
test_while_1.dcf
main() -> int {
        int x = 3;
        while (x < 5) {
                print_string("hi");
                x = x + 1;
        }
        return 0;
}
test_while_for.dcf
main() -> int {
        int i = 0;
        int j;
        while (i < 3) {
                for (j = i; j > 0; j = j - 1) {
```

```
                        print_string("ij\t");
                }
                i = i + 1;
        }
        return 0;
}
```

Test Results

```
tests/test_assign_1...OK
tests/test_assign_2...OK
tests/test_assign_3...OK
tests/test_assign_4...OK
tests/test_cast_1...OK
tests/test_cast_2...OK
tests/test_cast_3...OK
tests/test_cast_4...OK
tests/test_cast_5...OK
tests/test_class_1...OK
tests/test_class_2...OK
tests/test_class_3...OK
tests/test_class_4...OK
tests/test_continue_1...OK
tests/test_continue_2...OK
tests/test_declare_1...OK
tests/test_declare_2...OK
tests/test_for_1...OK
tests/test_for_2...OK
tests/test_for_while...OK
tests/test_func_call_1...OK
tests/test_func_call_2...OK
tests/test_func_call_3...OK
tests/test_gcd...OK
tests/test_global_hello_world...OK
tests/test_hello_world...OK
tests/test_if_1...OK
tests/test_if_2...OK
tests/test_if_3...OK
tests/test_if_4...OK
tests/test_integration_arraylist...OK
tests/test_integration_bankacct...OK
tests/test_integration_mergesort...OK
```

```
tests/test_list_1...OK
tests/test_list_2...OK
tests/test_list_access_1...OK
tests/test_lits_1...OK
tests/test_return_1...OK
tests/test_return_2...OK
tests/test_return_3...OK
tests/test_scope_1...OK
tests/test_scope_2...OK
tests/test_while_1...OK
tests/test_while_for...OK
tests/fail_assign_1...OK
tests/fail_assign_2...OK
tests/fail_break_1...OK
tests/fail_break_2...OK
tests/fail_const_1...OK
tests/fail_continue_1...OK
tests/fail_declare_1...OK
tests/fail_declare_2...OK
tests/fail_func_call_1...OK
tests/fail_return_1...OK
tests/fail_return_2...OK
tests/fail_return_3...OK
tests/fail_return_4...OK
tests/fail_return_5...OK
tests/fail_scope_1...OK
tests/fail_scope_2...OK
```

7. Lessons Learned

Hidy Han

Since the technical aspects in the project have been much emphasized, I'd like to share some other insights. At the first sight, implementing a new language may be a daunting task, especially when one also aims to be innovative. However, I feel that it is important to decide on a problem, define its scope and save sufficient time to refine it, rather than envisioning a lot of features at an early stage. Only after I spent a significant amount of time coding did I understand some advantages as well as some weaknesses in our current design. In my opinion, it is more efficient to go back and change some previous design choices as we code than to try to make everything

perfect at the beginning. If our entire timeline could be shifted one week earlier, we could have refined our current implementations much better.

On the other hand, teamwork is another challenging aspect of the project. It may often happen that some of your teammates must prioritize other personal commitments or for a variety of reasons not be a stable enough state to do much work. In these cases, communication becomes especially important, and team as a whole must be resilient. I am glad that my team has been overall very competent in handling these cases. To future groups, I suggest that all teammates should keep each other accountable and always reach consensus on your expectations.

JiaYan Hu

The most difficult part of the project was navigating the unfamiliar territory of LLVM and OCaml. Since this is the first time anyone has used LLVM within the group, it became clear to us early on that we were not going to be able to finish implementing all of the features that we had set out to accomplish in the initial LRM, thus we simply prioritized our features and did as many of them as we could. However, I feel that if we had each familiarized ourselves more with LLVM from the start, the process would perhaps have been easier.

Thus as advice to future groups, I would suggest getting yourselves familiarized with LLVM and OCaml documentation early on as to smooth the implementation process. Another important aspect of getting the most out of your project is to work steadily. Our team worked throughout the semester but took a few weeks off in between due to other coursework. However, if we had planned a bit better from the outset, I think we would have accomplished more. I suggest setting at least one implementation goal each week, and meet that goal as if it's just another mandatory deadline for the class.

Kim Tao

The most difficult part of the class was learning to work with LLVM, and even in the end I feel that it would have gone more smoothly if I understood LLVM better. However, I feel like I learned to appreciate functional programming and I learned a lot about the inner workings of a compiler. Although it was challenging, it was also extremely satisfying when a new feature would work. If I could go back and redo the project, I would reconsider the original plan to focus only on the most important component of our language, which is the object oriented part.

As advice to future readers, there is no such thing as starting too early. The key to getting your language done is via dictatorship, not democracy. The most important

part of the project is to start early, communicate with your team, and keep working continuously throughout the whole semester on the necessary features.

Kylie Wu

Writing a programming language is a huge endeavor, especially when implementing it in a functional language that the team was not super familiar with. Using OCaml to generate LLVM was something none of us had done before, so accomplishing different goals accordingly to our timeline proved to be difficult at times. We had to be especially detail-oriented when we were working on this, to ensure that we covered all cases and didn't leave anything that would make the compiler behave unexpectedly. The team was able to put together a good amount of features, but if we had more structure in our planning and were more rigorous about following such plans, we would have been able to do more.

Future groups really should take the time to explicitly lay out what they want to have done each week and start early. As seen in our graph of commit history, we started out strong, took a break, worked some more, took a break, and worked till the end. While this is better than not working on the project much at the beginning and then trying to cram it in at the end, we definitely could have done more if we hadn't dipped in some weeks due to other concurrent assignments in other classes.

8. Appendix

scanner.mll

```
(* Ocamllex scanner for DECAF *)

{
      open Parser
      (* Currently unused, keeping for potential future use *)
      let unescape s =
            Scanf.sscanf("\"" ^ s ^ "\"") "%S%!" (fun x-> x)
}

let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let escape = '\\' ['\\' ''' '"' 'n' 'r' 't']

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

```
| '('       { LPAREN }
| ')'       { RPAREN }
| '{'       { LBRACE }
| '}'       { RBRACE }
| '['       { LBRACK }
| ']'       { RBRACK }
| ';'       { SEMI }
| ','       { COMMA }
| '+'       { PLUS }
| '-'       { MINUS }
| '*'       { TIMES }
| '/'       { DIVIDE }
| '%'       { MOD }
| '='             { ASSIGN }
| "=="            { REQ }
| "==="           { VEQ }
| "!="            { RNEQ }
| "!=="           { VNEQ }
| '<'       { LT }
| "<="            { LEQ }
| '>'       { GT }
| ">="            { GEQ }
| "and"           { AND }
| "or"            { OR }
| "not"           { NOT }
| "if"            { IF }
| "else"    { ELSE }
| "elseif"  { ELSEIF }
| "break"   { BREAK }
| "continue"      { CONTINUE }
| "for"           { FOR }
| "while"   { WHILE }
| "return"  { RETURN }
| "int"           { INT }
| "bool"    { BOOL }
| "void"    { VOID }
| "char"    { CHAR }
| "string"  { STRING }
| "float"   { FLOAT }
| "true"    { TRUE }
| "false"   { FALSE }
| "null"    { NULL }
```

```
      | "->"            { ARROW }
      | "@"       { AMP }
      | "~"             { TILDE }
      | "."       { DOT }
      | "in"            { IN }
      | ":"       { SNGCOLON }
      | "::"            { DBLCOLON }
      | "throw"   { THROW }
      | "try"           { TRY }
      | "catch"   { CATCH }
      | "finally" { FINALLY }
      | "class"   { CLASS }
      | "self"    { SELF }
      | "super"   { SUPER }
      | "extends" { EXTENDS }
      | "implements"    { IMPLEMENTS }
      | "const"   { CONST }
      | ['0'-'9']+ as lxm { INTLIT(int_of_string lxm) }
      | ['0'-'9']+'.'['0'-'9']+ as lxm { FLTLIT(float_of_string lxm) }
      | ['a'-'z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
      | ['A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { CLASSID(lxm) }
      | '''((ascii|escape))''' as c { CHARLIT(c.[1]) }
      | '"'((ascii|escape)* as s)'"'{ STRLIT(unescape s) }
      | eof { EOF }
      | _ as char { raise (Failure("illegal character " ^ Char.escaped
char)) }

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

and singleLineComment = parse
 '\n' { token lexbuf }
| _         { singleLineComment lexbuf }
```

parser.mly

```
/* Ocamlyacc parser for DECAF */

%{ open Ast %}

%token SEMI LPAREN RPAREN LBRACE RBRACE LBRACK RBRACK COMMA
%token PLUS MINUS TIMES DIVIDE MOD ASSIGN
```

```
%token REQ VEQ RNEQ VNEQ LT LEQ GT GEQ TRUE FALSE AND OR NOT
%token RETURN IF ELSE ELSEIF BREAK CONTINUE FOR WHILE THROW TRY CATCH
FINALLY
%token INT BOOL VOID STRING CHAR FLOAT NULL
%token ARROW AMP TILDE DOT IN SNGCOLON DBLCOLON
%token CLASS SELF SUPER EXTENDS IMPLEMENTS CONST
%token <int> INTLIT
%token <float> FLTLIT
%token <char> CHARLIT
%token <string> STRLIT
%token <string> ID CLASSID
%token EOF

%nonassoc NOELSE NOFINALLY
%right ASSIGN
%left OR
%left AND
%left REQ RNEQ VEQ VNEQ
%left LT GT LEQ GEQ
%left IN
%right DBLCOLON TILDE AMP
%left PLUS MINUS MOD
%left TIMES DIVIDE
%right NOT NEG
%left LBRACK
%right DOT

%start program
%type <Ast.program> program

%%

program:
      global_decls EOF  { Program($1) }

global_decls:
        /* nothing */ { {
            cdecls = [];
            fdecls = [] } }
      | cdecl global_decls    { {
            cdecls = $1 :: $2.cdecls;
            fdecls = $2.fdecls      } }
```

```
      | fdecl global_decls    { {
            cdecls = $2.cdecls;
            fdecls = $1 :: $2.fdecls } }

cdecl:
        CLASS CLASSID LBRACE cbody RBRACE { {
            cname = $2;
            cbody = $4;
            sclass = None;
            interfaces = None } }
      | CLASS CLASSID EXTENDS CLASSID LBRACE cbody RBRACE   { {
            cname = $2;
            cbody = $6;
            sclass = Some $4;
            interfaces = None } }
      | CLASS CLASSID IMPLEMENTS classid_list_opt LBRACE cbody RBRACE
      { {
            cname = $2;
            cbody = $6;
            sclass = None;
            interfaces = $4 } }
      | CLASS CLASSID EXTENDS CLASSID IMPLEMENTS classid_list_opt LBRACE
cbody
        RBRACE { {
            cname = $2;
            cbody = $8;
            sclass = Some $4;
            interfaces = $6 } }

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

fdecl:
      ID LPAREN formals_opt RPAREN ARROW return_typ LBRACE stmt_list
```

```
RBRACE
      { {
            fname = $1;
            formals = $3;
            return_typ = $6;
            body = List.rev $8 } }
      | CLASSID LPAREN formals_opt RPAREN ARROW return_typ LBRACE
stmt_list RBRACE
      { {
            fname = $1;
            formals = $3;
            return_typ = $6;
            body = List.rev $8 } }

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

formal_list:
        typ ID                          { [Formal($1, $2)] }
      | formal_list COMMA typ ID    { Formal($3, $4) :: $1 }

typ:
        primitive { $1 }
      | array_typ { $1 }
      | obj_typ   { $1 }

return_typ:
        VOID            { Void }
      | typ       { $1 }

primitive:
        INT       { Int }
      | CHAR            { Char }
      | BOOL            { Bool }
      | STRING    { String }
      | FLOAT           { Float }

array_typ:
      LBRACK typ RBRACK { ArrayTyp($2) }

obj_typ:
```

```
      CLASSID                       { ClassTyp($1) }

classid_list_opt:
      /* nothing */ { Some [] }
      | classid_list          { Some (List.rev $1) }

classid_list:
        CLASSID                           { [$1] }
      | classid_list COMMA CLASSID  { $3 :: $1 }

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

stmt:
        expr SEMI { Expr $1 }
      | RETURN SEMI { Return Noexpr }
      | RETURN expr SEMI { Return $2 }
      /* if */
      | IF LPAREN expr RPAREN LBRACE stmt_list RBRACE %prec NOELSE
            { If($3, Block(List.rev $6), [], Block([])) }
      /* if-else*/
      | IF LPAREN expr RPAREN LBRACE stmt_list RBRACE ELSE LBRACE
stmt_list RBRACE
            { If($3, Block(List.rev $6), [],  Block(List.rev $10)) }
      /* if-elseif */
      | IF LPAREN expr RPAREN LBRACE stmt_list RBRACE
        elseifs
        %prec NOELSE
            { If($3, Block(List.rev $6), $8, Block([])) }
      /* if-elseif-else */
      | IF LPAREN expr RPAREN LBRACE stmt_list RBRACE
        elseifs
        ELSE LBRACE stmt_list RBRACE
            { If($3, Block(List.rev $6), $8, Block(List.rev $11)) }
      | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN LBRACE
stmt_list RBRACE
            { For($3, $5, $7, Block(List.rev $10)) }
      | WHILE LPAREN expr RPAREN LBRACE stmt_list RBRACE
            { While($3, Block(List.rev $6)) }
      | BREAK     SEMI { Break }
      | CONTINUE SEMI { Continue }
```

```
      | TRY LBRACE stmt_list RBRACE catch_list %prec NOFINALLY
          { TryCatch(Block(List.rev $3), $5, Block([])) }
      | TRY LBRACE stmt_list RBRACE catch_list FINALLY LBRACE stmt_list
RBRACE
          { TryCatch(Block(List.rev $3), $5, Block(List.rev $8)) }
      | THROW expr SEMI { Throw($2) }
      | typ ID SEMI     { LocalVar($1, $2, Noexpr) }
      | typ ID ASSIGN expr SEMI     { LocalVar($1, $2, $4) }
      | CONST primitive ID SEMI     { LocalConst($2, $3, Noexpr) }
      | CONST primitive ID ASSIGN expr SEMI     { LocalConst($2, $3, $5)
}

catch_list:
        CATCH LPAREN CLASSID ID RPAREN LBRACE stmt_list RBRACE
          { [Catch(ClassTyp($3), $4, Block(List.rev $7))] }
      | CATCH LPAREN CLASSID ID RPAREN LBRACE stmt_list RBRACE
catch_list
          { Catch(ClassTyp($3), $4, Block(List.rev $7)) :: $9}

elseifs:
        ELSEIF LPAREN expr RPAREN LBRACE stmt_list RBRACE
          { [Elseif($3, Block(List.rev $6))] }
      | ELSEIF LPAREN expr RPAREN LBRACE stmt_list RBRACE elseifs
          { Elseif($3, Block(List.rev $6)) :: $8 }

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

expr:
        lits                        { $1 }
      | 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 MOD expr         { Binop($1, Mod, $3) }
      | expr REQ expr         { Binop($1, Req, $3) }
      | expr VEQ expr         { Binop($1, Veq, $3) }
      | expr RNEQ expr  { Binop($1, Rneq, $3) }
      | expr VNEQ expr  { Binop($1, Vneq, $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 DBLCOLON expr    { Binop($1, Append, $3) }
      | expr AMP expr         { Binop($1, Concat, $3) }
      | expr IN expr          { Binop($1, In, $3) }
      | MINUS expr %prec NEG  { Unop(Neg, $2) }
      | NOT expr              { Unop(Not, $2) }
      | TILDE expr            { Unop(Remove, $2) }
      | LT typ GT expr  { Cast($2, $4) }
      | expr ASSIGN expr      { Assign($1, $3) }
      | LPAREN expr RPAREN    { $2 }
      | sequence  { ArrayCreate(fst $1, snd $1) }
      | expr sequence_access  { SeqAccess($1, $2) }
      | expr DOT ID     { FieldAccess($1, Id($3)) }
      | ID LPAREN actuals_opt RPAREN      { FuncCall($1, $3) }
      | expr DOT ID LPAREN actuals_opt RPAREN   { MethodCall($1, $3, $5)
}
      | obj_typ LPAREN actuals_opt RPAREN { ObjCreate($1, $3) }
      | SELF      { Self }
      | SUPER LPAREN actuals_opt RPAREN   { Super($3) }

lits:
        INTLIT                { IntLit($1) }
      | FLTLIT                { FloatLit($1) }
      | CHARLIT               { CharLit($1) }
      | STRLIT                { StringLit($1) }
      | TRUE                        { BoolLit(true) }
      | FALSE                       { BoolLit(false) }
      | ID                    { Id($1) }
      | NULL                        { Null }

sequence:
        LBRACK sequence COMMA expr RBRACK       { (fst $2, $4 :: (snd
$2)) }
      | LBRACK primitive COMMA expr RBRACK      { ($2, [$4]) }
      | LBRACK obj_typ COMMA expr RBRACK        { ($2, [$4]) }

sequence_access:
        LBRACK expr RBRACK    { $2 }
```

```
vdecl:
        typ ID SEMI     { ObjVar($1, $2, Noexpr) }
      | typ ID ASSIGN expr SEMI     { ObjVar($1, $2, $4) }
      | CONST primitive ID SEMI     { ObjConst($2, $3, Noexpr) }
      | CONST primitive ID ASSIGN expr SEMI     { ObjConst($2, $3, $5) }

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

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

ast.ml

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

(*open Char*)

type op = Add | Sub | Mult | Div | Mod | Req | Veq | Rneq | Vneq | Less
| Leq |
      Greater | Geq | And | Or | In | Append | Concat

type uop = Neg | Not | Remove

type typ = Int | Float | Bool | Char | String | Void | Null_t | Any |
      Tuple of typ | ArrayTyp of typ | ClassTyp of string | CharArray
of int

(* typ ID, e.g. int x, int[] y *)
type formal_param = Formal of typ * string

type classid_list = string list

type expr =
        IntLit of int
      | BoolLit of bool
      | FloatLit of float
      | CharLit of char
      | StringLit of string
      | Id of string
      | Null
```

```
      | Binop of expr * op * expr
      | Unop of uop * expr
      | Assign of expr * expr
      | Cast of typ * expr
      | ArrayCreate of typ * expr list
      | SeqAccess of expr * expr
      | FieldAccess of expr * expr
      | MethodCall of expr * string * expr list
      | FuncCall of string * expr list
      | ObjCreate of typ * expr list
      | Self
      | Super of expr list
      | Noexpr

type stmt =
        Block of stmt list
      | Expr of expr
      | Return of expr
      | If of expr * stmt * stmt list * stmt
      | Elseif of expr * stmt
      | For of expr * expr * expr * stmt
      | While of expr * stmt
      | Break
      | Continue
      | LocalVar of typ * string * expr
      | LocalConst of typ * string * expr
      | TryCatch of stmt * stmt list * stmt
      | Catch of typ * string * stmt
      | Throw of expr

type field = ObjVar of typ * string * expr | ObjConst of typ * string *
expr

type func_decl = {
      return_typ: typ;
      fname: string;
      formals: formal_param list;
      (* locals: local list; *)
      body: stmt list;
}

type class_body = {
```

```
      fields: field list;
      methods: func_decl list;
}

type class_decl = {
      cname: string;
      cbody: class_body;
      sclass: string option;
      interfaces: classid_list option;
}

type global_decls = {
      cdecls: class_decl list;
      fdecls: func_decl list;
}

type program = Program of global_decls

(* Pretty-printing functions *)

let rec string_of_typ = function
        Int -> "int"
      | Float -> "float"
      | Bool -> "bool"
      | String -> "string"
      | Void -> "void"
    | Null_t -> "null"
      | Any -> "any"
      | Char -> "char"
      | Tuple(t) -> "(" ^ string_of_typ t ^ ")"
      | ArrayTyp(t) -> "[" ^ string_of_typ t ^ "]"
      | ClassTyp(id) -> id
    | CharArray(n) -> "array of length " ^ string_of_int n

let string_of_op = function
        Add -> "+"
      | Sub -> "-"
      | Mult -> "*"
      | Div -> "/"
      | Mod -> "%"
      | Req -> "=="
      | Veq -> "==="
```

```
      | Rneq -> "!="
      | Vneq -> "!=="
      | Less -> "<"
      | Leq -> "<="
      | Greater -> ">"
      | Geq -> ">="
      | And -> "and"
      | Or -> "or"
      | In -> "in"
      | Append -> "::"
      | Concat -> "@"

let string_of_uop = function
        Neg -> "-"
      | Not -> "not"
      | Remove -> "~"

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

let rec string_of_expr = function
        IntLit(l) -> string_of_int l
      | FloatLit(f) -> string_of_float f
      | BoolLit(true) -> "true"
      | BoolLit(false) -> "false"
      | CharLit(c) -> Char.escaped c
      | StringLit(s) -> s
      | Id(s) -> s
      | Null -> "null"
      | 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
      | Cast(t, e) -> "<" ^ string_of_typ t ^ ">" ^ string_of_expr e
      | ArrayCreate(typ, exprs) ->
            let rec string_list exprs = match exprs with
                    [] -> ""
                  | [head] -> "[" ^ (string_of_typ typ) ^ ", " ^
                    (string_of_expr head) ^ "]"
                  | head :: tail -> "[" ^ (string_list tail) ^ ", " ^
(string_of_expr
                        head) ^ "]"
```

```
            in
            string_list exprs
      | SeqAccess(sequence, index) ->
            string_of_expr sequence ^
            "[" ^ string_of_expr index ^ "]"
      | FieldAccess(obj, field) -> string_of_expr obj ^ "." ^
string_of_expr field
      | MethodCall(obj, fname, args) ->
            string_of_expr obj ^ "." ^ fname ^
            "(" ^ String.concat ", " (List.map string_of_expr args) ^
")"
      | FuncCall(fname, args) ->
            fname ^ "(" ^ String.concat ", " (List.map string_of_expr
args) ^ ")"
      | ObjCreate(obj, args) ->
            string_of_typ obj ^
            "(" ^ String.concat ", " (List.map string_of_expr args) ^
")"
      | Self -> "self"
      | Super(args) ->
            "super(" ^ String.concat ", " (List.map string_of_expr
args) ^ ")"
      | Noexpr -> ""

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 *)
      | If(if_expr, if_block , [], Block([])) ->
            "if (" ^ string_of_expr if_expr ^ ") " ^ string_of_stmt
if_block
      (* if-else *)
      | If(if_expr, if_block, [], else_block) ->
            "if (" ^ string_of_expr if_expr ^ ") " ^ string_of_stmt
if_block ^
            "else " ^ string_of_stmt else_block
      (* if-elseif *)
      | If(if_expr, if_block, elseifs, Block([])) ->
            "if (" ^ string_of_expr if_expr ^ ") " ^ string_of_stmt
```

```
if_block ^
            String.concat "\n" (List.map string_of_stmt elseifs)
      (* if-elseif-else *)
      | If(if_expr, if_block, elseifs, else_block) ->
            "if (" ^ string_of_expr if_expr ^ ") " ^ string_of_stmt
if_block ^
            String.concat "\n" (List.map string_of_stmt elseifs) ^
            "else " ^ string_of_stmt else_block
      | Elseif(e, block) ->
            "elseif (" ^ string_of_expr e ^ ") " ^ string_of_stmt block
      | For(e1, e2, e3, block) ->
            "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^
" ; " ^
            string_of_expr e3  ^ ") " ^ string_of_stmt block ^ "\n"
      | While(e, block) ->
            "while (" ^ string_of_expr e ^ ")" ^ string_of_stmt block
      | Break -> "break;\n"
      | Continue -> "continue;\n"
      | LocalVar(t, id, e) ->
            string_of_typ t ^ " " ^ id ^ " " ^ string_of_expr e ^ ";\n"
      | LocalConst(t, id, e) ->
            "const" ^ string_of_typ t ^ " " ^ id ^ " " ^ string_of_expr
e ^ ";\n"
      | TryCatch(try_block, catch_list, finally_block) ->
            "try " ^ string_of_stmt try_block ^
            String.concat "\n" (List.map string_of_stmt catch_list) ^
            "\nfinally " ^ string_of_stmt finally_block
      | Catch(t, id, block) ->
            "catch (" ^ string_of_typ t ^ " " ^ id ^ ") " ^
string_of_stmt block
      | Throw(e) -> "throw " ^ string_of_expr e

let string_of_formal = function
        Formal(t, name) -> string_of_typ t ^ " " ^ name

let string_of_field = function
        ObjVar(t, id, e) ->
            string_of_typ t ^ " " ^ id ^ " = " ^ string_of_expr e ^
";\n"
      | ObjConst(t, id, e) ->
            "const" ^ string_of_typ t ^ " " ^ id ^ " = " ^
string_of_expr e ^ ";\n"
```

```
let string_of_func_decl func_decl =
      func_decl.fname ^ "(" ^
      String.concat ", " (List.map string_of_formal func_decl.formals)
^
      ") ->" ^ string_of_typ func_decl.return_typ ^ " {\n" ^
      String.concat "" (List.map string_of_stmt func_decl.body) ^ "}\n"

let string_of_class_decl class_decl =
      "class " ^ class_decl.cname ^ " {\n" ^
      String.concat "" (List.map string_of_field
class_decl.cbody.fields) ^ "\n" ^
      String.concat "\n" (List.map string_of_func_decl
class_decl.cbody.methods) ^
      "\n}\n"

let string_of_global_decls gdecls =
      String.concat "\n" (List.map string_of_class_decl gdecls.cdecls)
^
      String.concat "\n" (List.map string_of_func_decl gdecls.fdecls)

let string_of_program program = match program with
      Program gdecls -> string_of_global_decls gdecls
```

sast.ml

```
open Ast

type sexpr =
      SIntLit of int
    | SBoolLit of bool
    | SFloatLit of float
    | SCharLit of char
    | SStringLit of string
    | SId of string * typ
    | SNull
    | SBinop of sexpr * op * sexpr * typ
    | SUnop of uop * sexpr * typ
    | SAssign of sexpr * sexpr * typ
    | SCast of typ * sexpr
      | SArrayCreate of sexpr list * typ
      | SSeqAccess of sexpr * sexpr * typ
    | SFieldAccess of sexpr * sexpr * typ
```

```
    | SCall of string * sexpr list * typ
    | SMethodCall of sexpr * string * sexpr list * typ
    | SObjCreate of typ * sexpr list
    | SNoexpr

type sstmt =
      SBlock of sstmt list
    | SExpr of sexpr * typ
    | SReturn of sexpr * typ
    | SIf of sexpr * sstmt * sstmt list * sstmt
    | SElseif of sexpr * sstmt
    | SFor of sexpr * sexpr * sexpr * sstmt
    | SWhile of sexpr * sstmt
    | SBreak
    | SContinue
    | SLocalVar of typ * string * sexpr
    | SLocalConst of typ * string * sexpr

type sfunc_decl = {
    stype: typ;
    sfname: string;
    sformals: formal_param list;
    sbody: sstmt list;
}

type sclass_body = {
    sfields: field list;
    smethods: sfunc_decl list;
}

type sclass_decl = {
    scname: string;
    scbody: sclass_body;
}

type sprogram = {
    classes: sclass_decl list;
    functions: sfunc_decl list;
    reserved: sfunc_decl list;
}
```

semant.ml

```
(* Semantic checking for the DECAF compiler *)

open Ast
open Sast

module StringMap = Map.Make(String)

type class_map = {
      class_decl:       Ast.class_decl;
      class_methods:    Ast.func_decl StringMap.t;
      class_reserved_methods: Sast.sfunc_decl StringMap.t;
      class_fields:     Ast.field StringMap.t;
}

type env = {
      env_name:         string;
      env_locals:       typ StringMap.t;
      env_consts:       typ StringMap.t;
      env_params:       Ast.formal_param StringMap.t;
      env_ret_typ:      typ;
      env_reserved:     Sast.sfunc_decl list;
      env_class_maps:   class_map StringMap.t;
      (* Block of "block_name", local var map, and local const map *)
      env_blocks:       (string * (typ StringMap.t) * (typ StringMap.t))
list;
}

let update_env_name env name = {
      env_name = name;
      env_locals = env.env_locals;
      env_consts = env.env_consts;
      env_params = env.env_params;
      env_ret_typ = env.env_ret_typ;
      env_reserved = env.env_reserved;
      env_class_maps = env.env_class_maps;
      env_blocks = env.env_blocks;
}

let add_empty_block env block_str = {
      env_name = env.env_name;
      env_locals = env.env_locals;
```

```
      env_consts = env.env_consts;
      env_params = env.env_params;
      env_ret_typ = env.env_ret_typ;
      env_reserved = env.env_reserved;
      env_class_maps = env.env_class_maps;
      env_blocks = (block_str, StringMap.empty, StringMap.empty) ::
            env.env_blocks;
}

let update_env_class_maps env class_maps = {
      env_name = env.env_name;
      env_locals = env.env_locals;
      env_consts = env.env_consts;
      env_params = env.env_params;
      env_ret_typ = env.env_ret_typ;
      env_reserved = env.env_reserved;
      env_class_maps = class_maps;
      env_blocks = env.env_blocks;
}

(* Add new local to env *)
let add_env_local env id typ = {
      env_name =env.env_name;
      env_locals = StringMap.add id typ env.env_locals;
      env_consts = env.env_consts;
      env_params = env.env_params;
      env_ret_typ = env.env_ret_typ;
      env_reserved = env.env_reserved;
      env_class_maps = env.env_class_maps;
      env_blocks = env.env_blocks;
}

(* Add new const to env *)
let add_env_const env id typ = {
      env_name =env.env_name;
      env_locals = env.env_locals;
      env_consts = StringMap.add id typ env.env_consts;
      env_params = env.env_params;
      env_ret_typ = env.env_ret_typ;
      env_reserved = env.env_reserved;
      env_class_maps = env.env_class_maps;
      env_blocks = env.env_blocks;
```

```
}

(* Add local var to current block *)
let add_env_block_local env id typ = {
      env_name = env.env_name;
      env_locals = env.env_locals;
      env_consts = env.env_consts;
      env_params = env.env_params;
      env_ret_typ = env.env_ret_typ;
      env_reserved = env.env_reserved;
      env_class_maps = env.env_class_maps;
      env_blocks = match env.env_blocks with
              [] -> raise(Failure("No env blocks found"))
            | head :: tail ->
                  let block_str, local_map, const_map = head in
                  let new_local_map = StringMap.add id typ local_map in
                  (block_str, new_local_map, const_map) :: tail
}

(* Add local const to current block *)
let add_env_block_const env id typ = {
      env_name = env.env_name;
      env_locals = env.env_locals;
      env_consts = env.env_consts;
      env_params = env.env_params;
      env_ret_typ = env.env_ret_typ;
      env_reserved = env.env_reserved;
      env_class_maps = env.env_class_maps;
      env_blocks = match env.env_blocks with
              [] -> raise(Failure("No env blocks found"))
            | head :: tail ->
                  let block_str, local_map, const_map = head in
                  let new_const_map = StringMap.add id typ const_map in
                  (block_str, local_map, new_const_map) :: tail
}


let global_func_map_ref = ref StringMap.empty
let global_class_map_ref = ref StringMap.empty

let reserved_list =
(* Helper function to get all built-in functions. *)
```

```
      let reserved_struct name return_t formals =
            {
                  stype = return_t;
                  sfname = name;
                  sformals = formals;
                  sbody = [];
            }
      in
            (* wrapper types, so that Any can be used *)
      let reserved = [
            reserved_struct "print_string" (Void) ([Formal(String,
                  "string_arg")]);
            reserved_struct "print_int" (Void) ([Formal(Int,
"int_arg")]);
            reserved_struct "print_char" (Void) ([Formal(Char,
"char_arg")]);
            reserved_struct "print_float" (Void) ([Formal(Float,
                  "float_arg")]);
            reserved_struct "malloc" (CharArray(1)) ([Formal(Int,
"size")]);
            reserved_struct "cast" (Any) ([Formal(Any, "victim")]);
      ] in reserved

let reserved_map = List.fold_left (
                  fun map f -> StringMap.add f.sfname f map
            ) StringMap.empty reserved_list


(* global functions, main and constructors retain their original names
 * constructors retain their names because they are only the methods
that
 * can be called outside the class (i.e. env_name = "") w/out
 *    obj.method...
 *)
let get_fully_qualified_name class_name fname = match fname with
        "main" -> "main"
      | _ ->
            if (class_name = "" || class_name = fname) then fname
            else class_name ^ "." ^ fname

(* Put all global function declarations into a map *)
let get_global_func_map fdecls reserved_map =
```

```
      let map_global_funcs map fdecl =
            if (StringMap.mem fdecl.fname map) then
                  raise (Failure(" duplicate global function: " ^
fdecl.fname))
            else if (StringMap.mem fdecl.fname reserved_map) then
                  raise (Failure(fdecl.fname ^ " is a reserved
function."))
            else
                  StringMap.add fdecl.fname fdecl map
      in
      List.fold_left map_global_funcs StringMap.empty fdecls


(* Collect all class internals into a map as class_map's *)
let get_class_maps cdecls reserved_map =
      let map_class map cdecl =
            (* Map all fields, const and non-const. *)
            let map_fields map = function
                    ObjVar(_, name, _) as field ->
                        if (StringMap.mem name map) then
                              raise (Failure(" duplicate field name: "
^ name))
                        else
                              StringMap.add name field map
                  | ObjConst(_, name, _) as const_field ->
                        if (StringMap.mem name map) then
                              raise (Failure(" duplicate const field
name: " ^ name))
                        else
                              StringMap.add name const_field map
            in

            (* Map all functions within class declarations. *)
            let map_functions map fdecl =
                  let func_full_name = get_fully_qualified_name
cdecl.cname
                        fdecl.fname
                  in
                  (* allow duplicate functions; however, only
                   * the last declaration is picked up. Because we parse
super
```

```
                 * super class before super class before child class,
this
                 * means that child class can overwrite ancestors'
                 * declarations *)
                if (StringMap.mem fdecl.fname reserved_map) then
                      raise (Failure(fdecl.fname ^ " is a reserved
function."))
                else
                      StringMap.add func_full_name fdecl map
            in
            (* Map class names. *)
            let sclass_name = match cdecl.sclass with
                  None -> ""
                  | Some str -> str
            in
            let rec get_all_inherited_methods sclass_name lst =
                  let super_cdecl = List.filter (fun cdecl ->
                        (cdecl.cname = sclass_name)) cdecls in
            let super_cdecl = List.hd super_cdecl in
            (* Remove constructor *)
            let inherited_methods = List.filter (fun met ->
                  (met.fname <> sclass_name)) super_cdecl.cbody.methods
in
            let res = inherited_methods @ lst in
                  match super_cdecl.sclass with
                          None -> res
                        | Some str -> get_all_inherited_methods str res
                  in
            if (StringMap.mem cdecl.cname map) then
                  raise (Failure(" duplicate class name: " ^
cdecl.cname))
            else if (sclass_name <> "" && not
                  (StringMap.mem sclass_name map)) then
                  raise (Failure("superclass " ^ sclass_name ^ " doesn't
exist"))
            else if (sclass_name <> "") then (
                        let superclass = StringMap.find sclass_name map
in
                        let all_inherited_methods =
get_all_inherited_methods
                              sclass_name [] in
                        StringMap.add cdecl.cname
```

```
                        {
                              class_fields = List.fold_left map_fields
                                    superclass.class_fields
cdecl.cbody.fields;
                              class_methods = List.fold_left
map_functions
                                    StringMap.empty
                                    (all_inherited_methods @
cdecl.cbody.methods);
                              class_reserved_methods = reserved_map;
                              class_decl = cdecl;
                        } map
            )
            else (
                        StringMap.add cdecl.cname
                        {
                              class_fields = List.fold_left map_fields
                                    StringMap.empty cdecl.cbody.fields;
                              class_methods = List.fold_left
map_functions
                                    StringMap.empty cdecl.cbody.methods;
                              class_reserved_methods = reserved_map;
                              class_decl = cdecl;
                        } map
            )
      in List.fold_left map_class StringMap.empty cdecls

let get_scdecl_from_cdecl class_maps sfdecls cdecl =
      let class_map = StringMap.find cdecl.cname class_maps in
      let field_list = StringMap.fold (fun _ v lst ->
            v :: lst
      ) class_map.class_fields []
      in
      {
            scname = cdecl.cname;
            scbody = {
                  sfields = field_list;
                  smethods = sfdecls;
            }
      }

let get_type_from_sexpr = function
```

```
        SIntLit(_) -> Int
      | SBoolLit(_) -> Bool
      | SFloatLit(_) -> Float
      | SCharLit(_) -> Char
      | SStringLit(_) -> String
      | SId(_, t) -> t
      | SNull -> Null_t
      | SBinop(_, _, _, t) -> t
      | SUnop(_, _, t) -> t
      | SAssign(_, _, t) -> t
      | SCast(t, _) -> t
      | SArrayCreate(_, t) -> t
      | SSeqAccess(_, _, t) -> t
      | SFieldAccess(_, _, t) -> t
      | SCall(_, _, t) -> t
      | SMethodCall(_, _, _, t) -> t
      | SObjCreate(t, _) -> t
      | SNoexpr -> Void

let rec get_sexprl env el =
      let env_ref = ref(env) in
      let rec helper = function
            hd :: tl -> let a_head, env = get_sexpr !env_ref hd in
                  env_ref := env;
                  a_head :: (helper tl)
            | [] -> []
      in (helper el), !env_ref

(* Check a binop is valid *)
and check_binop env expr1 op expr2 =
      let sexpr1, _ = get_sexpr env expr1 in
      let sexpr2, _ = get_sexpr env expr2 in

      let typ1 = get_type_from_sexpr sexpr1 in
      let typ2= get_type_from_sexpr sexpr2 in

      match op with
              Add | Sub | Mult | Div | Mod ->
                  check_arithmetic_op sexpr1 sexpr2 op typ1 typ2, env
            | Less | Leq | Greater | Geq ->
                  check_relational_op sexpr1 sexpr2 op typ1 typ2, env
            | Veq | Vneq | Req | Rneq ->
```

```
                check_equality_op sexpr1 sexpr2 op typ1 typ2, env
            | And | Or ->
                check_logical_op sexpr1 sexpr2 op typ1 typ2, env
            | _ -> raise(Failure("Operation " ^ string_of_op op ^
                " not supported."))

(* Check an arithmetic operation is valid *)
and check_arithmetic_op sexpr1 sexpr2 op typ1 typ2 = match typ1, typ2
with
        Int, Int | Float, Float -> SBinop(sexpr1, op, sexpr2, typ1)
      (* NOTE: I didn't allow char ops but we could allow it *)
      | _ -> raise(Failure("Arithmetic operation not allowed between
types "
            ^ string_of_typ typ1 ^ " and " ^ string_of_typ typ2))

(* Check a relational operation is valid *)
and check_relational_op sexpr1 sexpr2 op typ1 typ2 = match typ1, typ2
with
        Int, Int | Float, Float -> SBinop(sexpr1, op, sexpr2, Bool)
      | _ -> raise(Failure("Relational operation not allowed between
types "
            ^ string_of_typ typ1 ^ " and " ^ string_of_typ typ2))

(* Check an equality operation is valid *)
and check_equality_op sexpr1 sexpr2 op typ1 typ2 = match op with
        Veq | Vneq -> (match typ1, typ2 with
              Int, Int | Float, Float | Bool, Bool | Char, Char |
                  String, String -> SBinop(sexpr1, op, sexpr2, Bool)
            | _ ->
            raise(Failure("Value equality/inequality operators cannot be
" ^
                  "used for types " ^ string_of_typ typ1 ^ " and " ^
                  string_of_typ typ2)))
      | Req | Rneq -> (match typ1, typ2 with
              ClassTyp(classname1), ClassTyp(classname2) ->
                  if classname1 = classname2 then
                        SBinop(sexpr1, op, sexpr2, Bool)
                  else
                        raise(Failure("Unmatching class types used in
referential "
                              ^ "equality/inequality operation"))
```

```
            | _ -> raise(Failure("Referential equality/inequality
operators " ^
                  "cannot be used for types " ^
                  string_of_typ typ1 ^ " and " ^ string_of_typ typ2)))
      | _ -> raise(Failure("Equality operation not supported"))

(* Check a logical operation is valid *)
and check_logical_op sexpr1 sexpr2 op typ1 typ2 = match typ1, typ2 with
        Bool, Bool -> SBinop(sexpr1, op, sexpr2, Bool)
      | _ ->
            raise(Failure("Boolean operation only allowed between two
boolean"
            ^ " expressions"))

and check_unop env op expr =
      let get_neg_op uop sexpr typ = match uop with
              Neg -> SUnop(uop, sexpr, typ)
            | _ ->
            raise(Failure("Illegal unary operator applied to numeric
type"))
      in
      let get_not_op uop sexpr typ = match uop with
              Not -> SUnop(uop, sexpr, typ)
            | _ ->
            raise(Failure("Illegal unary operator applied to boolean
type"))
      in

      let sexpr, _ = get_sexpr env expr in
      let typ = get_type_from_sexpr sexpr in
      (match typ with
              Int | Float -> get_neg_op op sexpr typ, env
            | Bool -> get_not_op op sexpr typ, env
            | _ ->
            raise(Failure("Unop can only be applied to numeric or
boolean" ^
                  "types")))

(* List assignment helpers *)
(* Returns true if it is a list type *)
and is_list typ = match typ with
        ArrayTyp(_) -> true
```

```
      | _ -> false

(* Returns the level of nesting in the list *)
and get_nesting_level lst = match lst with
        ArrayTyp(inner_typ) -> 1 + get_nesting_level inner_typ
      | _ -> 1

(* Check assignment, returns SAssign on success *)
and check_assign env expr1 expr2 =
      let sexpr1, env  = get_sexpr env expr1 in
      let sexpr2, env = get_sexpr env expr2 in

      let typ1 = get_type_from_sexpr sexpr1 in
      let typ2 = get_type_from_sexpr sexpr2 in

      match (typ1, sexpr2) with
              ClassTyp(_), SNull -> SAssign(sexpr1, sexpr2, typ1), env
            | _ ->
      (* Only allow assignment lhs to be id or object field *)
      match sexpr1 with
              SId(id, _) ->
                  if StringMap.mem id env.env_consts then
                        raise(Failure("Cannot assign to const id " ^
id))
                  (* Check types match *)
                  else if typ1 = typ2 then
                        SAssign(sexpr1, sexpr2, typ2), env
                  else
                        raise(Failure("Cannot assign type " ^
                              string_of_typ typ2 ^ " to type " ^
string_of_typ typ1))
            | SSeqAccess(_, index_sexpr, _) ->
                  if get_type_from_sexpr index_sexpr = Int && typ1 =
typ2 then
                        SAssign(sexpr1, sexpr2, typ1), env
                  else
                        raise(Failure("Invalid sequence access
assignment"))

            | SFieldAccess(_, sid, _) ->
                  let id = match sid with
                          SId(id, _) -> id
```

```
                        | _ -> raise(Failure("Object access only
allowed on ids"))
                  in
                  (* Ensure id is not a const *)
                  if StringMap.mem id env.env_consts then
                        raise(Failure("Cannot assign to const id " ^
id))
                  (* Check types match *)
                  else if typ1 = typ2 then
                        SAssign(sexpr1, sexpr2, typ2), env
                  else
                        raise(Failure("Cannot assign type " ^
                              string_of_typ typ2 ^ " to type " ^
string_of_typ typ1))
            | _ -> raise(Failure("Invalid assignment: " ^
                  string_of_expr expr1 ^ " = " ^ string_of_expr expr2))

and check_method_call env e fname el =
      let obj_id = match e with
              Id(id) -> id
            | _ -> raise(Failure("unsupported chained call")) in
      (* can process recursively for chained calls *)
      check_func_call env fname el obj_id

(* func and method call; obj_id used to distinguish: if funcCall,
 * obj_id = "", if methodCall, obj_id = id of that obj
 *)
and check_func_call env fname el obj_id =
      let (local_context, local_context_typ) =
      if obj_id = "" then
            ("", Null_t)
      else
            let obj_typ = get_id_typ env obj_id in
                  (string_of_typ obj_typ, obj_typ) in
            let global_func_map = !global_func_map_ref in
            let global_class_map = !global_class_map_ref in
            let sel, env = get_sexprl env el in
            let check_param formal param =
                  let f_typ = match formal with
                        Formal(t, _) -> t in
                  let p_typ = get_type_from_sexpr param in
                  if (f_typ = p_typ) then param
```

```
                  else raise(Failure("Incorrect type passed to
function"))
                  in
            let handle_params formals params =
                  match formals, params with
                        [], [] -> []
                      | [], _ -> raise(Failure("Formals and actuals
mismatch"))
                      | _, [] ->
                            raise(Failure("Cannot pass void as
function params"))
                      | _ ->
                            let len1 = List.length formals in
                            let len2 = List.length params in
                            if len1 <> len2 then
                                  raise (Failure("Formal and actual
argument numbers mismatch."))
                            else
                                  List.map2 check_param formals sel
                            in

      let full_name = get_fully_qualified_name local_context fname in
      try (
            let the_func = StringMap.find fname reserved_map in
            let actuals = handle_params the_func.sformals sel in
            SCall(fname, actuals, the_func.stype), env)
      with | Not_found ->
      try(
            let the_func = StringMap.find fname global_func_map in
            let actuals = handle_params the_func.formals sel in
            SCall(fname, actuals, the_func.return_typ), env)
      with | Not_found ->
            try (
                  let cmap =
                        try StringMap.find local_context
global_class_map
                        with | Not_found ->
                              raise(Failure("Class undefined " ^
local_context))
                  in
                  let the_func = StringMap.find full_name
cmap.class_methods in
```

```
                let actuals = handle_params the_func.formals sel in
                      SMethodCall(SId(obj_id, local_context_typ),
full_name,
                                  actuals, the_func.return_typ), env)
            with | Not_found ->
                  raise (Failure("Function " ^ fname ^ " not found."))

(* Currently only casts primitives, and RHS must be a literal *)
and check_cast env to_typ expr =
      let sexpr, _ = get_sexpr env expr in
      let from_typ = get_type_from_sexpr sexpr in
      (* Check cast is valid from_typ -> to_typ *)
      let scast = match from_typ with
              Bool -> (match to_typ with
                    Bool | Int | Float | String -> SCast(to_typ, sexpr)
                  | _ -> raise(Failure("Cannot cast " ^ string_of_typ
from_typ ^
                        " to " ^ string_of_typ to_typ))
                  )
            | Int -> (match to_typ with
                    Int | Bool | Float | String | Char -> SCast(to_typ,
sexpr)
                  | _ -> raise(Failure("Cannot cast " ^ string_of_typ
from_typ ^
                        " to " ^ string_of_typ to_typ))
                  )
            | Float -> (match to_typ with
                    Float | Bool | Char | Int | String  ->
SCast(to_typ, sexpr)
                  | _ -> raise(Failure("Cannot cast " ^ string_of_typ
from_typ ^
                        " to " ^ string_of_typ to_typ))
                  )
            | Char -> (match to_typ with
                    Char | Bool | Int | Float -> SCast(to_typ, sexpr)
                  | _ -> raise(Failure("Cannot cast " ^ string_of_typ
from_typ ^
                        " to " ^ string_of_typ to_typ))
                  )
            | _ -> raise(Failure("Cannot cast " ^ string_of_typ
from_typ ^
                  " to " ^ string_of_typ to_typ))
```

```
      in
      scast, env

and check_lst_create env typ exprs =
      let sexprs, env = get_sexprl env exprs in
      (* Check that the types of all sexprs match *)
      let rec check_elem_typ sexprs = match sexprs with
              [] -> ()
            | head :: tail ->
                  let typ1 = get_type_from_sexpr head in
                  if typ1 <> Int then (
                        raise(Failure("Array dimensions must be int")))
                  else
                        (* Check rest of list *)
                        check_elem_typ (tail)
      in check_elem_typ sexprs;

      let rec nested_array_typ exprs = match exprs with
              [] -> typ
            | _ :: tail -> ArrayTyp(nested_array_typ tail)
      in
      (* Make array of type *)
      SArrayCreate(sexprs, nested_array_typ exprs), env

and check_seq_access env lst index =
      (* Check first expr is a list, second and third are ints *)
      let lst_sexpr, _ = get_sexpr env lst in
      let lst_typ = get_type_from_sexpr lst_sexpr in
      let index_sexpr, _ = get_sexpr env index in
      let index_typ = get_type_from_sexpr index_sexpr in
      let check_access_types = match lst_typ, index_typ with
              ArrayTyp(_), Int -> true
            | _ -> false
      in

      (* Unwrap type from list *)
      let typ = get_type_from_sexpr lst_sexpr in
      let typ = match typ with
              ArrayTyp(t) -> t
            | _ -> raise(Failure("Invalid sequence access."))
      in
```

```
      if not check_access_types then
            (* Invalid type used in sequence access *)
            raise(Failure("Invalid type used in sequence access: " ^
            (string_of_typ lst_typ)))
      else
            SSeqAccess(lst_sexpr, index_sexpr, typ), env

and check_field_access env obj field =
      (* Find class exists *)
      let check_class_id expr = match expr with
              Id(id) ->
                  let ctyp = get_id_typ env id in
                  SId(id, ctyp)
            | Self -> SId("self", ClassTyp(env.env_name))
            | _ -> raise(Failure("No matching class found for id " ^
                  string_of_expr expr))
      in

      let get_class_name obj = match obj with
            ClassTyp(name) -> name
            | _ -> raise(Failure("Expected object type"))
      in

      (* Find the matching field for this particular class *)
      let rec check_field lhs_env class_sid top_env expr =
            let class_name = get_class_name class_sid in
            match expr with
                  Id(id) -> (
                        try (
                              let class_map = StringMap.find class_name
                                    lhs_env.env_class_maps in
                              let match_field f = match f with
                                      ObjVar(dt, _, _) -> SId(id, dt),
lhs_env
                                    | ObjConst(dt, _, _) -> SId(id, dt),
lhs_env
                              in
                              match_field (StringMap.find id
class_map.class_fields))
                        with | Not_found -> raise(Failure("Unrecognized
field")))
                  | FieldAccess(e1, e2) -> (
```

```
                        let old_env = lhs_env in
                        let lhs, new_lhs_env = check_field lhs_env
class_sid
                              top_env e1 in
                        let lhs_typ = get_type_from_sexpr lhs in
                        let new_env = update_env_name new_lhs_env
                              (get_class_name lhs_typ) in
                        let rhs, _ = check_field new_env lhs_typ
lhs_env e2 in
                        let rhs_typ = get_type_from_sexpr rhs in
                        SFieldAccess(lhs, rhs, rhs_typ), old_env)
                  | _ -> raise(Failure("Unrecognized datatype"))
      in
      let sexpr1 = check_class_id obj in
      let typ = get_type_from_sexpr sexpr1 in
      let obj_env = update_env_name env (get_class_name typ) in
      let sexpr2, _ = check_field obj_env typ env field in
      let field_type = get_type_from_sexpr sexpr2 in
      SFieldAccess(sexpr1, sexpr2, field_type), env

and check_object_create env t el =
      let s = string_of_typ t in
      let sel, env = get_sexprl env el in
      let _ =
            try StringMap.find s env.env_class_maps
            with | Not_found ->
                  raise(Failure("cannot construct undefined class
object"))
      in
      (* don't support constructor overloading *)
      SObjCreate(t, sel), env

and get_sexpr env expr = match expr with
        IntLit(i) -> SIntLit(i), env
      | BoolLit(b) -> SBoolLit(b), env
      | FloatLit(f) -> SFloatLit(f), env
      | CharLit(c) -> SCharLit(c), env
      | StringLit(s) -> SStringLit(s), env
      | Id(id) -> SId(id, (get_id_typ env id)), env
      | Null -> SNull, env
      | Binop(e1, op, e2) ->  check_binop env e1 op e2
      | Unop(op, e) -> check_unop env op e
```

```
      | Assign(e1, e2) -> check_assign env e1 e2
      | Cast(t, e) -> check_cast env t e
      | ArrayCreate(t, exprs) -> check_lst_create env t exprs
      | SeqAccess(lst_expr, index) ->
            check_seq_access env lst_expr index
      | FieldAccess(classid, field) -> check_field_access env classid
field
      | MethodCall(e, str, el) -> check_method_call env e str el
      | FuncCall(str, el) -> check_func_call env str el ""
      | ObjCreate(t, el) -> check_object_create env t el
      | Self -> SId("self", ClassTyp(env.env_name)), env
      | Noexpr -> SNoexpr, env
      | _ -> raise(Failure("cannot convert to sexpr"))

(* Looks up the type of a variable/constant *)
and get_id_typ env id =
      (* Search block declarations *)
      let rec find_block_id id blocks = match blocks with
              [] -> Void
            | head :: tail ->
                  let _, local_map, const_map = head in
                  if StringMap.mem id local_map then
                        StringMap.find id local_map
                  else if StringMap.mem id const_map then
                        StringMap.find id const_map
                  else
                        find_block_id id tail
      in
      let block_typ = find_block_id id env.env_blocks in
      if block_typ != Void then
            block_typ
      else if StringMap.mem id env.env_locals then
            (* Found in function local declarations *)
            StringMap.find id env.env_locals
      else if StringMap.mem id env.env_consts then
            (* Found in function local declarations *)
            StringMap.find id env.env_consts
      else if StringMap.mem id env.env_params then
            (* Found in function parameters *)
            let param = StringMap.find id env.env_params in
            (function Formal(typ, _) -> typ) param
      else
```

```
      (* Note: Object fields are id's that are NOT handled by this
       * currently *)
            (* Found a matching class name *)
            raise(Failure("Unknown identifier: " ^ id))

let rec check_block env stmtl = match stmtl with
        [] -> SBlock([SExpr(SNoexpr, Void)]), env
      | _ ->
            let stmtl, _ = get_sstmtl env stmtl in
            SBlock(stmtl), env

and check_expr env expr =
      let sexpr, env = get_sexpr env expr in
      let styp = get_type_from_sexpr sexpr in
      SExpr(sexpr, styp), env

and check_return env expr =
      let sexpr, env = get_sexpr env expr in
      let styp = get_type_from_sexpr sexpr in
            match styp, env.env_ret_typ with
                    Null_t, ClassTyp(_) -> SReturn(sexpr, styp), env
                  | _ ->
                        if env.env_ret_typ = styp then
                              SReturn(sexpr, styp), env
                        else
                              raise(Failure("Return type mismatch"))

and check_if env if_expr if_stmts elseifs else_stmts =
      let new_env = add_empty_block env "if" in
      let if_sexpr, if_env = get_sexpr new_env if_expr in
      let if_typ = get_type_from_sexpr if_sexpr in
      (* Check whether if expr is a bool *)
      let check_if_expr =
            if if_typ = Bool then
                  ()
            else
                  raise(Failure("If condition must be a bool"))
      in
      check_if_expr;
      let if_sstmts, _ = get_sstmt if_env if_stmts in
      (* Generate list of selseifs *)
      let rec get_selseifs elseifs = match elseifs with
```

```
              [] -> []
            | head :: tail -> match head with
                  Elseif(elseif_expr, elseif_stmts) ->
                      let elseif_sexpr, elseif_env = get_sexpr
new_env
                            elseif_expr in
                      let elseif_typ = get_type_from_sexpr
elseif_sexpr in
                      let elseif_sstmts, _ = get_sstmt elseif_env
elseif_stmts in
                      if elseif_typ = Bool then
                            SElseif(elseif_sexpr, elseif_sstmts) ::
                                  get_selseifs tail
                      else
                            raise(Failure("Elseif condition must be a
bool"))
                | _ -> raise(Failure("Non-elseif found in elseif
list"))
      in
      let selseifs = get_selseifs elseifs in
      let else_sstmts, _ = get_sstmt new_env else_stmts in
      SIf(if_sexpr, if_sstmts, selseifs, else_sstmts), env

and check_for env expr1 expr2 expr3 stmts =
      let new_env = add_empty_block env "for" in
      let sexpr1, _ = get_sexpr new_env expr1 in
      let sexpr2, _ = get_sexpr new_env expr2 in
      let sexpr3, _ = get_sexpr new_env expr3 in
      let sstmts, _ = get_sstmt new_env stmts in
      SFor(sexpr1, sexpr2, sexpr3, sstmts), env

and check_while env expr stmts =
      let new_env = add_empty_block env "while" in
      let sexpr, new_env = get_sexpr new_env expr in
      let typ = get_type_from_sexpr sexpr in
      let sstmts, _ = get_sstmt new_env stmts in
      match typ with
              Bool | Void -> SWhile(sexpr, sstmts), env
            | _ -> raise(Failure("While condition only allows bool or
void" ^
                  " expression"))
```

```
and check_break env =
      (* Check break is in a loop body *)
      let rec find_loop blocks = match blocks with
              [] -> false
            | head :: tail ->
                  let block_str, _, _ = head in
                  if block_str = "for" || block_str = "while" then
                        true
                  else
                        find_loop tail
      in
      if find_loop env.env_blocks then
            SBreak, env
      else
            raise(Failure("Break can only be used within a loop"))

and check_continue env =
      (* Check continue is in a loop body *)
      let rec find_loop blocks = match blocks with
              [] -> false
            | head :: tail ->
                  let block_str, _, _ = head in
                  if block_str = "for" || block_str = "while" then
                        true
                  else
                        find_loop tail
      in
      if find_loop env.env_blocks then
            SContinue, env
      else
            raise(Failure("Continue can only be used within a loop"))


(* Verify local var type, add to local declarations *)
and check_local_var env typ id expr =
      if StringMap.mem id env.env_locals ||
            StringMap.mem id env.env_consts then
            raise(Failure("Duplicate local declaration: " ^ id))
      else
            (* Look in block stack for duplicate declaration *)
            let rec search_block_stack id blocks = match blocks with
                    [] -> true
```

```
                  | head :: tail ->
                        let _, local_map, const_map = head in
                        if StringMap.mem id local_map ||
                              StringMap.mem id const_map then
                              raise(Failure("Duplicate block
declaration: " ^ id))
                        else
                              search_block_stack id tail
            in
            let _ = search_block_stack id env.env_blocks in
            (* If in a block, add to local block declarations, else add
to
             * local declaration stack*)
            let env = match env.env_blocks with
                    [] -> add_env_local env id typ
                  | _ -> add_env_block_local env id typ
            in
            let sexpr, env = get_sexpr env expr in
            let _ = get_type_from_sexpr sexpr in
            (* If object type, check class name exists *)
            match typ with
                    ClassTyp(classname) ->
                        if StringMap.mem classname env.env_class_maps
then
                              SLocalVar(typ, id, sexpr), env
                        else
                              raise(Failure("Unknown class type: " ^
classname))
                  | _ -> (match sexpr with
                            SNoexpr -> SLocalVar(typ, id, sexpr), env
                        | _ ->
                              (* Check types match *)
                              let typ_sexpr = get_type_from_sexpr sexpr
in
                              if typ = typ_sexpr then
                                    SLocalVar(typ, id, sexpr), env
                              else
                                    raise(Failure("Declared type of " ^
id ^
                                    " and assignment type " ^
string_of_typ typ_sexpr ^
                                    " do not match"))
```

```
                    )

(* Verify local const type and add to local declarations *)
and check_local_const env typ id expr =
      if StringMap.mem id env.env_locals ||
            StringMap.mem id env.env_consts then
            raise(Failure("Duplicate local declaration: " ^ id))
      else
            (* Look in block stack for duplicate declaration *)
            let rec search_block_stack id blocks = match blocks with
                    [] -> true
                  | head :: tail ->
                        let _, local_map, const_map = head in
                        if StringMap.mem id local_map ||
                              StringMap.mem id const_map then
                              raise(Failure("Duplicate block
declaration: " ^ id))
                        else
                              search_block_stack id tail
            in
            ignore(search_block_stack id env.env_blocks);

            (* If in a block, add to local block declarations, else add
to
             * local declaration stack*)
            let env = match env.env_blocks with
                    [] -> add_env_const env id typ
                  | _ -> add_env_block_const env id typ
            in
            let sexpr, env = get_sexpr env expr in
            match sexpr with
                    SNoexpr ->
                        SLocalConst(typ, id, sexpr), env
                  | _ ->
                        let typ_expr = get_type_from_sexpr sexpr in
                        if typ = typ_expr then
                              SLocalConst(typ, id, sexpr), env
                        else
                              raise(Failure("Declared type of " ^ id ^
                              " and assignment type " ^ (string_of_typ)
typ_expr ^
```

```
                                " do not match"))

and get_sstmt env stmt = match stmt with
        Block(blk) -> check_block env blk
      | Expr(expr) -> check_expr env expr
      | Return(expr) -> check_return env expr
      | If(if_expr, if_stmts, elseifs, else_stmts) ->
            check_if env if_expr if_stmts elseifs else_stmts
      | For(expr1, expr2, expr3, stmts) -> check_for env expr1 expr2
expr3
            stmts
      | While(expr, stmts) -> check_while env expr stmts
      | Break -> check_break env
      | Continue -> check_continue env
      | LocalVar(typ, id, expr) -> check_local_var env typ id expr
      | LocalConst(typ, id, expr) -> check_local_const env typ id expr
      | _ -> raise(Failure("unrecognized statement"))

and get_sstmtl env stmtl =
      let env_ref = ref(env) in
      let rec helper = function
            head :: tail ->
                  let sstmt, env = get_sstmt !env_ref head in
                  env_ref := env;
                  sstmt :: (helper tail)
            | [] -> []
      in
      let sstmts = (helper stmtl), !env_ref in sstmts

(* Check that a block of statements has some return statement in it *)
let rec check_block_return sblock =
      let sstmts = match sblock with
              SBlock(sstmts) -> sstmts
            | _ -> raise(Failure("Can only check SBlocks"))
      in
      let rec find_return sstmts = match sstmts with
              [] -> false
            | head :: tail -> (match head with
                    SReturn(_, _) -> true
                  | SIf(_, if_sstmts, elseifs, else_sstmts) ->
                        check_if_return if_sstmts elseifs else_sstmts
                  | _ -> find_return tail
```

```
                )
      in
      find_return sstmts

(* Check whether an if-elseif-else block will definitely return *)
and check_if_return if_sstmts elseifs else_sstmts =
      let if_returns = check_block_return if_sstmts in
      let else_returns = check_block_return else_sstmts in
      if not if_returns || not else_returns then
            false
      else
            let rec check_elseifs_return elseifs = match elseifs with
                    [] -> true
                  | head :: tail ->
                        let selseif = (match head with
                                SElseif(sexpr, stmt) -> sexpr, stmt
                              | _ -> raise(Failure("Non-elseif found in
" ^
                                    "elseifs"))
                              )
                        in
                        let _, elseif_sstmts = selseif in
                        let elseif_returns = check_block_return
elseif_sstmts in
                        if not elseif_returns then
                              false
                        else
                              check_elseifs_return tail
            in
            check_elseifs_return elseifs

(* Return type is handled in check_return *)
let check_func_has_return sfbody return_typ =
      let len = List.length sfbody in
      if return_typ = Void then
            SExpr(SNoexpr, Void)
      else if len = 0 && return_typ != Void then
            raise(Failure("Cannot have void return type and empty
function"))
      else
            (* Check if there is any return statement in func body *)
            let rec find_func_return sstmts = match sstmts with
```

```
                  [] -> false
                | head :: tail -> (match head with
                        SReturn(_, _) -> true
                      | SIf(_, if_sstmts, elseifs, else_sstmts) ->
                            (* Check if there is an if-elseif-else
block which is
                            guaranteed to return *)
                            let guaranteed_if_return =
                            check_if_return if_sstmts elseifs
else_sstmts
                            in
                            if guaranteed_if_return then
                                  true
                            else
                                  find_func_return tail
                      | _ -> find_func_return tail
                      )
            in
            if find_func_return sfbody then
                  SExpr(SNoexpr, Void)
            else
                  raise(Failure("Missing return statement for a
function that " ^
                  "does not return void"))

(* Return sfdecl for constructor *)
let get_constructor_from_fdecl cname fdecl env =
      let class_typ = ClassTyp(cname) in
      let init_self = [SLocalVar(class_typ, "self", SCall("cast",
                  [SCall("malloc", [SCall("sizeof", [SId("dummy",
class_typ)],
                  Int)], CharArray(1))], class_typ))]
      in
      let env = {
            env_name = cname;
            env_locals = StringMap.empty;
            env_consts = StringMap.empty;
            env_params = env.env_params;
            env_ret_typ = class_typ;
            env_reserved = env.env_reserved;
            env_class_maps = env.env_class_maps;
            env_blocks = [];
```

```
      } in
      (* Does not check return statement: append return in the end *)
      let func_sbody, _ = get_sstmtl env fdecl.body in
      {
            stype = class_typ;
            sfname = get_fully_qualified_name cname fdecl.fname;
            sformals = fdecl.formals;
            sbody = init_self @ func_sbody @ [SReturn(SId("self",
class_typ),
                  class_typ)];
      }

(* Convert fdecl to sfdecl, constructors handled separately by
 * get_constructor_from_decl *)
let get_sfdecl_from_fdecl class_maps reserved cname fdecl =
      let is_global = (cname = "") in
      (* global, main() in a class (and constructor) do not append self
       * to fdecl *)
      let class_formal = Formal(ClassTyp(cname), "self") in
      let get_params_map map formal = match formal with
            Formal(_, name) -> StringMap.add name formal map in
            let all_formals =
                  if (is_global || fdecl.fname = "main") then
      fdecl.formals
                  else (class_formal :: fdecl.formals)
      in
      let parameters = List.fold_left get_params_map StringMap.empty
            all_formals in
      let env = {
            env_name = cname;
            env_locals = StringMap.empty;
            env_consts = StringMap.empty;
            env_params = parameters;
            env_ret_typ = fdecl.return_typ;
            env_reserved = reserved;
            env_class_maps = class_maps;
            env_blocks = [];
      } in
      let is_constructor = ((not is_global) && fdecl.fname = cname) in
      let sfdecl =
            if is_constructor then
                  get_constructor_from_fdecl cname fdecl env
```

```
            else
                  let func_sbody, _ = get_sstmtl env fdecl.body in
                  let func_sbody = List.rev(check_func_has_return
func_sbody
                        fdecl.return_typ :: List.rev(func_sbody)) in
                  {
                        stype = fdecl.return_typ;
                        sfname = get_fully_qualified_name cname
fdecl.fname;
                        sformals = all_formals;
                        sbody = func_sbody;
                  }
            in
            sfdecl

(* Overview function to generate sast. We perform main checks here. *)
let get_sast class_maps reserved cdecls fdecls  =
      let find_main f = match f.sfname with
              "main" -> true
            | _ -> false
      in
      let check_main functions =
            let all_main_decls = List.find_all find_main functions
            in
            if (List.length all_main_decls < 1 ) then
                  raise (Failure("Main not defined."))
            else if (List.length all_main_decls  > 1) then
                  raise (Failure("More than 1 main function defined."))
            else List.hd all_main_decls
      in
      let find_constructor scdecl =
      let cons_name = scdecl.scname in
      let is_func_constructor f = (f.sfname = cons_name) in
      let scmethods = scdecl.scbody.smethods in
            try let _ = List.find is_func_constructor scmethods in
                  scmethods
                  (* only when a class called Main it does not need to
have
                   * constructor *)
            with | Not_found ->
                  if cons_name = "Main" then scmethods
                  else raise(Failure("This class has no constructor"))
```

```
            in
            let get_all_methods cname =
                  let class_map = StringMap.find cname class_maps in
                  let method_map = class_map.class_methods in
                  (* Collect all method declarations in a list,
including
                   * inherited ones *)
            StringMap.fold (fun _ v l -> v :: l) method_map []
            in
      let check_and_convert_cdecl cdecl =
            let sfunc_lst = List.fold_left (fun ls f ->
            (get_sfdecl_from_fdecl class_maps reserved cdecl.cname f)
::
                  ls) [] (get_all_methods cdecl.cname) in
            let scdecl = get_scdecl_from_cdecl class_maps sfunc_lst
cdecl in
            let _ = find_constructor scdecl in (scdecl, sfunc_lst) in
            let iter_cdecls t c =
                  let (scdecl, sfunc_lst) = check_and_convert_cdecl c
in
                        (scdecl :: fst t, sfunc_lst @ snd t)
            in
            let scdecl_lst, sfunc_lst = List.fold_left iter_cdecls ([],
[])
                  cdecls in
      let get_sfdecls l f =
            let sfdecl = (get_sfdecl_from_fdecl class_maps reserved ""
f) in
                  (sfdecl :: l) in
      let global_sfdecls = List.fold_left get_sfdecls [] fdecls in
      (* Check that there is one main function. *)
      let _ = check_main (global_sfdecls @ sfunc_lst) in
      {
            classes = scdecl_lst;
            functions = global_sfdecls;
            reserved = reserved
      }

let check program = match program with
      Program(globals) ->
            let global_func_map = get_global_func_map globals.fdecls
                  reserved_map
```

```
            in global_func_map_ref := global_func_map;
            let class_maps = get_class_maps globals.cdecls reserved_map
in
                  global_class_map_ref := class_maps;
            let sast = get_sast class_maps reserved_list globals.cdecls
                  globals.fdecls in
            sast
```

codegen.ml

```
(* Code generation: translate takes a semantically checked AST and
produces LLVM IR

LLVM tutorial: Make sure to read the OCaml version of the tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)

open Sast
open Semant

module L = Llvm
module A = Ast
module S = Sast
module Se = Semant

module Hash = Hashtbl

module StringMap = Map.Make(String)
let global_classes:(string, L.lltype) Hash.t = Hash.create 50
let local_params:(string, L.llvalue) Hash.t = Hash.create 50
let local_values:(string, L.llvalue) Hash.t = Hash.create 50
let class_self:(string, L.llvalue) Hash.t = Hash.create 50
let class_fields:(string, int) Hash.t = Hash.create 50

let context = L.global_context()
```

```
let codegen_module = L.create_module context "DECAF Codegen"
let builder = L.builder context

let i1_t = L.i1_type context;;
let i8_t = L.i8_type context;;
let i32_t = L.i32_type context;;
let f_t = L.double_type context;;
let str_t = L.pointer_type i8_t;;
let void_t = L.void_type context;;

let rec get_llvm_type = function
        A.Int -> i32_t
      | A.Float -> f_t
      | A.Bool -> i1_t
      | A.Char -> i8_t
      | A.Void -> void_t
      | A.String -> str_t
      | A.Null_t -> i32_t
      | A.ArrayTyp(typ) -> L.pointer_type (get_llvm_type typ)
      | A.ClassTyp(name) -> L.pointer_type(find_global_class name)
      | _ -> raise(Failure("Type not yet supported."))

and find_global_class name =
      try Hash.find global_classes name
      with Not_found -> raise(Failure("Invalid class name " ^ name))

let rec id_gen llbuilder id is_deref =
      if is_deref then
            if Hash.mem local_values id then
                  let _val = Hash.find local_values id in
                  L.build_load _val id llbuilder
            else if Hash.mem local_params id then
                  Hash.find local_params id
            else
                  raise(Failure("Unknown variable " ^ id))
      else
            if Hash.mem local_values id then
                  Hash.find local_values id
            else if Hash.mem local_params id then
                  Hash.find local_params id
            else
                  raise(Failure("Unknown variable " ^ id))
```

```
and func_lookup fname = match (L.lookup_function fname codegen_module)
with
              None -> raise(Failure(" function " ^ fname ^ " does not
exist."))
            | Some func -> func

and string_gen llbuilder s =
      L.build_global_stringptr s "tmp" llbuilder

and sstmt_gen llbuilder loop_stack = function
      (* NOTE: this requires a function body to be non-empty *)
        SBlock(stmts) -> List.hd(List.map (sstmt_gen llbuilder
loop_stack)
            stmts)
      | SExpr(sexpr, _) -> sexpr_gen llbuilder sexpr
      | SReturn(sexpr, _) -> ret_gen llbuilder sexpr
      | SIf(if_sexpr, if_stmts, elseifs, else_sstmts) ->
            if_gen llbuilder loop_stack if_sexpr if_stmts elseifs
else_sstmts
      | SFor(sexpr1, sexpr2, sexpr3, sstmts) ->
            for_gen llbuilder loop_stack sexpr1 sexpr2 sexpr3 sstmts
      | SWhile(sexpr, sstmts) -> while_gen llbuilder loop_stack sexpr
sstmts
      | SBreak -> break_gen llbuilder loop_stack
      | SContinue -> continue_gen llbuilder loop_stack
      | SLocalVar(typ, id, sexpr) -> local_var_gen llbuilder typ id
sexpr
      | SLocalConst(typ, id, sexpr) -> local_var_gen llbuilder typ id
sexpr
      | _ -> raise(Failure("Unknown statement reached."))

and sexpr_gen llbuilder = function
        SIntLit(i) -> L.const_int i32_t i
      | SBoolLit(b) -> if b then L.const_int i1_t 1 else L.const_int
i1_t 0
      | SCharLit(c) -> L.const_int i8_t (Char.code c)
      | SFloatLit(f) -> L.const_float f_t f
      | SStringLit(s) -> string_gen llbuilder s
      | SId(id, _) -> id_gen llbuilder id true
      | SNull -> L.const_null i32_t
      | SBinop(sexpr1, op, sexpr2, typ) ->
```

```
            binop_gen llbuilder sexpr1 op sexpr2 typ
      | SUnop(op, sexpr, typ) ->
            unop_gen llbuilder op sexpr typ
      | SAssign(sexpr1, sexpr2, typ) -> assign_gen llbuilder sexpr1
sexpr2 typ
      | SCast(to_typ, sexpr) -> cast_gen llbuilder to_typ sexpr
      | SArrayCreate(sexprl, typ) -> lst_create_gen llbuilder typ
sexprl
      | SSeqAccess(lst_sexpr, index, _) ->
            seq_access_gen llbuilder lst_sexpr index false
      | SFieldAccess(c, rhs, _) -> field_access_gen llbuilder c rhs
true
      | SCall(fname, sexpr_list, stype) -> call_gen llbuilder fname
sexpr_list
            stype
      | SMethodCall(sexpr, fname, sexpr_list, stype) -> method_call_gen
            llbuilder sexpr fname sexpr_list stype
            (* difference is insert self as the first argument *)
      | SObjCreate(typ, sexprl) -> obj_create_gen llbuilder typ sexprl
      | SNoexpr -> L.const_int i32_t 0

and binop_gen llbuilder sexpr1 op sexpr2 typ =
      let lexpr1 = sexpr_gen llbuilder sexpr1 in
      let lexpr2 = sexpr_gen llbuilder sexpr2 in

      let typ1 = get_type_from_sexpr sexpr1 in
      let typ2 = get_type_from_sexpr sexpr2 in

      let int_ops expr1 binop expr2 = match binop with
              A.Add -> L.build_add expr1 expr2 "int_addtmp" llbuilder
            | A.Sub -> L.build_sub expr1 expr2 "int_subtmp" llbuilder
            | A.Mult -> L.build_mul expr1 expr2 "int_multop" llbuilder
            | A.Div -> L.build_sdiv expr1 expr2 "int_divop" llbuilder
            | A.Mod -> L.build_srem expr1 expr2 "int_modop" llbuilder
            | A.Veq -> L.build_icmp L.Icmp.Eq expr1 expr2 "int_eqtmp"
llbuilder
            | A.Vneq -> L.build_icmp L.Icmp.Ne expr1 expr2 "int_neqtmp"
                llbuilder
            | A.Less -> L.build_icmp L.Icmp.Slt expr1 expr2
"int_lesstmp"
                llbuilder
            | A.Leq -> L.build_icmp L.Icmp.Sle expr1 expr2 "int_leqtmp"
```

```
                llbuilder
            | A.Greater -> L.build_icmp L.Icmp.Sgt expr1 expr2
"int_greatertmp"
                llbuilder
            | A.Geq -> L.build_icmp L.Icmp.Sge expr1 expr2 "int_geqtmp"
                llbuilder
            | A.And -> L.build_and expr1 expr2 "int_andtmp" llbuilder
            | A.Or -> L.build_or expr1 expr2 "int_ortmp" llbuilder
            | _ -> raise(Failure("Unsupported operator for integers"))
      in

      let float_ops expr1 binop expr2 = match binop with
              A.Add -> L.build_fadd expr1 expr2 "flt_addtmp" llbuilder
            | A.Sub -> L.build_fsub expr1 expr2 "flt_subtmp" llbuilder
            | A.Mult -> L.build_fmul expr1 expr2 "flt_multop" llbuilder
            | A.Div -> L.build_fdiv expr1 expr2 "flt_divop" llbuilder
            | A.Mod -> L.build_frem expr1 expr2 "flt_modop" llbuilder
            | A.Veq -> L.build_fcmp L.Fcmp.Oeq expr1 expr2 "flt_eqtmp"
llbuilder
            | A.Vneq -> L.build_fcmp L.Fcmp.One expr1 expr2
"flt_neqtmp"
                llbuilder
            | A.Less -> L.build_fcmp L.Fcmp.Olt expr1 expr2
"flt_lesstmp"
                llbuilder
            | A.Leq -> L.build_fcmp L.Fcmp.Ole expr1 expr2 "flt_leqtmp"
                llbuilder
            | A.Greater -> L.build_fcmp L.Fcmp.Ogt expr1 expr2
"flt_greatertmp"
                llbuilder
            | A.Geq -> L.build_fcmp L.Fcmp.Oge expr1 expr2 "flt_geqtmp"
                llbuilder
            | _ -> raise(Failure("Unsupported operator for floats"))
      in

      match typ with
              A.Int -> int_ops lexpr1 op lexpr2
            | A.Float -> float_ops lexpr1 op lexpr2
            | A.Bool -> (match typ1, typ2 with
                A.Int, A.Int | A.Bool, A.Bool -> int_ops lexpr1 op
lexpr2
                | A.Float, A.Float -> float_ops lexpr1 op lexpr2
```

```
                  | _, _ -> raise(Failure("Cannot perform operations on
types "
                        ^ A.string_of_typ typ1 ^ " and " ^
A.string_of_typ typ2)))
            | _ -> raise(Failure("Unrecognized data type in binop"))

and unop_gen llbuilder unop sexpr typ =
      let unop_lval = sexpr_gen llbuilder sexpr in

      let build_unop op unop_typ lval = match op, unop_typ with
              A.Neg, A.Int -> L.build_neg lval "neg_int_tmp" llbuilder
            | A.Neg, A.Float -> L.build_fneg lval "neg_flt_tmp"
llbuilder
            | A.Not, A.Bool -> L.build_not lval "not_bool_tmp"
llbuilder
            | _ -> raise(Failure("Unsupported unop for " ^
A.string_of_uop op ^
                  " and type " ^ A.string_of_typ typ))
      in

      match typ with
              A.Int | A.Float | A.Bool -> build_unop unop typ unop_lval
            | _ -> raise(Failure("Invalid type for unop: " ^
A.string_of_typ
                  typ))

and cast_gen llbuilder to_typ sexpr =
      let lexpr = sexpr_gen llbuilder sexpr in
      let from_typ = get_type_from_sexpr sexpr in
      match from_typ with
              A.Bool -> (
                  match to_typ with
                    A.Bool -> lexpr
                  | A.Char -> L.build_zext lexpr i8_t "bool_char_cast"
llbuilder
                  | A.Int -> L.build_zext lexpr i32_t "bool_int_cast"
llbuilder
                  | A.Float -> L.build_uitofp lexpr f_t
"bool_float_cast"
                        llbuilder
                  | _ -> raise(Failure("Invalid cast from " ^
A.string_of_typ
```

```
                        from_typ ^ " to " ^ A.string_of_typ to_typ))
                )
            | A.Int -> (match to_typ with
                    A.Int -> lexpr
                  | A.Bool ->
                        let zero = L.const_int i32_t 0 in
                        L.build_icmp L.Icmp.Ne lexpr zero
"int_bool_cast" llbuilder
                  | A.Char -> L.build_trunc lexpr i8_t "int_char_cast"
llbuilder
                  | A.Float -> L.build_sitofp lexpr f_t
"int_float_cast" llbuilder
                  | _ -> raise(Failure("Invalid cast from " ^
A.string_of_typ
                        from_typ ^ " to " ^ A.string_of_typ to_typ))
                )
            | A.Float -> (match to_typ with
                    A.Float -> lexpr
                  | A.Char -> L.build_fptoui lexpr i8_t
"float_char_cast"
                        llbuilder
                  | A.Bool ->
                        let zero = L.const_float f_t 0.0 in
                        L.build_fcmp L.Fcmp.One lexpr zero
"float_bool_cast"
                              llbuilder
                  | A.Int -> L.build_fptosi lexpr i32_t
"float_int_cast" llbuilder
                  | _ -> raise(Failure("Invalid cast from " ^
A.string_of_typ
                        from_typ ^ " to " ^ A.string_of_typ to_typ))
                )
            | A.Char -> (match to_typ with
                    A.Char -> lexpr
                  | A.Bool ->
                        let zero = L.const_int i8_t 0 in
                        L.build_icmp L.Icmp.Ne lexpr zero
"char_bool_cast" llbuilder
                  | A.Int -> L.build_zext lexpr i32_t "char_int_cast"
llbuilder
                  | A.Float -> L.build_uitofp lexpr f_t
"char_float_cast"
```

```
                        llbuilder
                  | _ -> raise(Failure("Invalid cast from " ^
A.string_of_typ
                        from_typ ^ " to " ^ A.string_of_typ to_typ))
                  )
            | _ -> raise(Failure("Invalid cast from " ^ A.string_of_typ
from_typ
                  ^ " to " ^ A.string_of_typ to_typ))

(* Assignment instruction generation *)
and assign_gen llbuilder sexpr1 sexpr2 typ =

      let lhs, is_obj_access = match sexpr1 with
              SId(id, _) -> id_gen llbuilder id false, false
            | SSeqAccess(lst_sexpr, index,  _) ->
                  seq_access_gen llbuilder lst_sexpr index true, true
            | SFieldAccess(id, field, _) -> field_access_gen llbuilder
id field
                  false, true
            | _ -> raise(Failure("Unable to assign."))
      in

      let rhs = match sexpr2 with
              SId(id, typ) -> (match typ with
                    A.ClassTyp(_) -> id_gen llbuilder id false
                  | _ -> id_gen llbuilder id true)
            | SFieldAccess(id, field, _) -> field_access_gen llbuilder
id field
                  true
            | _ -> sexpr_gen llbuilder sexpr2
      in

      let rhs = match typ with
              A.ClassTyp(_) ->
                  if is_obj_access then
                        rhs
                  else
                        L.build_load rhs "tmp" llbuilder
            | A.Null_t -> L.const_null (get_llvm_type typ)
            | _ -> rhs
      in
```

```
      ignore(L.build_store rhs lhs llbuilder);
      rhs

and lst_create_gen llbuilder typ sexprl =
      let e = List.hd sexprl in
      let t = get_llvm_type typ in
      let size = sexpr_gen llbuilder e in
      let size_t = L.build_intcast (L.size_of t) i32_t "tmp" llbuilder
in
      let size = L.build_mul size_t size "tmp" llbuilder in
      let size_real = L.build_add size (L.const_int i32_t 1) "tmp"
llbuilder
      in
      let arr = L.build_array_malloc t size_real "tmp" llbuilder in
      let arr = L.build_pointercast arr t "tmp"
      llbuilder in
      (* store this dimension *)
      let arr_len_ptr = L.build_pointercast arr (L.pointer_type i32_t)
"tmp"
            llbuilder in
            ignore(L.build_store size_real arr_len_ptr llbuilder);
            arr

(* Access a list *)
and seq_access_gen llbuilder lst_sexpr index is_assign =
      let lst = sexpr_gen llbuilder lst_sexpr in
      let sindex = sexpr_gen llbuilder index in
      let sindex = L.build_add sindex (L.const_int i32_t 1) "list_index"
            llbuilder in
      let _val = L.build_gep lst [| sindex |] "list_access" llbuilder in
      if is_assign then
            _val
      else
            L.build_load _val "list_access_val" llbuilder

and field_access_gen llbuilder id rhs isAssign =
      let check_id id =
            match id with
             SId(s, _) -> id_gen llbuilder s false
            (* array *)
            | _ -> raise(Failure("expected access lhs to be an id"))
      in
```

```
      let rec check_rhs par_exp par_typ rhs =
            let class_name = A.string_of_typ par_typ in
            match rhs with
            SId(s, d) ->
                  let field_name = (class_name ^ "." ^ s) in
                  let field_index = Hash.find class_fields field_name
in
                  let _val = L.build_struct_gep par_exp field_index s
llbuilder in
                  let _val = match d with
                          A.ClassTyp(_) ->
                              if isAssign then L.build_load _val s
llbuilder
                              else _val
                        | _ ->
                              if isAssign then L.build_load _val s
llbuilder
                              else _val
                  in
                  _val
            | SCall(fname, exprl, ftyp) -> call_gen llbuilder fname
exprl ftyp
            | SFieldAccess(e1, e2, _) ->
                        let e1_typ = Se.get_type_from_sexpr e1 in
                        let e1 = check_rhs par_exp par_typ e1 in
                        let e2 = check_rhs e1 e1_typ e2 in
                        e2
            | _ -> raise(Failure("illegal rhs type for access"))
      in
      let id_typ = Se.get_type_from_sexpr id in
      let id = check_id id in
      let rhs = check_rhs id id_typ rhs in
      rhs

and obj_create_gen llbuilder typ sexprl =
      let f = func_lookup (A.string_of_typ typ) in
      let params = List.map (sexpr_gen llbuilder) sexprl in
      let obj = L.build_call f (Array.of_list params) "tmp" llbuilder
in
      obj
```

```
and cast_malloc_gen llbuilder sexprl stype =
      let cast_malloc llbuilder lhs newTyp =
            match newTyp with
                  A.ClassTyp(c) -> let obj_llvm_typ = get_llvm_type
(A.ClassTyp(c)) in L.build_pointercast lhs obj_llvm_typ "tmp" llbuilder
            | _ as c -> raise(Failure("Cannot cast to " ^
A.string_of_typ c))
      in
      let sexpr = List.hd sexprl in
      let lhs = match sexpr with
              SId(id, _) -> id_gen llbuilder id false
            | SFieldAccess(e1, e2, _) -> field_access_gen llbuilder e1
e2 false
            | _ -> sexpr_gen llbuilder sexpr
      in
      cast_malloc llbuilder lhs stype

and method_call_gen llbuilder obj_expr fname sexprl styp =
      match obj_expr with
      SId(_, obj_typ) ->
            let the_func = func_lookup fname in
            let match_sexpr se = match se with
                  SId(id, d) -> let isDeref = match d with
                          A.ClassTyp(_) -> false
                        | _ -> true
            in id_gen llbuilder id isDeref
      | se -> sexpr_gen llbuilder se in
            let lhs = match_sexpr obj_expr in
            let self_param = L.build_pointercast lhs
                  (get_llvm_type obj_typ) "tmp" llbuilder in
            let params = List.map match_sexpr sexprl in
                  (match styp with
                  A.Void -> L.build_call the_func (Array.of_list
                        (self_param :: params)) "" llbuilder
                  | _ -> L.build_call the_func (Array.of_list
                        (self_param :: params)) "tmp" llbuilder)
      | _ -> raise(Failure("unsupported chained method call"))

and func_call_gen llbuilder fname sexprl stype =
      let the_func = func_lookup fname in
      let params = List.map (sexpr_gen llbuilder) sexprl in
      match stype with
```

```
      A.Void -> L.build_call the_func (Array.of_list params) ""
llbuilder
            | _ -> L.build_call the_func (Array.of_list params) "tmp"
llbuilder

and call_gen llbuilder fname sexprl stype =
            match fname with
                  (* full list of built-in and linked functions just
for clarity*)
                    "print_string" | "print_int" | "print_float" |
"print_char" ->
                                print_gen llbuilder sexprl
                  | "malloc" -> func_call_gen llbuilder fname sexprl
stype
                  | "cast" -> cast_malloc_gen llbuilder sexprl stype
                  | "sizeof" -> sizeof_gen llbuilder sexprl
                  | _ -> func_call_gen llbuilder fname sexprl stype

and sizeof_gen llbuilder el =
      let typ = Se.get_type_from_sexpr (List.hd el) in
      let typ = get_llvm_type typ in
      let size = L.size_of typ in
      L.build_bitcast size i32_t "tmp" llbuilder

(* Helper method to generate print function for strings. *)
and print_gen llbuilder sexpr_list =
      let params = List.map (fun expr -> sexpr_gen llbuilder expr)
sexpr_list
      in
      let param_types = List.map get_type_from_sexpr sexpr_list in
      let get_format typ = match typ with
              A.Int -> "%d"
            | A.Float -> "%f"
            | A.String -> "%s"
            | A.Char -> "%c"
            | _ -> raise (Failure("cannot print type " ^
A.string_of_typ typ))
      in
      let format_str = List.fold_left (fun s t -> s ^ get_format t) ""
            param_types in
      let str = sexpr_gen llbuilder (SStringLit(format_str)) in
            let zero = L.const_int i32_t 0 in
```

```
            let s = L.build_in_bounds_gep str [| zero |] "tmp" llbuilder
in
            L.build_call (func_lookup "printf")
                  (Array.of_list (s :: params)) "print" llbuilder

and ret_gen llbuilder sexpr =
      match sexpr with
              SId(name, t) -> (
                  match t with
                        | A.ClassTyp(_) -> L.build_ret (id_gen
llbuilder name false)
                                 llbuilder
                        | _ -> L.build_ret (id_gen llbuilder name true)
llbuilder
              )
            | SFieldAccess(e1, e2, _) -> L.build_ret (field_access_gen
llbuilder
                  e1 e2 true) llbuilder
            | SNoexpr -> L.build_ret_void llbuilder
            | _ -> L.build_ret (sexpr_gen llbuilder sexpr) llbuilder

and if_gen llbuilder loop_stack if_sexpr if_sstmts elseifs else_sstmts
=

      (* Initial bb *)
      let start_bb = L.insertion_block llbuilder in
      let parent_func = L.block_parent start_bb in

      let if_bb = L.append_block context "if" parent_func in
      (* if bb *)
      let if_body_bb = L.append_block context "if_body" parent_func in

      (* Create if bb statements *)
      L.position_at_end if_body_bb llbuilder;
      ignore(sstmt_gen llbuilder loop_stack if_sstmts);

      let new_if_body_bb = L.insertion_block llbuilder in

      (* elseif bbs *)
      let rec make_elseif_bbs elseifs = match elseifs with
              [] -> []
            | head :: tail ->
```

```
                  let selseif = match head with
                          SElseif(sexpr, sstmt) -> (sexpr, sstmt)
                        | _ ->
                             raise(Failure("Unexpected non-elseif in
elseif list"))
                  in
                  let elseif_sexpr, elseif_sstmts = selseif in

                  (* elseif bb *)
                  let elseif_bb = L.append_block context "elseif"
parent_func in
                  let elseif_body_bb = L.append_block context
"elseif_body"
                        parent_func in

                  (* Create elseif bb statements *)
                  L.position_at_end elseif_body_bb llbuilder;
                  ignore(sstmt_gen llbuilder loop_stack elseif_sstmts);

                  let new_elseif_bb = L.insertion_block llbuilder in
                  (elseif_sexpr, elseif_bb, elseif_body_bb,
new_elseif_bb) ::
                        make_elseif_bbs tail
      in
      let elseif_bbs = make_elseif_bbs elseifs in
      let if_elseif_bbs = (if_sexpr, if_bb, if_body_bb, new_if_body_bb)
::
            elseif_bbs in

      (* else bb *)
      let else_body_bb = L.append_block context "else_body" parent_func
in

      (* Create else bb statements *)
      L.position_at_end else_body_bb llbuilder;
      ignore(sstmt_gen llbuilder loop_stack else_sstmts);

      let new_else_body_bb = L.insertion_block llbuilder in

      (* Merge if-elseif-else bbs *)
      let merge_bb = L.append_block context "merge" parent_func in
      L.position_at_end merge_bb llbuilder;
```

```
      (* else bb -> else llvalue *)
      let else_bb_val = L.value_of_block new_else_body_bb in

      (* Initial bb -> if bb *)
      L.position_at_end start_bb llbuilder;
      ignore(L.build_br if_bb llbuilder);

      (* Go to start bb and add conditional branch to next conditional
       * block *)
      let rec build_cond_brs if_elseif_bbs = match if_elseif_bbs with
              head :: next :: tail ->
                  let head_sexpr, head_bb, head_body_bb, _ = head in
                  let _, next_bb, _, _ = next in

                  (* Build expr for cond br, then build cond br *)
                  L.position_at_end head_bb llbuilder;
                  let head_lexpr = sexpr_gen llbuilder head_sexpr in
                  ignore(L.build_cond_br head_lexpr head_body_bb next_bb
                        llbuilder);

                  build_cond_brs (next :: tail)
            | head :: _ ->
                  let head_sexpr, head_bb, head_body_bb, _ = head in

                  L.position_at_end head_bb llbuilder;
                  let head_lexpr = sexpr_gen llbuilder head_sexpr in
                  ignore(L.build_cond_br head_lexpr head_body_bb
else_body_bb
                        llbuilder)
            | [] -> ()
      in
      build_cond_brs if_elseif_bbs;

      (* Create merge bb at end of if bb *)
      let rec build_merge_brs if_elseif_bbs = match if_elseif_bbs with
              head :: tail ->
                  let _, _, _, new_head_bb = head in

                  L.position_at_end new_head_bb llbuilder;
                  ignore(L.build_br merge_bb llbuilder);
```

```
                build_merge_brs tail
          | [] -> ()
      in
      build_merge_brs if_elseif_bbs;

      (* Create merge bb at end of else bb *)
      L.position_at_end new_else_body_bb llbuilder;
      ignore(L.build_br merge_bb llbuilder);

      (* Go to end of merge *)
      L.position_at_end merge_bb llbuilder;

      else_bb_val

and for_gen llbuilder loop_stack sexpr1 sexpr2 sexpr3 sstmts =
      let parent_func = L.block_parent (L.insertion_block llbuilder) in

      (* Build initialization expr *)
      ignore(sexpr_gen llbuilder sexpr1);

      (* bb's for body, step, condition, and exit *)
      let body_bb = L.append_block context "loop_body" parent_func in
      let step_bb = L.append_block context "loop_step" parent_func in
      let cond_bb = L.append_block context "loop_cond" parent_func in
      let exit_bb = L.append_block context "loop_exit" parent_func in

      let new_loop_stack = (step_bb, exit_bb) :: loop_stack in

      (* Init block -> cond *)
      ignore(L.build_br cond_bb llbuilder);

      (* Reorder blocks: bb, step, cond, exit*)
      let bb = L.insertion_block llbuilder in
      L.move_block_after bb step_bb;
      L.move_block_after step_bb cond_bb;
      L.move_block_after cond_bb exit_bb;
      ignore(L.build_br step_bb llbuilder);
      (* At exit bb, jump to step bb *)

      (* Build step *)
      L.position_at_end step_bb llbuilder;
      ignore(sexpr_gen llbuilder sexpr3);
```

```
      ignore (L.build_br cond_bb llbuilder);

      (* Build conditional branch to exit bb *)
      L.position_at_end cond_bb llbuilder;
      let cond_lexpr = sexpr_gen llbuilder sexpr2 in
      ignore(L.build_cond_br cond_lexpr body_bb exit_bb llbuilder);

      (* Build body bb stmts *)
      L.position_at_end body_bb llbuilder;
      ignore(sstmt_gen llbuilder new_loop_stack sstmts);
      ignore(L.build_br step_bb llbuilder);

      (*Continue building from loop exit *)
      L.position_at_end exit_bb llbuilder;

      L.const_null i32_t

and while_gen llbuilder loop_stack sexpr sstmts =
      for_gen llbuilder loop_stack (SIntLit(0)) sexpr (SIntLit(0))
sstmts

(* Branch to nearest loop exit *)
and break_gen llbuilder loop_stack =
      match loop_stack with
              head :: _ ->
                  L.build_br (snd head) llbuilder
            | [] -> raise(Failure("Break found in non-loop"))

(* Branch to nearest loop step *)
and continue_gen llbuilder loop_stack =
      match loop_stack with
              head :: _ ->
                  L.build_br (fst head) llbuilder
            | [] -> raise(Failure("Continue found in non-loop"))

(* Generates a local variable declaration *)
and local_var_gen llbuilder typ id sexpr =
      let lst, ltyp, flag = match typ with
              A.ClassTyp(classname) -> (L.build_add (L.const_int i32_t
0)
                  (L.const_int i32_t 0) "nop" llbuilder),
                  find_global_class classname, false
```

```
            | _ -> (L.build_add (L.const_int i32_t 0) (L.const_int
i32_t 0)
                  "nop" llbuilder), get_llvm_type typ, false
      in

      let alloc = L.build_alloca ltyp id llbuilder in
      Hash.add local_values id alloc;
      if flag = false then (
            let lhs = SId(id, typ) in
            match sexpr with
                        SNoexpr -> alloc
                        | _ -> assign_gen llbuilder lhs sexpr typ )
      else
            let lhs_gen = id_gen llbuilder id false in
            ignore(L.build_store lst lhs_gen llbuilder);
            alloc

(* Declare all built-in functions. This should match the functions
added in
 * semant.ml
 *)
let construct_library_functions =
      let print_type = L.var_arg_function_type i32_t [| L.pointer_type
i8_t |]
      in
      let _ = L.declare_function "printf" print_type codegen_module
      in
            let malloc_type = L.function_type (str_t) [| i32_t |] in
            let _ = L.declare_function "malloc" malloc_type
codegen_module
            in
      ()

let init_params func formals =
      let formal_array = Array.of_list (formals) in
      Array.iteri (fun index value ->
            let name = formal_array.(index) in
                        match name with A.Formal(_, n) ->
            L.set_value_name n value;
            Hash.add local_params n value; ) (L.params func)

let func_stub_gen sfdecl =
```

```
      let param_types = List.rev (
            List.fold_left
                  (fun x -> (function A.Formal(t, _) -> get_llvm_type t
:: x))
                  [] sfdecl.sformals
            )
      in
      let stype = L.function_type (get_llvm_type sfdecl.stype)
(Array.of_list
            param_types)
      in
      L.define_function sfdecl.sfname stype codegen_module

let func_body_gen sfdecl =
      Hash.clear local_values;
      Hash.clear local_params;
      let func = func_lookup sfdecl.sfname in
      let llbuilder = L.builder_at_end context (L.entry_block func) in
      let _ = init_params func sfdecl.sformals in
      (* Stack of loop blocks *)
      let loop_stack = [] in
      let _ = sstmt_gen llbuilder loop_stack (SBlock(sfdecl.sbody)) in
            if sfdecl.stype = A.Void then
                  ignore(L.build_ret_void llbuilder);
            ignore(L.build_unreachable llbuilder)

let class_gen s =
            let typ = L.named_struct_type context s.scname in
            Hash.add global_classes s.scname typ;

      let typ = Hash.find global_classes s.scname in
      let typ_lst = List.map (function
            A.ObjVar(t, _, _) | A.ObjConst(t, _, _) ->
                  get_llvm_type t) s.scbody.sfields in
      let name_lst = List.map (function
            A.ObjVar(_, n, _) | A.ObjConst(_, n, _) ->
                  n) s.scbody.sfields in
      (* adding i32_t and key *)
      let typ_array = (Array.of_list typ_lst) in
            List.iteri (fun i name ->
                  let full_name = s.scname ^ "." ^ name in
                  Hash.add class_fields full_name i;
```

```
            ) name_lst;
            L.struct_set_body typ typ_array true;

      let _ = List.map (fun f -> func_stub_gen f) s.scbody.smethods in
      let res = List.map (fun f -> func_body_gen f) s.scbody.smethods
in
            res

let translate sprogram =
      let _ = construct_library_functions in
      let classes = List.rev sprogram.classes in
      let _ = if (List.length sprogram.classes > 0) then List.map
            (fun s -> class_gen s) classes else [] in
      let _ = List.map (fun f -> func_stub_gen f) sprogram.functions in
      let _ = List.map (fun f -> func_body_gen f) sprogram.functions in
      codegen_module
```

decaf.ml

```
(* Top-level of the DECAF compiler: scan & parse the input,
 * check the resulting AST, generate LLVM IR, and dump the module. *)
module L = Llvm

type action = Ast | LLVM_IR | Compile

let _ = let action = if Array.length Sys.argv > 1 then
    List.assoc Sys.argv.(1) [ ("-a", Ast);
                              ("-l", LLVM_IR);
                              ("-c", Compile) ]
    else Ast in
    let lexbuf = Lexing.from_channel stdin in
    let ast = Parser.program Scanner.token lexbuf in
    let sast = Semant.check ast in
    match action with
      Ast -> print_string (Ast.string_of_program ast)
    | LLVM_IR -> print_string (L.string_of_llmodule (Codegen.translate
sast))
    | Compile -> let m = Codegen.translate sast in
      Llvm_analysis.assert_valid_module m;
      print_string (Llvm.string_of_llmodule m)
```