The Coral Programming Language

Programming Languages and Translators Final Report
Fall 2018

Jacob Austin, Matthew Bowers, Rebecca Cawkwell, Sanford Miller
ja3067 mlb2251 rgc2137 skm2159

December 20, 2018

Contents

1 Introduction

The Coral Programming Language is an imperative and functional scripting language inspired by Python, with optional static typing used to ensure type safety and optimize code. It roughly aims to be to Python as TypeScript is to JavaScript. The basic syntax is identical to Python, and any valid Coral program can also be run by a Python interpreter. Coral also uses the newly introduced optional static typing syntax found in Python 3.7, allowing variables and function declarations to be tagged with specific types. Unlike in Python, these types are not merely cosmetic. The Coral typing system will be checked and enforced at compile time and runtime, preventing errors due to invalid type usage in large codebases, and will also be used to optimize the performance of the language where possible. The goal is to create a language that is as convenient as Python and as safe as an explicitly typed language with superior performance to existing scripting languages.

1.1 Background

Several attempts have been made in the past twenty years to develop optimizing compilers or more efficient interpreters for Python. For example, the Numba project uses decorators to compile designated Python functions to LLVM code using a Just-in-Time Compiler (or JIT). Other languages like Nim have embraced the simplicity of the Python model, but created a new programming language which violates some of Python's language conventions in exchange for more efficient optimization. While this approach allows broader classes of optimization, we wanted to develop a language which could be used an an alternative Python compiler for existing Python code instead of requiring new syntax.

Some theoretical work on the typing system used by Coral was done by Jeremy Siek at Indiana University, who developed a theory of gradual typing in the Lambda Calculus. Some of our type inference system was derived from this model, although much of it was developed from scratch for the Coral language.

1.2 Goals

1.2.1 Familiarity

Coral's syntax is identical to Python's, with all its usual convenience. Python programmers can simply copy their code into the Coral compiler, and they can add type annotations which can be used to help performance and improve safety. Python is already an easy-to-learn language, and Coral adds no additional difficult to the learning curve. Coral also imposes no further restriction on the kinds of functions which can be run.

1.2.2 Type Safety

Coral provides type safety where desired. Type specifiers on variables and functions enforce correct usage in large code bases. Sometimes it is helpful to have more flexibility with what kinds of types a variable can take or what kinds of arguments can be passed to it. Coral allows that. But if you have a large code-base with functions or arguments of known type, it can be helpful to explicitly declare those types and have them guaranteed by the compiler and runtime environment. Coral provides that too. Coral generally strives to catch as many type errors as possible at compile time, but will fall back to runtime checks where needed.

1.2.3 Code Optimization

Because of Coral's type inference system and assisted by type annotations, Coral is able to compile code to far more efficient machine code that Python. Even without type-hints, Coral can often optimize code to be nearly as fast as equivalent C code, and type hints can allow it to overcome ambiguities in type-inference that would otherwise prevent Coral from optimizing fully. This natural interplay between typed code and optimization is at the heart of the Coral language.

1.2.4 Seamless Interplay between Typed and Untyped Code

The core philosophy of Coral is that you dont pay a penalty if you dont type your code, and typed functions can be called with untyped arguments and vice versa. This preserves all the convenience of Python, while giving the programmer the freedom to be more explicit where desired.

2 Coral Tutorial

Thank your for downloading Coral! The following tutorial provides instructions to set up your environment and how to compile Coral programs.

2.1 Set up your Environment

Before installing Coral, ensure that you have installed OCaml and LLVM. GCC must also be installed. Ocaml and LLVm can be installed on Mac OS with

```
> brew install opam
> brew install llvm
> opam install llvm
```

and on Ubuntu with

```
> sudo apt-get install opam
> sudo apt-get install llvm
> opam install llvm
```

Other Linux distributions can be installed similarly using your distribution's package manager. If the above fails, try instead running the following on Mac OS (or the equivalent on Linux):

```
> brew install llvm
> opam depext conf-llvm.6.0.0
> opam install llvm
> export PATH=/usr/local/opt/llvm@6/bin:$PATH
```

To add the llvm installation to your $PATH variable permanently, you can copy the last line above to your .bashrc or .bash_profile file.

2.2 Downloading and Building Coral

First, clone the Coral Repository with the following command:

```
> git clone https://github.com/ja3067/Coral.git
```

Next, compile the Coral compiler using the following commands. This will automatically run Coral's test suite as well.

```
> cd Coral
> make
```

2.3 Writing and Compiling a Simple Program: GCD

Once Coral has been compiled, you can begin writing your own Coral programs. Use the Language Reference Manual to learn syntax as well as the limitations of the Coral language.

The following sample code is an implementation of gcd, a simple program in Coral.

```
def gcd(a, b):
    while a != b:
        if a < b:
            b = b-a
        else:
            a = a-b
    return a

a = 1234342213
b = 334232
print(gcd(a,b))
```

Save this code into a gcd.cl file. To compile and run this code, type the following commands into your terminal:

```
> ./coral.native gcd.cl
1019
```

Which coral.native runs the file directly, it also generates a main files and source.ll and source.s which contain the llvm and assembly code respectively.

2.4 Using The Coral Interpreter

Coral also has a build-in interpreter. To use the interpreter, simply run ./coral.native in the Coral directory. This will open an interactive window like the OCaml or Python interpreter in which you can run any valid Coral programs. The following is an example of gcd code run in the interpreter:

```
> ./coral.native
Welcome to the Coral programming language!
>>> def gcd(a, b)
...     while a ! = b:
...         if a > b:
...             a = a-b
...         else:
```

```
...             b = b-a
...     a
...
>>> gcd(5,10)
5.
>>>
```

2.5 Debugging Options

Both the compiler and interpreter support several debugging modes which can be useful for visualizing and understanding the type inference system of the Coral language. To run Coral with this debugging information displayed, add the -d flag to any interpreter or compiler calls.

There is also a -no-compile flag which, when specified, causes the compiler to run semantic checking (with or without the debug flag) without compiling or running the code.

3 Language Reference Manual

3.1 Lexical Conventions

3.1.1 Comments

Coral has both single-line and multi-line comments. Any tokens following a # symbol are considered part of a single-line comment and are not lexed or parsed.

```
x = 25 # x is in inches
# x is the average length of a coral snake
```

Multiline comments begin and end with triple quotes.

```
"""
Coral snakes are known for their red, white and black banding
"""
```

Within triple quotes, single or double quotes can be arbitrarily nested.

3.1.2 Identifiers

Valid identifiers are made from roman alphabetic characters and digits, as well as underscores. An identifier must begin with a letter, can contain an underscore and cannot be a Coral keyword.

```
# valid identifiers
pinkPython
GardenSnakeCount
snake_length
babysnake4

# invalid identifiers
```

```
25coral
5
```

3.1.3 Operators

Coral uses the following reserved operators:

```
+  -  <  >  <=  >=  !=  ==  *  /  = += -= *= /=
```

These operators will be explained in subsequent sections.

3.1.4 Keywords

Keywords are reserved identifiers. They cannot be used as ordinary identifiers for other purposes. Corals keywords are:

```
not if else elif assert pass continue break class for while def int float str bool
func int[] float[] str[] bool[] and or in return is None range type print import
global await from as nonlocal async yield raise except finally is
lambda try with True False
```

To ensure compatibility of Coral with Python, using unimplemented Python keywords returns a compile-time error. All Python keywords are captured by the Coral lexer, but some will throw RuntimeErrors. Those unimplement keywords are currently: global, await, from, as, nonlocal, async, yield, raise, except, finally, is, lambda, try, with, break, continue, class, elif, assert, pass.

3.1.5 Indentation

Coral uses indentation (tabs) to determine the grouping of statements. A given lines indentation is determined by the number of tabs preceding the first character. Statements are grouped by indentation level. Control flow keywords like if, else, for, and while must be followed by a colon, and the subsequent lines included in that control flow must be indented at the same level, unless a further control flow keyword is used. This behavior is identical to Python. No line may be indented more than one level beyond the current indentation level.

```
for i in [1, 2, 3]:
        print(i)

if x == 3:
        x = 4

while x < 3:
        if x < 5:
                return x
        else:
                return x + 1
```

3.1.6 Separators

Coral uses parentheses to override the default precedence of expression evaluation, semicolons to separate two consecutive expressions on the same line, tabs to denote control flow, and newlines to separate statements.

```
x = (a + b) * c # overrides default operator precedence
x = 3; y = x + 1 # allows two expressions in one line
if x == 3:
    print(x) # control flow
```

3.1.7 Literals

Literals represents strings or one of Coral's primitive types: float, char, int, and boolean.

Float Literals

A float literal is a number with a whole number, optional decimal point, a fraction, and an exponent.

```
(([0-9]+\.[0-9]*)|([0-9]*\.[0-9]+))((e|E)(\+|-)?[0-9]+)?|[0-9]+((e|E)(\+|-)?[0-9]+))
```

Examples of float literals:

```
25
2.5
0.000407
24.
1e+3
12.6E-2
```

String Literals

A string literal is a sequence of characters enclosed in single or double quotation marks, i.e. abcdefghijklmnopqrstuvwxyz. The matching regex is

```
("[^"'\\]*(\\.[^"\\]*)*")|('[^"'\\]*(\\.[^'\\]*)*')
```

Example of string literals:

```
"Hello world"
'Here are 4 words'
```

Int Literals

An integer literal is any sequence of integers between 0 and 9. The matching regex is [0-9]+.

Boolean Literals

Boolean types represent true and false. They are represented in Coral by the True and False keywords.

3.2 Data Types

Coral represents all pieces of data as either an object or a primitive.

3.2.1 Primitives

Primitives are series of bytes of some fixed length, and there are four primitive types in Coral:

int (4 bytes, 2's complement)

float (8 bytes, IEEE standard double)

char (1 byte, ASCII)

bool (1 byte, 00000001 for true, 00000000 for false)

Note that there is no separate double type.

3.2.2 Objects

Any piece of data that can't be inferred to be one of the primitive types is represented as an object. An object has an associated type and an associated value. The value can contain primitives and/or references to other objects. How to interpret the content of the value of an object is determined by its type. References are completely under the hood and not user-accessible as pointers.

3.2.3 Mutability

The primitive objects in Coral are immutable, including ints, floats, and booleans. Strings are not primitives, but are also immutable. All operations which modify these objects actually return new objects. All user defined types, as well as those in the standard library, are mutable, in the sense that modifying them does not overwrite the underlying object. Assigning to any variable will still overwrite its underlying object, i.e. x = 3; x = 4 assigns an integer literal with value 3 to the variable x, and then assigns a different integer literal with the value 4 to the same variable. Likewise, x = [1, 2, 3]; x = [4, 5, 6] will assign one object of type list to x, and then assign a different object to it. On the other hand, x = [1, 2, 3], x[1] = 4 will modify the underlying data associated with the variable x, returning x = [1, 4, 3].

3.2.4 Standard Library Types

Coral provides two built-in types, lists and strings. These will be expanded on in a later section, but Lists are implemented as static arrays and Strings are implemented as lists. Coral strings are immutable and support a rich variety of operations. They are fully memory safe. Likewise, lists are arrays which can contain objects of any type and can be modified with functional or imperative syntax.

```
x = "Hello" # this is a string
y = "World"
z = x + " " + y # z is now "Hello World"

x = [1, 2, "h"]
x[2] # returns "h"
```

3.2.5 None

None is keyword that returns a reference to a predetermined object of a special type. There is only ever one object of this type.

3.2.6 Memory Model

In Coral, with a few exceptions due to typing optimizations, all names are simply identifiers used to look up data in a hashtable.

All function calls are "pass by name", and do not involve copying data. Thus the user does not have to worry about passing an object "by reference" or returning "by reference" or "by value". Thus, for example

```
x = [1, 2, 3]
y = x
y[1] = 5 # x and y are now both [1, 5, 3]
```

or similarly

```
def foo(x):
    x[1] = 5

y = [1, 2, 3]

foo(y) # y is now [1, 5, 3]
```

On the other hand, assignment always acts as an assignment operator, and will not change the value of a bound variable, i.e.

```
def foo(x):
    x = 5 # assigns a new value to x in the local scope

y = 3

foo(y) # y is still 3
```

This behavior is universally consistent with Python.

3.3 The Coral Type System

3.3.1 Overview

The Coral language uses a gradual typing system which combines type inference with dynamic typing. Even in the absence of static type hints, the Coral language makes an effort to infer types, optimize code, and raise errors where invalid types have been used. In the presence of explicit types, type checking can be performed, improving the safety of the code.

All explicit type hints are guaranteed, and errors will be raised for any invalid type usage at compile or runtime. While Coral strives for a relatively error-free runtime experience, as a language without mandatory static typing, many errors cannot be caught at compile time. These issues will still be caught by the runtime environment, and the execution of the program will abort when these are detected. This behavior can be enabled or disabled at compile time. When types cannot be inferred, Coral will behave exactly like Python as a dynamically typed language with generic objects passed to all functions and hash-lookups performed at runtime to determine which function to execute.

3.3.2 Explicit Typing

Variable declarations, formal parameters, and return values can be marked with explicit types, allowing for compile-time and runtime type-checking and enabling the compiler to optimize performance. Typed variables are denoted with a colon followed by a type, like int, float, str, or char.

```
x : int = 3
x : str = "Hello World"
x : char = 'a'
x : char = "a" # also works
```

Once an identifier has been associated with a type, it cannot be assigned to an object of a different type. Hence,

```
x = 5 # x is an int for the moment
x = "not anymore" # note that this is legal Python (and Coral)

x : int = 5 # x is an int forever and ever
x = "this isn't allowed in Coral"
x = 4 # but this is allowed
```

3.3.3 Optimization

When the compiler can infer that a piece of data will be both immutable and a consistent type, it unboxes the data, converting it to a primitive instead of a generic object with type determined at runtime.

As much as possible, typed code does not behave fundamentally differently from untyped code, even if that leads to a performance penalty relative to pure C. For example (list) index bounds will still be checked at runtime, unlike in C. This means the runtime performance wont be quite as optimized as C, but removing these runtime checks when typing code would be confusing for the user.

3.3.4 Typed Lists

Lists are a built-in class that stores a series of pieces of data in an array. When every element of a list can be inferred to be of the same type and that type is a primitive type, usually the array is converted from an array of references, which can point to objects of various types, to an array of primitives, which must all be of the same type. Lists don't behave exactly like C arrays; Python-style conveniences like bounds-checking remain to ensure Python memory safety remains intact.

It is difficult for the compiler to determine that every element in a list is the same type, even if most or all non-list variables in a program are explicitly typed, so if object-to-primitive conversion in a list is desired, best practice is to type it explicitly.

```
my_list = [4, 3, "1"] # my_list contains an array of references (to objects of different types)

my_list = [4, 3, 1] # still array of references, which all just happen to be of the same type

my_list : int[] = [4, 3, 1] # array of ints
```

Note: when you access an item in a list, de-referencing happens under the hood and doesn't require any action on the part of the programmer.

3.3.5 Function Typing

A function can be declared with or without explicit typing. Functions with explicit typing have their typing behavior absolutely guaranteed, either at compile or runtime. Coral makes every effort to check the types of function calls at compile time, but if that is not possible, they will fail when the code is executed at runtime. A function in which every type can be inferred can be compiled to efficient machine code. Thus this explicit typing is highly recommended for bottleneck code.

For example, this function takes two integer arguments and returns a string:

```
def foo(x : int, y : int) -> str:
    if x == 3:
        return "hello"
    elif y == 3:
        return "goodbye"
```

This function can be compiled to efficient code as a result of the copious type hints. It will fail if called with invalid arguments, i.e.

```
foo(3, "hello") # fails
x : int = foo(3, 4) # fails
```

3.4 Statements and Expressions

3.4.1 Statements

A Coral program is made up of a list of statements (stmt). A statement is one of the following:

```
Expression
Function definition
Return statement
If statement
If-else statement
For loop
Range loop
While loop
Assignment
```

Blocks of statements can be enclosed using indentation as discussed earlier.

If-Else Statements

If statements consist of a condition (an expression) and a series of statements. The series of statements is evaluated if the condition evaluates to True. If the condition evaluates to False, either the program continues or an optional else clause is executed. In pseudocode, this is:

```
if condition:
        # series of statements
```

```
else:
    # other series of statements
```

While Statements

Similar to a for statement, a while statement consists of a condition and a series of statements. The statements are repeatedly evaluated as long as the condition remains True before each repetition.

```
while condition:
        # series of statements
```

For Statements

For statements in Coral iterate over lists. They consist of a looping variable, an instance of a list to be looped over, and a series of statements. The series of statements is evaluated for each item in the list in order, where the looping variable is assigned to the first element of the list for the first iteration, the second for the second iteration, etc. Note that assigning over the looping variable will not change the original variable in the list.

```
for x in some_list:
        print(x)
        x = 1 # this will not affect some_list
```

Range Statements

Range statements are a variant of for loops iterating over a range operator. Whereas in Python, range is implemented as an iterator, in Coral this sort of operation is simply syntactic sugar for a while loop which can be optimized more readily that a for each loop over a large list object. For example:

```
for x in range(10000):
        print(x)
        x = 1 # this will not affect some_list
```

This can be highly efficient, and its use is encouraged over true for each list iteration where possible.

3.4.2 Expressions and Operators

Expressions are parts of statements that are evaluated into expressions and variables using operators. These can be arithmetic expressions which includes an operand, an arithmetic operator, an operand, and a function call, which calls a function and returns a value.

Unary Operators

Coral has two types of operator: uop and operator.

Uops are unary operations that act on an expression. In Coral, the unary operators are NEG and NOT, i.e. ”-” and ”not”.

NEG negates an integer or floating point literal, as in

```
x = -5 # represents the negative number 5
```

NOT represents negation of a boolean expression, but can also be applied to integers and floating point numbers, where any non-zero number is considered to be True, and False otherwise.

```
a = True
b = not a # b equals False
c = not a # c equals False
c = not 5 # c equals False
```

Binary Operators

The following list describes the binary operators in Coral. All of these operators act on two expressions:

```
| Binop of expr * operator * expr
```

Unless otherwise stated, they act only on primitives.

1. Assignment Operator

   The assignment operator stores values into variables. This is done with the = symbol. The right side is stored in the variable on the left side. This can be applied to any type.

   ```
   x = 3.6 # 3.6 is stored in the variable x
   ```

2. Arithmetic Operator

   (a) Addition is performed on two values of the same type. This can be applied to strings.

   ```
   5 + 10 # Evaluates to 15
   15.1 + 14.9 # Evaluates to 30
   "hello" + "world" # Evaluates to "helloworld"
   ```

   The Coral language also permits automatic type conversion between integers and floating point numbers. For example

   ```
   5 + 10.0 # Evaluates to 15.0
   15.1 + 15 # Evaluates to 30.1
   ```

   (b) Subtraction is performed on two values of the same type, again with the possibility for automatic type conversion between integers and floats.

   ```
   5-10 # Evaluates to -5
   10.5 - 5.4 # Evaluates to 5.1
   ```

   (c) Multiplication is performed on two values of the same type, again with the possibility for automatic type conversion between integers and floats.

   ```
   5 * 5 # Evaluates to 25
   20.0 * .25 # Evaluates to 5.0
   ```

   (d) Division is performed on two values of the same type, again with the possibility for automatic type conversion between integers and floats.

```
10 / 2 # Evaluates to 5
10.8 / 2 # Evaluates to 5.4
```

3. Relational Operators

   Relational operators determine how the operands relate to another. There are two values as inputs and returns either true or false. These are the ==, !=, ¡, ¡=, ¿, ¿=.

```
x = 1
y = 2
z = (x > y) # Evaluates to false
z = (x < y) # Evaluates to true
x == y # Evaluates to false
x != y # Evaluates to true
```

4. Assignment Operators

   Assignment operators include the usual = operator, but also operators like +=, -=, *=, /=, and **= which perform arithmetic expressions on their arguments and then perform assignment. These arithmetic assignment operators are purely syntactic sugar for true addition, and is not a separate operator. For example:

```
x = 1
y = 2
x += y
x -= y
x *= y
x /= y
```

3.4.3 Operator Precedence

The following is an operator precedence table for unary and binary operators, from lowest to highest precedence.

Operator	Meaning	Associativity
;	Sequencing	Left
=	Assignment	Right
.	Access	Left
or	Or	Left
and	And	Left
= !=	Equality / inequality	Left
>=<>= <=	Comparison	Left
+ -	Addition / Subtraction	Left
* /	Multiplication / Division	Left
not	Negation / Not	Right

3.4.4 Functions

A function is a type of statement. It takes in a list of arguments and returns one value. The body of a function is delimited by indentation as described in the first section. Functions can be written either

specifying or not specifying the return type. An example of these two types of function declarations are as follows:

```
def f_not_spec(x,y):
        returns x + y

def f_spec(x : int, y : int) -> int:
        if x == 1:
        return x
    else:
        return x + y
```

3.4.5 Function Calls

Functions are called using their identifier and arguments inside parentheses. For example:

```
f_not_spec(1,2)  # function call which evaluates to 3
f_spec(1,2)  # function call which evaluates to 1
```

If the function is called with invalid arguments, it will either fail to compile or fail at runtime with a TypeError.

Variable Assignment from Functions

A function may be called as the right-hand side of a variable assignment. The variable would be assigned to the return value of the function.

```
example = f_not_spec(1,2) # 3 would be assigned to example
```

3.5 Standard Library

3.5.1 Lists

Coral has a built-in list data structure with dynamic length that can hold objects of arbitrary type. They behave identically to Python lists, and support the following operations:

Method/operator	Behavior
lt[n]	returns the nth element of the list

3.5.2 Strings

Strings are implemented as a list of char in Coral, which supports indexing and a variety of useful methods for handling text-based data. Strings are immutable, so any operation on an existing string will return a new string.

Method/operator	Behavior
str[n]	returns the nth character in str
str1 + str2	returns the concatenations of str1 and str2
str.upper()	returns a string with all characters uppercase
str.lower()	Returns a string with all characters lowercase

3.5.3 print()

print(x) sends a string representation of x to standard output. The nature of that representation depends on the type of x. If x is one of the four primitive types, there is a built-in print for it, e.g print(3) invokes print(x : int) → str. Strings and lists do not currently have a print function, but we are currently working on implementing that feature as well.

3.5.4 Casting

Python automatically casts between certain types, and this is done quite liberally. In our case, we would like to have these features, but they are not yet implemented. Currently, integer and floating point literals cannot be converted from one to the other, and will throw errors if used together. For example, at the moment,

```
3.0 * 4 # throws an error
3 * 4 # valid
3. * 4 # valid
4.0 / 3 # throws an error
```

3.6 Exceptions

3.6.1 Compile Time Exceptions

Coral cannot be fully type inferred at compile time while retaining the type flexibility that exists in Python. At compile time Coral checks for invalid assignments to explicitly types variables and invalid argument and return types for functions and operators. These examples are also caught in the interpreter. For example:

```
>>> def foo() -> int:
...     return "hello"
...
STypeError: invalid return type
>>> def add(x : int[]): ...     sum = 0
...     for i in x:
...         sum += i
...     return sum
...
>>> print(add([1,2,3]))
6
>>>print(add([1.0,2.0, 3.0]))
STypeError: invalid type assigned to x
```

This also checks invalid number of arguments in function calls and invalid arguments to binary and unary operations.

Our compile time errors all begin with S, as in STypeError, SSyntaxError, and SNotImplemented-Error.

3.6.2 Runtime Exceptions

Runtime exceptions only occur on types that aren't inferrable. Coral checks for invalid assigned, invalid argument types, initialization and list bounds.

For example, the following returns a "RuntimeError: unsupported operand type(s) for binary *" error:

```
1  def dynamic():
2      if x == 3:
3          return 3
4      else:
5          return "hello"
6  x = 3
7  print(dynamic() * dynamic())
8  x = 4
9  print(dynamic() * dynamic())
```

RuntimeErrors are only inserted where the compile-time semantic checking is unable to infer the types. This prevents much overhead from this sort of type checking.

4 Project Plan

4.1 Our Development Process

4.1.1 Planning

In order to complete Coral in a timely fashion, we had team meetings approximately twice a week. One of these meetings was on Tuesday, during which we met with our TA, Lauren Arnett. We would show Lauren updates to our language as well as establish what we hoped to show her the following week. These meetings were an opportunity to ask questions and receive feedback as well as support the continued accomplishment of Coral development deadlines.

Our group also met weekly either on Monday or Friday evenings. During these meetings we discussed the goals and needed steps to implement Coral. Any notes we took were on a shared Google Doc. After these meetings, there were set expectations of what each member was to accomplish by our next meeting time.

Our day to day communication happened over Facebook Messenger. This allowed us to work efficiently as someone was typically always active to answer questions or discuss an implementation point.

4.1.2 Specifications

The goal of Coral was to be compatible with Python. Our plan was to implement the basic skeleton of Python, including basic operators, functions and lists. The rest of our time was be spent on optimizing programs where everything was both immutable and fully type inferred.

4.1.3 Development

At the beginning of development, we had several large group meetings where we decided on the high-level design of our language. Our language was first developed on the semant side, where an interpreter was also created for testing. Then, as we approached the Hello World deadline, the codegen side of Coral was developed. Following a bare-bones set-up of codegen.ml, more operators, functions and other features were added to codegen by one team member and added to semant by another.

4.1.4 Testing

Features were tested before being merged with other branches of the repository. Tests featured both success and fail cases. These tests were then added to the testing suite that would automatically by tested when the Coral compiler was created.

4.2 Software Development Tools

We used the following programming and development environments when creating Coral:

- **Libraries and Languages**: Ocaml Version 4.07.0 including Ocamlyacc and Ocammllex and LLVM Version 6.0.1 was used.
- **Software**: Development was done in vim, Sublime and IntelliJ.
- **OS**: Development was done on OSX 10.13 and Ubuntu 18.04.1.
- **Version Control**: Github-Hosted Git Repository

4.3 Style Guide

We followed the following style guidelines while developing our compiler:

- Use of 2-space or 4-space for space indentation. The goal is to be consistent at least across the program.
- Use of under_scores for naming variables

4.4 Roles and Responsibilities

Team Member	**Responsibility**
Jacob Austin	Scanner, Parser, Semant
Matthew Bowers	Codegen- Objects, Functions and Optimization
Rebecca Cawkwell	Test Suite, Final Report
Sanford Miller	Codegen - Lists, Operator, For Loops

4.5 Project Timeline

4.6 Project Log

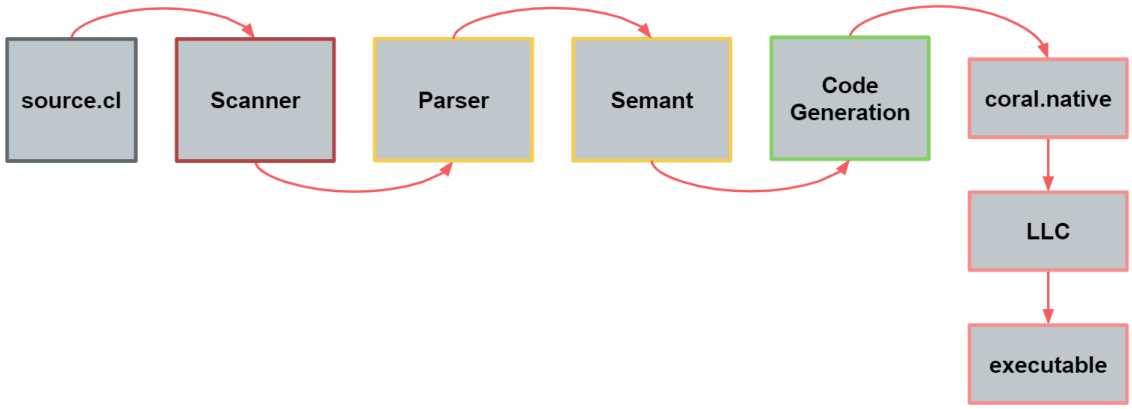
The following project log shows xxx commits from October x through December 19.

Insert git log

Date	Milestone
Sept 17	Initial Discussion to decide language
Sept 19	Language Proposal
Oct 7	Initial Parser and Interpreter
Oct 15	Language Reference Manual
Nov 13	First Program (gcd) Demo
Dec 11	Implement functions, operators, lists
Dec 13	All Day Group Meeting - merging of branches and optimization
Dec 18	Final Presentation and Report

5 Architectural Design

5.1 Block Diagram



5.2 Scanner

File: scanner.mll

The scanner takes in the program file and tokenizes it into tabs, literals, identifiers and keywords. Indentation characters (tabs in the case of Coral) are preserved because Coral, like Python, uses an indentation bases syntax system. Comments are removed during this stage. The scanner throws an error for unimplemented python keywords and syntactically invalid identifiers or literals.

Jacob implemented the scanner.

5.3 Parser and Semantic Checker

Files: ast.ml, parser.mly, semant.ml, sast.ml, utilities.ml

The parser is written in Ocamlyacc. First, it parses the lexing stream into a list which can then be used to extract indentation/tabs. This uses a special parser directive, tokenize, which simply returns the input stream as a list of tokens. Then the parser, using the ast, generates an Abstract Syntax Tree. The primary parser is relatively straightforward, and follows micro-C quite closely. It supports arbitrary assignment to lvalue expressions, including list indexing and variables.

Then semant takes the Abstract Syntax Tree and returns a syntactically checked AST with partial type inference and syntax checking. This semantic checking includes a simple type inference sys-

tem which assigns types to primitives and then propagates these types up the ast to higher level expressions. The type inference mechanism also uses explicitly declared types to help it infer types in conditional branches. When a conditional branch is encountered, if types differ between the two branches, semant reverts to a "Dynamic type" or Dyn, which can be any type and does not raise errors when passed to functions with explicit types. This type inference information is stored in a StringMap which is passed around the expression and statement evaluation functions. Separate maps are used for locals and globals, since Coral, like Python, only allows the global scope and the most immediate local scope in variable evaluation. We decided against implementing closures because we did not have enough time, but we might extend our language with closures in the future.

Inferred types are compared against explicitly declared types at this semant stage, and many errors are caught due to invalid usage. These checks include invalid types in assignments, function calls, function return types, list access, function arguments. We also check if functions are called with the correct number of arguments where possible. Sometimes this information is impossible to determine due to the dynamic nature of Python's function definitions, but we do our best to make this inference where possible. We also check the arguments to binary operations to make sure they are valid. We do no sideways

Jacob implemented the Parser and Semantic checker.

5.4 Code Generation

File: codegen.ml

The code generator takes in the semantically checked SAST and builds the LLVM. For dynmaic objects (where the type is unknown at compile time), coral object pointers are placed on the stack. For immutable objects with known types, the data itself is simply placed on the stack.

The record "state" propagates through the program (returned by both expr() and stmt()) and keeps track of the namespace, current function, current builder, a record of all the functions that have already been optimized, and several useful state flags. The function expr() returns a Box(v) or Raw(v), where "v" is an llvalue, for example "v" would be an i32 in the case of expr returning a type-known integer. Raw(v) always corresponds to typed known values, while Box(v) refers to any sort of value that's stored in a coral cobject that is any value of unknown type, function object, etc. The namespace holds a map from Bind(string,typ) to the addresses of locals and globals. These addresses are wrapped in an algebreic datatype: BoxAddr(a) where "a" is a coral object pointer llvalue, and RawAddr(a) where "a" is an int/bool/float pointer.

Coral objects are implemented as a type pointer and a data pointer, in a data model very similar to that of Python.

Matt and Sanford implemented Code Generation.

6 Testing

Individual features and aspects of Coral were developed in individual branches. Before being merged into other branches these features were tested using test programs written by the team member who was working on the feature. There is at least one test case for any new feature/commit. All team members were responsible for contributing to testing development.

There are both success and fail test cases. There are also tests dedicated to testing specifically semant (and typing). In total there are 124 test cases.

6.1 Testing Suite and Automation

All tests are stored in the /test/ folder. Success test cases have test-*.cl or stest-*cl (for semant checks). Fail cases follow a similar format of fail-*.cl or sfail-*.cl (for semant checks). All four of these different cases have a file with the expected output, *.out.

Testing automation is based off the Micro-C testing suite provided. The testall.sh script compiles and runs all of the *.cl files in the /test folder and compares the output to the corresponding *.out file. This script is run when Coral is compiled using make.

6.2 Testing Optimization

In order to test our optimization, we compared our optimized programs to our unoptimized versions. We compared the lines of code that we were able to reduce by. In fully optimized code, LLVM loc count drops by at least 1000 lines. This reduces the binary sizes by tens of kilobytes.

We also tested optimization by comparing the speed of running a fully optimized Coral program in both Coral and Python. Since Coral is cross compatible we are able to run the same program in both languages. The runtime performance increases by as much as 100x for programs involving frequent heap allocations in Python. For example, the following lines of code can be run in both Coral and Python:

```
x = 1000000000
count = 0
while x > 0:
    count += 1
        x -= 1
print(count)
```

In Coral this program is about 40 times faster. Coral runs this in .4 seconds while in Python it takes 23.4 seconds.

6.3 Example Input-Output

Since our LLVM output files are large, we have moved the example code to the end of the file, directly before the appendix.

7 Lessons Learned

7.1 Jacob Austin

Developing a programming language is hard, and sometimes you just need to stay up all night developing the parser or fixing bugs in order to meet the project deadline. I would definitely recommend developing a nice environment for testing and debugging, since that saves more time than it takes to build. Eventually we built an interpreter that displayed the lexer, parser, and semant outputs for easy debugging. We also had several nice test scripts and a nice coral.ml compiler setup which made testing easier, and made it easy to catch when changes broke the library.

I'd also strongly recommend planning changes out before trying to implement them. There's a sort of OCaml syndrome where you have an idea, spend 30 minutes refactoring match statements to add it, and then realize it's a bad idea. Try drawing new architectures out on paper and then

implementing them. Having a strong semant.ml foundation is also a good idea. If you can catch a lot of errors and have a strong idea of how the language will be structures in semant, you can avoid a lot of pain later when you run into issues in codegen.

7.2 Matthew Bowers

Throughout this project I learned a lot about the value of planning out features in rigorous detail before even beginning to implement them. The first few weeks of working on Coral I would try to dive directly into whatever idea I had that I thought would work, and found out pretty quickly that this would lead to way more work in the long run as I'd get several hours into implementing a certain feature or datamodel only to realize that there is a far more efficient or convenient way to do it. That either resulted in a lot of time spent on massive refactoring, or just ugly code.

In particular when working on the data model I found sketching diagrams to be incredibly useful. I spent about 16 hours planning out the precise theory of how our object-based model would work before writing any code, and the same went for optimization. I'd plan for hours before realizing a problem with the system, but it was much easier to rectify and improve on the model without having written any code for it.

I also realized something I really like about functional programming is the ability to use algebreic datatypes and natural automatic documentation. For example the lookup() function takes a Bind(name,typ) for us, but often in code that bind had to be constructed on the fly so it would look something like "lookup namespace (Bind(name,ty))" which made it extremely clear exactly the type that lookup was taking. Overall I've really loved my work in function programming and I'd like to keep working with them in the future in some capacity.

7.3 Rebecca Cawkwell

Try not to get concussions - they really impede you from being a productive and functioning human being.

But more on topic, the planning stages of development, take a lot of time, but are worth it. It is more productive to spend time upfront discussing exactly what needs to be accomplished and considering all of the possible edge cases. One of the reasons I think our group was able to accomplish what we wanted to is that we spent a lot of time talking/writing/drawing through the plan.

7.4 Sanford Miller

Over the course of this project, I came to a greater appreciation of the importance of planning code in advance. Almost everyone who programs on projects of meaningful scale eventually learns that lesson (usually the hard way); fortunately, school is a relatively forgiving environment in which to discover that if don't plan them out, 200 lines of code can take not 10 but 100 times the time of 20.

A (somewhat) more unique insight from my experience with Coral is the value of planning iteratively. Our team meet consistently at least once a week throughout the semester. Earlier on, we tended to talk for extended periods of time (2-3 hours), but usually the important decisions and insights happened in the first half and hour. We got tired and sometimes talked in circles – not because weren't smart enough to come up solutions to our problems, but because the conversation had moved into purely theoretical future aspects of Coral about which none of us had real knowledge or thought-through ideas. Often those same issues would be resolved in minutes at the next meeting. So, as the semester went on, we started to cut our "main" meeting shorter and to instead check in with each other frequently during the week either in person or through Facebook messenger, which lead to much more productive and focused discussions.

Unrelatedly, it's been a blast to start the process of learning how to think in a functional way; Ocaml represents, unlike the vast majority of things proclaimed to be paradigm-shifting, a very different way of thinking about a domain – I found it extremely frustrating at first and often struggled to follow what was going on in the slides, but I'm glad I pushed through.

7.4.1 GCD program

The following is a GCD program written in Coral with the corresponding LLVM output.

```
1  def gcd(a,b):
2          while a != b:
3                  if a>b:
4                          a = a-b
5                  else:
6                          b = b-a
7          return a
8
9  print(gcd(1234342213,334232))
```

The following is the LLVM output simplified by the LLVM optimization (opt) pre-processor. The original binary was over 1600 lines, but when optiized only about 100 were relevant.

```
    ; ModuleID = 'Coral'
source_filename = "Coral"

%CType.0 = type { %CObj* (%CObj*, %CObj*)*, %CObj* (%CObj*, %CObj*)*, %CObj* (%CObj*, %CObj*)*, %CObj
%CObj = type { i8*, %CType.0* }
%CList = type { i8*, i32, i32 }

@ctype_func = global %CType.0 { %CObj* (%CObj*, %CObj*)* null, %CObj* (%CObj*, %CObj*)* null, %CObj*
@ctype_int = global %CType.0 { %CObj* (%CObj*, %CObj*)* @int_add, %CObj* (%CObj*, %CObj*)* @int_sub,
@ctype_float = global %CType.0 { %CObj* (%CObj*, %CObj*)* @float_add, %CObj* (%CObj*, %CObj*)* @floa
@ctype_bool = global %CType.0 { %CObj* (%CObj*, %CObj*)* null, %CObj* (%CObj*, %CObj*)* null, %CObj*
@ctype_char = global %CType.0 { %CObj* (%CObj*, %CObj*)* null, %CObj* (%CObj*, %CObj*)* null, %CObj*
@ctype_list = global %CType.0 { %CObj* (%CObj*, %CObj*)* null, %CObj* (%CObj*, %CObj*)* null, %CObj*
@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.3 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@gcd_func_obj = global %CObj zeroinitializer
@gcd_func = global %CObj* @gcd_func_obj
@fmt.4 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.5 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.6 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
@fmt.7 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.8 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.9 = private unnamed_addr constant [4 x i8] c"%g\0A\00"

define i32 @func_heapify(%CObj*) {
entry:
  ret i32 0
}
```

```
declare noalias i8* @malloc(i32)

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

declare i32 @exit(i32)

define i32 @main() {
entry:
  %result = call i32 @gcd(i32 1234342213, i32 334232)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.4, i32
  ret i32 0
}

define i32 @gcd(i32, i32) {
entry:
  %a_int = alloca i32
  %b_int = alloca i32
  %a_int1 = alloca i32
  store i32 %0, i32* %a_int1
  store i32 %1, i32* %b_int
  br label %while

while:                                            ; preds = %merge, %entry
  %a8 = load i32, i32* %a_int1
  %b9 = load i32, i32* %b_int
  %binop_result10 = icmp ne i32 %a8, %b9
  br i1 %binop_result10, label %while_body, label %merge11

while_body:                                       ; preds = %while
  %a = load i32, i32* %a_int1
  %b = load i32, i32* %b_int
  %binop_result = icmp sgt i32 %a, %b
  br i1 %binop_result, label %then, label %else

merge:                                            ; preds = %else, %then
  br label %while

then:                                             ; preds = %while_body
  %a2 = load i32, i32* %a_int1
  %b3 = load i32, i32* %b_int
  %binop_result4 = sub i32 %a2, %b3
  store i32 %binop_result4, i32* %a_int1
  br label %merge

else:                                             ; preds = %while_body
  %b5 = load i32, i32* %b_int
  %a6 = load i32, i32* %a_int1
  %binop_result7 = sub i32 %b5, %a6
  store i32 %binop_result7, i32* %b_int
  br label %merge

merge11:                                          ; preds = %while
  %a12 = load i32, i32* %a_int1
  ret i32 %a12
```

```
}
```

7.5 Dynamic and Not-Dynamic

```
l = [1, 0, 5, 12]
a = l[2]

for i in l:
    print(i)

print(a)

def foo():
    if x == 3:
        return x
    else:
        return 4.0

x = 5

print(foo())
```

The LLVM output is below. This uses a combination of optimized and un-optimized code, so the full output is needed. It seems the in general LLVM is capable of removing what we do not use.

```
; ModuleID = 'Coral'
source_filename = "Coral"

%CType.0 = type { %CObj* (%CObj*, %CObj*)*, %CObj* (%CObj*, %CObj*)*, %CObj* (%CObj*, %CObj*)*, %COb
%CObj = type { i8*, %CType.0* }
%CList = type { i8*, i32, i32 }

@ctype_func = global %CType.0 { %CObj* (%CObj*, %CObj*)* null, %CObj* (%CObj*, %CObj*)* null, %CObj*
@ctype_int = global %CType.0 { %CObj* (%CObj*, %CObj*)* @int_add, %CObj* (%CObj*, %CObj*)* @int_sub,
@ctype_float = global %CType.0 { %CObj* (%CObj*, %CObj*)* @float_add, %CObj* (%CObj*, %CObj*)* @floa
@ctype_bool = global %CType.0 { %CObj* (%CObj*, %CObj*)* null, %CObj* (%CObj*, %CObj*)* null, %CObj*
@ctype_char = global %CType.0 { %CObj* (%CObj*, %CObj*)* null, %CObj* (%CObj*, %CObj*)* null, %CObj*
@ctype_list = global %CType.0 { %CObj* (%CObj*, %CObj*)* null, %CObj* (%CObj*, %CObj*)* null, %CObj*
@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.3 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@x_int = global i32 0
@fmt.4 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.5 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.6 = private unnamed_addr constant [4 x i8] c"%g\0A\00"

define %CObj* @int_add(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
```

```
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i32*
  %data = load i32, i32* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i32*
  %data6 = load i32, i32* %x45
  %result_data = add i32 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i
  %__new_dataptr = bitcast i8* %malloccall7 to i32*
  %dataptr_as_i8 = bitcast i32* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_int, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i32 %result_data, i32* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @int_sub(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i32*
  %data = load i32, i32* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i32*
  %data6 = load i32, i32* %x45
  %result_data = sub i32 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i
  %__new_dataptr = bitcast i8* %malloccall7 to i32*
  %dataptr_as_i8 = bitcast i32* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_int, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i32 %result_data, i32* %__new_dataptr
```

```
  ret %CObj* %__new_objptr
}

define %CObj* @int_mul(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i32*
  %data = load i32, i32* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i32*
  %data6 = load i32, i32* %x45
  %result_data = mul i32 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i
  %__new_dataptr = bitcast i8* %malloccall7 to i32*
  %dataptr_as_i8 = bitcast i32* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_int, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i32 %result_data, i32* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @int_div(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i32*
  %data = load i32, i32* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i32*
  %data6 = load i32, i32* %x45
  %result_data = sdiv i32 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i
  %__new_dataptr = bitcast i8* %malloccall7 to i32*
```

```
  %dataptr_as_i8 = bitcast i32* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_int, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i32 %result_data, i32* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @int_eq(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i32*
  %data = load i32, i32* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i32*
  %data6 = load i32, i32* %x45
  %result_data = icmp eq i32 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @int_neq(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i32*
  %data = load i32, i32* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i32*
```

```
  %data6 = load i32, i32* %x45
  %result_data = icmp ne i32 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @int_lesser(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i32*
  %data = load i32, i32* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i32*
  %data6 = load i32, i32* %x45
  %result_data = icmp slt i32 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @int_leq(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
```

```
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i32*
  %data = load i32, i32* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i32*
  %data6 = load i32, i32* %x45
  %result_data = icmp sle i32 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @int_greater(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i32*
  %data = load i32, i32* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i32*
  %data6 = load i32, i32* %x45
  %result_data = icmp sgt i32 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @int_geq(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
```

```
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i32*
  %data = load i32, i32* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i32*
  %data6 = load i32, i32* %x45
  %result_data = icmp sge i32 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @int_and(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i32*
  %data = load i32, i32* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i32*
  %data6 = load i32, i32* %x45
  %result_data = and i32 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i
  %__new_dataptr = bitcast i8* %malloccall7 to i32*
  %dataptr_as_i8 = bitcast i32* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_int, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i32 %result_data, i32* %__new_dataptr
```

```
  ret %CObj* %__new_objptr
}

define %CObj* @int_or(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i32*
  %data = load i32, i32* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i32*
  %data6 = load i32, i32* %x45
  %result_data = or i32 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i
  %__new_dataptr = bitcast i8* %malloccall7 to i32*
  %dataptr_as_i8 = bitcast i32* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_int, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i32 %result_data, i32* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @int_neg(%CObj* %remote_self_p) {
entry:
  %x2 = getelementptr inbounds %CObj, %CObj* %remote_self_p, i32 0, i32 1
  %x3 = load %CType.0*, %CType.0** %x2
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %x22 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x33 = load i8*, i8** %x22
  %x4 = bitcast i8* %x33 to i32*
  %data = load i32, i32* %x4
  %result_data = sub i32 0, %data
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall4 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i
  %__new_dataptr = bitcast i8* %malloccall4 to i32*
  %dataptr_as_i8 = bitcast i32* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_int, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
```

```
  store i32 %result_data, i32* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @int_not(%CObj* %remote_self_p) {
entry:
  %x2 = getelementptr inbounds %CObj, %CObj* %remote_self_p, i32 0, i32 1
  %x3 = load %CType.0*, %CType.0** %x2
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %x22 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x33 = load i8*, i8** %x22
  %x4 = bitcast i8* %x33 to i32*
  %data = load i32, i32* %x4
  %result_data = xor i32 %data, -1
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall4 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i
  %__new_dataptr = bitcast i8* %malloccall4 to i32*
  %dataptr_as_i8 = bitcast i32* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_int, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i32 %result_data, i32* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @float_add(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to double*
  %data = load double, double* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to double*
  %data6 = load double, double* %x45
  %result_data = fadd double %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr (double, double* null, i3
  %__new_dataptr = bitcast i8* %malloccall7 to double*
  %dataptr_as_i8 = bitcast double* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_float, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
```

```
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store double %result_data, double* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @float_sub(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to double*
  %data = load double, double* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to double*
  %data6 = load double, double* %x45
  %result_data = fsub double %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr (double, double* null, i3
  %__new_dataptr = bitcast i8* %malloccall7 to double*
  %dataptr_as_i8 = bitcast double* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_float, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store double %result_data, double* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @float_mul(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to double*
  %data = load double, double* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to double*
  %data6 = load double, double* %x45
  %result_data = fmul double %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
```

```
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr (double, double* null, i3
  %__new_dataptr = bitcast i8* %malloccall7 to double*
  %dataptr_as_i8 = bitcast double* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_float, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store double %result_data, double* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @float_div(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to double*
  %data = load double, double* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to double*
  %data6 = load double, double* %x45
  %result_data = fdiv double %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr (double, double* null, i3
  %__new_dataptr = bitcast i8* %malloccall7 to double*
  %dataptr_as_i8 = bitcast double* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_float, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store double %result_data, double* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @float_eq(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to double*
  %data = load double, double* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
```

```
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to double*
  %data6 = load double, double* %x45
  %result_data = fcmp ueq double %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @float_neq(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to double*
  %data = load double, double* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to double*
  %data6 = load double, double* %x45
  %result_data = fcmp une double %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @float_lesser(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
```

```
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to double*
  %data = load double, double* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to double*
  %data6 = load double, double* %x45
  %result_data = fcmp ult double %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @float_leq(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to double*
  %data = load double, double* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to double*
  %data6 = load double, double* %x45
  %result_data = fcmp ule double %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @float_greater(%CObj* %remote_self_p, %CObj* %remote_other_p) {
```

```
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to double*
  %data = load double, double* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to double*
  %data6 = load double, double* %x45
  %result_data = fcmp ugt double %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @float_geq(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to double*
  %data = load double, double* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to double*
  %data6 = load double, double* %x45
  %result_data = fcmp uge double %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
```

```
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @float_neg(%CObj* %remote_self_p) {
entry:
  %x2 = getelementptr inbounds %CObj, %CObj* %remote_self_p, i32 0, i32 1
  %x3 = load %CType.0*, %CType.0** %x2
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %x22 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x33 = load i8*, i8** %x22
  %x4 = bitcast i8* %x33 to double*
  %data = load double, double* %x4
  %result_data = fsub double -0.000000e+00, %data
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall4 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr (double, double* null, i3
  %__new_dataptr = bitcast i8* %malloccall4 to double*
  %dataptr_as_i8 = bitcast double* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_float, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store double %result_data, double* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @bool_eq(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i1*
  %data = load i1, i1* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i1*
  %data6 = load i1, i1* %x45
  %result_data = icmp eq i1 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
```

```
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @bool_neq(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i1*
  %data = load i1, i1* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i1*
  %data6 = load i1, i1* %x45
  %result_data = icmp eq i1 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @bool_lesser(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i1*
  %data = load i1, i1* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i1*
  %data6 = load i1, i1* %x45
  %result_data = icmp slt i1 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
```

```
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @bool_leq(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i1*
  %data = load i1, i1* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i1*
  %data6 = load i1, i1* %x45
  %result_data = icmp sle i1 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @bool_greater(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i1*
  %data = load i1, i1* %x4
```

```
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i1*
  %data6 = load i1, i1* %x45
  %result_data = icmp sgt i1 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @bool_geq(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i1*
  %data = load i1, i1* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i1*
  %data6 = load i1, i1* %x45
  %result_data = icmp sge i1 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @bool_and(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
```

```
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i1*
  %data = load i1, i1* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i1*
  %data6 = load i1, i1* %x45
  %result_data = and i1 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @bool_or(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i1*
  %data = load i1, i1* %x4
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %x45 = bitcast i8* %x34 to i1*
  %data6 = load i1, i1* %x45
  %result_data = or i1 %data, %data6
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall7 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}
```

```
define %CObj* @bool_neg(%CObj* %remote_self_p) {
entry:
  %x2 = getelementptr inbounds %CObj, %CObj* %remote_self_p, i32 0, i32 1
  %x3 = load %CType.0*, %CType.0** %x2
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %x22 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x33 = load i8*, i8** %x22
  %x4 = bitcast i8* %x33 to i1*
  %data = load i1, i1* %x4
  %result_data = sub i1 false, %data
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall4 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall4 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @bool_not(%CObj* %remote_self_p) {
entry:
  %x2 = getelementptr inbounds %CObj, %CObj* %remote_self_p, i32 0, i32 1
  %x3 = load %CType.0*, %CType.0** %x2
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %x22 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x33 = load i8*, i8** %x22
  %x4 = bitcast i8* %x33 to i1*
  %data = load i1, i1* %x4
  %result_data = xor i1 %data, true
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall4 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall4 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @char_eq(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
```

```
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %data = load i8, i8* %x3
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %data5 = load i8, i8* %x34
  %result_data = icmp eq i8 %data, %data5
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall6 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall6 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @char_neq(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %data = load i8, i8* %x3
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %data5 = load i8, i8* %x34
  %result_data = icmp eq i8 %data, %data5
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall6 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall6 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @char_lesser(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
```

```
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %data = load i8, i8* %x3
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %data5 = load i8, i8* %x34
  %result_data = icmp slt i8 %data, %data5
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall6 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall6 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @char_leq(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %data = load i8, i8* %x3
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %data5 = load i8, i8* %x34
  %result_data = icmp sle i8 %data, %data5
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall6 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall6 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}
```

```
define %CObj* @char_greater(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %data = load i8, i8* %x3
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %data5 = load i8, i8* %x34
  %result_data = icmp sgt i8 %data, %data5
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall6 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall6 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @char_geq(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %data = load i8, i8* %x3
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %data5 = load i8, i8* %x34
  %result_data = icmp sge i8 %data, %data5
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %malloccall6 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32)
  %__new_dataptr = bitcast i8* %malloccall6 to i1*
  %dataptr_as_i8 = bitcast i1* %__new_dataptr to i8*
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_bool, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %dataptr_as_i8, i8** %datafieldptr
  store i1 %result_data, i1* %__new_dataptr
  ret %CObj* %__new_objptr
```

```
}

define %CObj* @char_and(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %data = load i8, i8* %x3
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %data5 = load i8, i8* %x34
  %result_data = add i8 %data, %data5
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %__new_dataptr = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i3
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_char, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %__new_dataptr, i8** %datafieldptr
  store i8 %result_data, i8* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @char_or(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %data = load i8, i8* %x3
  %x23 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x34 = load i8*, i8** %x23
  %data5 = load i8, i8* %x34
  %result_data = or i8 %data, %data5
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %__new_dataptr = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i3
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_char, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %__new_dataptr, i8** %datafieldptr
  store i8 %result_data, i8* %__new_dataptr
  ret %CObj* %__new_objptr
}
```

```
define %CObj* @char_not(%CObj* %remote_self_p) {
entry:
  %x2 = getelementptr inbounds %CObj, %CObj* %remote_self_p, i32 0, i32 1
  %x3 = load %CType.0*, %CType.0** %x2
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %x22 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x33 = load i8*, i8** %x22
  %data = load i8, i8* %x33
  %result_data = xor i8 %data, -1
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr (i1*,
  %__new_objptr = bitcast i8* %malloccall to %CObj*
  %__new_dataptr = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* null, i32 1) to i3
  %ctypefieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 1
  store %CType.0* @ctype_char, %CType.0** %ctypefieldptr
  %datafieldptr = getelementptr inbounds %CObj, %CObj* %__new_objptr, i32 0, i32 0
  store i8* %__new_dataptr, i8** %datafieldptr
  store i8 %result_data, i8* %__new_dataptr
  ret %CObj* %__new_objptr
}

define %CObj* @list_idx(%CObj* %remote_self_p, %CObj* %remote_other_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %other_p_p = alloca %CObj*
  store %CObj* %remote_other_p, %CObj** %other_p_p
  %other_p_p2 = load %CObj*, %CObj** %other_p_p
  %__gep_addr = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %__objptr = load i8*, i8** %__gep_addr
  %__clistptr = bitcast i8* %__objptr to %CList*
  %x2 = getelementptr inbounds %CObj, %CObj* %other_p_p2, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %x4 = bitcast i8* %x3 to i32*
  %data = load i32, i32* %x4
  %__gep_addr3 = getelementptr inbounds %CList, %CList* %__clistptr, i32 0, i32 2
  %__capacity = load i32, i32* %__gep_addr3
  %__inbounds = icmp slt i32 %data, %__capacity
  %__gep_addr4 = getelementptr inbounds %CList, %CList* %__clistptr, i32 0, i32 0
  %__gep_addr_as_cobjptrptrptr = bitcast i8** %__gep_addr4 to %CObj***
  %__gep_addr_as_cobjptrptr = load %CObj**, %CObj*** %__gep_addr_as_cobjptrptrptr
  %__gep_addr_as_cobjptrptr5 = getelementptr %CObj*, %CObj** %__gep_addr_as_cobjptrptr, i32 %data
  %__cobjptr = load %CObj*, %CObj** %__gep_addr_as_cobjptrptr5
  ret %CObj* %__cobjptr
}

define %CObj* @func_call(%CObj* %remote_self_p, %CObj** %remote_argv) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %argv_p = alloca %CObj**
```

```
  store %CObj** %remote_argv, %CObj*** %argv_p
  %argv = load %CObj**, %CObj*** %argv_p
  %x2 = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %x3 = load i8*, i8** %x2
  %fnptr = bitcast i8* %x3 to %CObj* (%CObj**)*
  %result = call %CObj* %fnptr(%CObj** %argv)
  ret %CObj* %result
}

define i32 @int_print(%CObj* %remote_self_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %dat = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %dat2 = bitcast i8** %dat to i32**
  %raw_data_addr = load i32*, i32** %dat2
  %raw_data = load i32, i32* %raw_data_addr
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0,
  ret i32 0
}

define i32 @char_print(%CObj* %remote_self_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %dat = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %raw_data_addr = load i8*, i8** %dat
  %raw_data = load i8, i8* %raw_data_addr
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.3, i32
  ret i32 0
}

define i32 @float_print(%CObj* %remote_self_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %dat = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %dat2 = bitcast i8** %dat to double**
  %raw_data_addr = load double*, double** %dat2
  %raw_data = load double, double* %raw_data_addr
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1, i32
  ret i32 0
}

define i32 @bool_print(%CObj* %remote_self_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %dat = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %dat2 = bitcast i8** %dat to i1**
```

```
  %raw_data_addr = load i1*, i1** %dat2
  %raw_data = load i1, i1* %raw_data_addr
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2, i32
  ret i32 0
}

define i32 @int_heapify(%CObj* %remote_self_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %dat = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %dat2 = bitcast i8** %dat to i32**
  %raw_data_addr = load i32*, i32** %dat2
  %raw_data = load i32, i32* %raw_data_addr
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i3
  %heap_data_p = bitcast i8* %malloccall to i32*
  store i32 %raw_data, i32* %heap_data_p
  %heap_data_p3 = bitcast i32* %heap_data_p to i8*
  store i8* %heap_data_p3, i8** %dat
  ret i32 0
}

define i32 @float_heapify(%CObj* %remote_self_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %dat = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %dat2 = bitcast i8** %dat to double**
  %raw_data_addr = load double*, double** %dat2
  %raw_data = load double, double* %raw_data_addr
  %malloccall = tail call i8* @malloc(i32 ptrtoint (double* getelementptr (double, double* null, i32
  %heap_data_p = bitcast i8* %malloccall to double*
  store double %raw_data, double* %heap_data_p
  %heap_data_p3 = bitcast double* %heap_data_p to i8*
  store i8* %heap_data_p3, i8** %dat
  ret i32 0
}

define i32 @bool_heapify(%CObj* %remote_self_p) {
entry:
  %self_p_p = alloca %CObj*
  store %CObj* %remote_self_p, %CObj** %self_p_p
  %self_p_p1 = load %CObj*, %CObj** %self_p_p
  %dat = getelementptr inbounds %CObj, %CObj* %self_p_p1, i32 0, i32 0
  %dat2 = bitcast i8** %dat to i1**
  %raw_data_addr = load i1*, i1** %dat2
  %raw_data = load i1, i1* %raw_data_addr
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32))
  %heap_data_p = bitcast i8* %malloccall to i1*
  store i1 %raw_data, i1* %heap_data_p
  %heap_data_p3 = bitcast i1* %heap_data_p to i8*
  store i8* %heap_data_p3, i8** %dat
```

```
  ret i32 0
}

define i32 @func_heapify(%CObj*) {
entry:
  ret i32 0
}

declare noalias i8* @malloc(i32)

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

declare i32 @exit(i32)

define i32 @main() {
entry:
  store i32 3, i32* @x_int
  ret i32 0
}
```

8 Appendix

Makefile

```
.PHONY: all clean byte native fn


OCB_FLAGS = -tag bin_annot -I src  -use-ocamlfind -lib unix
OCB = ocamlbuild $(OCB_FLAGS)

all: clean native
  ./coral.native -r llvm-test.cl

fn: clean native
  ./coral.native -r fn-test.cl

clean:
  $(OCB) -clean
  rm -rf testall.log *.diff *.out main source.ll source.o source.s .ll

native:
  $(OCB) coral.native

byte:
  $(OCB) coral.byte

test: native
  ./testall.sh

# Building the tarball

```

```
#TESTS = add1

#TESTFILES = $(TESTS:%=test-%.out)

#TARFILES = ast.ml codegen.ml Makefile _tags coral.ml parsesr.mly README \
    scanner.mll semant.ml testall.sh \
```

testall.sh

```
/bin/sh

Regression testing script for Coral
Taken largely from provide MicroC script
 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
I="lli"
LI="/usr/local/opt/llvm/bin/lli"

Path to the LLVM compiler
C="llc"

Path to the C compiler
="cc"

Path to the coralc compiler.
RAL="./coral.native"

Set time limit for all operations
imit -t 30

oballog=testall.log
 -f $globallog
ror=0
obalerror=0

ep=0

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


gnalError() {
  if [ $error -eq 0 ] ; then
cho "FAILED"
rror=1
  fi
  echo "  $1"

```

```
46
47 Compare <outfile> <reffile> <difffile>
48 Compares the outfile with reffile.  Differences, if any, written to difffile
49 mpare() {
50   generatedfiles="$generatedfiles $3"
51   echo diff -b $1 $2 ">" $3 1>&2
52   diff -b "$1" "$2" > "$3" 2>&1 || {
53 ignalError "$1 differs"
54 cho "FAILED $1 differs from $2" 1>&2
55   }
56
57
58 Run <args>
59 Report the command, run it, and report any errors
60 n() {
61   echo $* 1>&2
62   eval $* || {
63 ignalError "$1 failed on $*"
64 eturn 1
65   }
66
67
68 RunFail <args>
69 Report the command, run it, and expect an error
70 nFail() {
71   echo $* 1>&2
72   eval $* && {
73 ignalError "failed: $* did not report an error"
74 eturn 1
75   }
76   return 0
77
78
79
80 eckLLVM() {
81   error=0
82   basename=`echo $1 | sed 's/.*\\///
83                             s/.cl//'`
84   reffile=`echo $1 | sed 's/.cl$//'`
85   basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."
86
87   echo -n "$basename..."
88
89   echo 1>&2
90   echo "###### Testing $basename" 1>&2
91
92   generatedfiles=""
93
94   generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe ${basename}.out" &&
95
96   Run "$CORAL" "$1" ">" "${basename}.out" &&
97   Compare ${basename}.out ${reffile}.out ${basename}.diff
98
99   # Report the status and clean up the generated files
```

```
100
101   if [ $error -eq 0 ] ; then
102   if [ $keep -eq 0 ] ; then
103       rm -f $generatedfiles
104   fi
105   echo "OK"
106   echo "###### SUCCESS" 1>&2
107   else
108   echo "###### FAILED" 1>&2
109   globalerror=$error
110   fi
111
112
113 eckSemant() {
114   error=0
115   basename=`echo $1 | sed 's/.*\\///
116                             s/.cl//'`
117   reffile=`echo $1 | sed 's/.cl$//'`
118   basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."
119
120   echo -n "$basename..."
121
122   echo 1>&2
123   echo "###### Testing $basename" 1>&2
124
125   generatedfiles=""
126
127   generatedfiles="$generatedfiles ${basename}.out" &&
128
129   Run "$CORAL" "-no-compile" "$1" ">" "${basename}.out" &&
130   Compare ${basename}.out ${reffile}.out ${basename}.diff
131
132   # Report the status and clean up the generated files
133
134   if [ $error -eq 0 ] ; then
135 f [ $keep -eq 0 ] ; then
136    rm -f $generatedfiles
137 i
138 cho "OK"
139 cho "###### SUCCESS" 1>&2
140   else
141 cho "###### FAILED" 1>&2
142 lobalerror=$error
143   fi
144
145
146  [ $# -ge 1 ]
147 en
148   files=$@
149 se
150   files="tests/test-*.cl tests/sfail-*.cl tests/stest-*.cl" # tests/fail-*.cl
151
152
```

```
r file in $files

  case $file in

stest-*)
   CheckSemant $file 2>> $globallog
   ;;
sfail-*)
     CheckSemant $file 2>> $globallog
      ;;
  *test-*)
      CheckLLVM $file 2>> $globallog
      ;;
  *fail-*)
      CheckLLVM $file 2>> $globallog
      ;;
)
   echo "unknown file type $file"
   globalerror=1
   ;;
  esac
ne

it $globalerror
```

8.1 Optimization Files

codegen.ml

```
(* Code generation: translate takes a semantically checked SAST and generates LLVM for it
See codegenutils.ml for some type definitions used heavily throughout this file.
*)

module L = Llvm
open Ast
open Sast
open Codegenutils


let initial_list_size = 10
let list_growth_factor = 2


(* translate : Sast.program -> Llvm.module *)
let translate prgm =   (* note this whole thing only takes two things: globals= list of (typ,name) (
  let context = L.global_context () in  (* context keeps track of global vars and stuff i think *)

  (* Create the LLVM compilation module boolo which
     we will generate code *)
  let the_module = L.create_module context "Coral" in  (* the_module will hold all functs + global v
  let pm() = L.dump_module the_module in

  (* Get types from the context *)
```

```
let int_t      = L.i32_type    context
and float_t    = L.double_type context
and bool_t     = L.i1_type     context
and char_t     = L.i8_type     context in

(* ptr types *)
let int_pt = L.pointer_type int_t
and float_pt = L.pointer_type float_t
and bool_pt = L.pointer_type bool_t
and char_pt = L.pointer_type char_t in
let char_ppt = L.pointer_type char_pt in

(* define cobj and ctype structs *)
let cobj_t = L.named_struct_type context "CObj" in (*define a named struct*)
let cobj_pt = L.pointer_type cobj_t in
let cobj_ppt = L.pointer_type cobj_pt in

(* all generic userdef functions follow this type *)
let userdef_fn_t = L.function_type cobj_pt [| cobj_ppt |] in   (* takes an argv *)
let userdef_fn_pt = L.pointer_type userdef_fn_t in

(* define cobj_list and ctype_list structs *)
let clist_t = L.named_struct_type context "CList" in (*define a named struct*)
let clist_pt = L.pointer_type clist_t in

(* define ctype and ctype structs *)
let ctype_t = L.named_struct_type context "CType" in (*define a named struct*)
let ctype_pt = L.pointer_type ctype_t in
let ctype_ppt = L.pointer_type ctype_pt in

(* cobj idxs *)
let cobj_data_idx = 0
and cobj_type_idx = 1 in

(* clist idxs *)
let clist_data_idx = 0
and clist_len_idx = 1
and clist_cap_idx = 2
in

(* ctype idx *)
let ctype_add_idx = 0
and ctype_sub_idx = 1
and ctype_mul_idx = 2
and ctype_div_idx = 3
and ctype_exp_idx = 4
and ctype_eq_idx = 5
and ctype_neq_idx = 6
and ctype_lesser_idx = 7
and ctype_leq_idx = 8
and ctype_greater_idx = 9
and ctype_geq_idx = 10
and ctype_and_idx = 11
and ctype_or_idx = 12
```

```
and ctype_neg_idx = 13
and ctype_not_idx = 14
and ctype_idx_idx = 15
and ctype_call_idx = 16
and ctype_heapify_idx = 17
and ctype_print_idx = 18
and num_ctype_idxs = 19 in (**must update when adding idxs! (tho not used anywhere yet)**)

(* type sigs for fns in ctype *)
let ctype_add_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
and ctype_sub_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
and ctype_mul_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
and ctype_div_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
and ctype_exp_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
and ctype_eq_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
and ctype_neq_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
and ctype_lesser_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
and ctype_leq_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
and ctype_greater_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
and ctype_geq_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
and ctype_and_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
and ctype_or_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
and ctype_neg_t = L.function_type cobj_pt [| cobj_pt |]
and ctype_not_t = L.function_type cobj_pt [| cobj_pt |]
and ctype_idx_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
and ctype_call_t = L.function_type cobj_pt [| cobj_pt ; cobj_ppt |]
and ctype_heapify_t = L.function_type int_t [| cobj_pt |]
and ctype_print_t = L.function_type int_t [| cobj_pt |] in

(* type sigs for ptrs to fns in ctype *)
let ctype_add_pt = L.pointer_type ctype_add_t
and ctype_sub_pt = L.pointer_type ctype_sub_t
and ctype_mul_pt = L.pointer_type ctype_mul_t
and ctype_div_pt = L.pointer_type ctype_div_t
and ctype_exp_pt = L.pointer_type ctype_exp_t
and ctype_eq_pt = L.pointer_type ctype_eq_t
and ctype_neq_pt = L.pointer_type ctype_neq_t
and ctype_lesser_pt = L.pointer_type ctype_lesser_t
and ctype_leq_pt = L.pointer_type ctype_leq_t
and ctype_greater_pt = L.pointer_type ctype_greater_t
and ctype_geq_pt = L.pointer_type ctype_geq_t
and ctype_and_pt = L.pointer_type ctype_and_t
and ctype_or_pt = L.pointer_type ctype_or_t
and ctype_neg_pt = L.pointer_type ctype_neg_t
and ctype_not_pt = L.pointer_type ctype_not_t
and ctype_idx_pt = L.pointer_type ctype_idx_t
and ctype_call_pt = L.pointer_type ctype_call_t
and ctype_heapify_pt = L.pointer_type ctype_heapify_t
and ctype_print_pt = L.pointer_type ctype_print_t in

let ctype_t = L.named_struct_type context "CType" in (*define a named struct*)
let ctype_pt = L.pointer_type ctype_t in

(* set ctype and cobj struct bodies *)
```

```
ignore(L.struct_set_body cobj_t [| char_pt; ctype_pt |] false);
ignore(L.struct_set_body clist_t [| char_pt; int_t; int_t |] false);
ignore(L.struct_set_body ctype_t [|
        ctype_add_pt;
        ctype_sub_pt;
        ctype_mul_pt;
  ctype_div_pt;
  ctype_exp_pt;
  ctype_eq_pt;
  ctype_neq_pt;
  ctype_lesser_pt;
        ctype_leq_pt;
        ctype_greater_pt;
        ctype_geq_pt;
        ctype_and_pt;
        ctype_or_pt;
        ctype_neg_pt;
  ctype_not_pt;
        ctype_idx_pt;
  ctype_call_pt;
  ctype_heapify_pt;
        ctype_print_pt |] false);

 let get_t = function
   | "int" -> int_t
   | "float" -> float_t
   | "bool" -> bool_t
   | "char" -> char_t
   | "list" -> clist_t
 in

 let build_ctype_fn fname ftype = (* ftype = "ctype_add_t" etc *)
   let the_function = L.define_function fname ftype the_module in
   let builder = L.builder_at_end context (L.entry_block the_function) in
   (the_function, builder)
 in

(* here's how you go from a cobj to the data value: *)
let build_getdata_cobj data_type cobj_p b =  (* data_type = int_t etc *)
  (*let x1 = L.build_load (lookup_global_binding "a") "x1" b in*)
  let x2 = L.build_struct_gep cobj_p cobj_data_idx "x2" b in
  let x3 = L.build_load x2 "x3" b in
  let x4 = L.build_bitcast x3 (L.pointer_type data_type) "x4" b in
  let data = L.build_load x4 "data" b in
  data
in

(* here's how you go from a cobj to the data value: *)
let build_gettype_cobj cobj_p b =  (* data_type = int_t etc *)
  let x2 = L.build_struct_gep cobj_p cobj_type_idx "x2" b in
  let x3 = L.build_load x2 "x3" b in
  let x4 = L.build_bitcast x3 (L.pointer_type ctype_t) "x4" b in
  x4
in
```

```
let build_getlist_cobj cobj_p b =
  let gep_addr = L.build_struct_gep cobj_p cobj_data_idx "__gep_addr" b in
  let objptr = L.build_load gep_addr "__objptr" b in
  L.build_bitcast objptr clist_pt "__clistptr" b
in

let build_getlen_clist clist_p b =
  let gep_addr = L.build_struct_gep clist_p clist_len_idx "__gep_addr" b in
  let gep_addr_as_intptr = L.build_bitcast gep_addr int_pt "__gep_addr_as_intptr" b in
  let length = L.build_load gep_addr_as_intptr "__length" b in
  length
in

let build_getcap_clist clist_p b =

  let gep_addr = L.build_struct_gep clist_p clist_cap_idx "__gep_addr" b in
  let gep_addr_as_intptr = L.build_bitcast gep_addr int_pt "__gep_addr_as_intptr" b in
  let capacity = L.build_load gep_addr_as_intptr "__capacity" b in
  capacity
in

let build_idx self_p other_p name b =

  (* TODO: throw error if array bounds exceeded *)
  let capacity = build_getcap_clist self_p b in
  let inbounds = L.build_icmp L.Icmp.Slt other_p capacity "__inbounds" b in (* other_p is index be

  (* get elememnt *)
  let gep_addr = L.build_struct_gep self_p clist_data_idx "__gep_addr" b in
  let gep_addr_as_cobjptrptrptr = L.build_bitcast gep_addr (L.pointer_type (L.pointer_type cobj_pt
  let gep_addr_as_cobjptrptr = L.build_load gep_addr_as_cobjptrptrptr "__gep_addr_as_cobjptrptr" b
  let gep_addr_as_cobjptrptr = L.build_gep gep_addr_as_cobjptrptr [| other_p |] "__gep_addr_as_cob
  let cobjptr = L.build_load gep_addr_as_cobjptrptr "__cobjptr" b in
  cobjptr
in

let built_ops =
         let typs = ["int"; "float"; "bool"; "char"; "list"] in

         let ops = [
           Oprt("add", Some((L.build_add), int_t), Some((L.build_fadd), float_t), None, None, None
     Oprt("sub", Some((L.build_sub), int_t), Some((L.build_fsub), float_t), None, None, None);
     Oprt("mul", Some((L.build_mul), int_t), Some((L.build_fmul), float_t), None, None, None);
     Oprt("div", Some((L.build_sdiv), int_t), Some((L.build_fdiv), float_t), None, None, None);
     Oprt("exp", None, None, None, None, None);
     Oprt("eq", Some((L.build_icmp L.Icmp.Eq), bool_t), Some((L.build_fcmp L.Fcmp.Ueq), bool_t), S
     Oprt("neq", Some((L.build_icmp L.Icmp.Ne), bool_t), Some((L.build_fcmp L.Fcmp.Une), bool_t),
     Oprt("lesser", Some((L.build_icmp L.Icmp.Slt), bool_t), Some((L.build_fcmp L.Fcmp.Ult), bool_
     Oprt("leq", Some((L.build_icmp L.Icmp.Sle), bool_t), Some((L.build_fcmp L.Fcmp.Ule), bool_t),
     Oprt("greater", Some((L.build_icmp L.Icmp.Sgt), bool_t), Some((L.build_fcmp L.Fcmp.Ugt), bool
     Oprt("geq", Some((L.build_icmp L.Icmp.Sge), bool_t), Some((L.build_fcmp L.Fcmp.Uge), bool_t),
     Oprt("and", Some((L.build_and), int_t), None, Some((L.build_and), bool_t), Some((L.build_add)
     Oprt("or", Some((L.build_or), int_t), None, Some((L.build_or), bool_t), Some((L.build_or), ch
```

```
Uoprt("neg", Some((L.build_neg), int_t), Some((L.build_fneg), float_t), Some((L.build_neg), b
Uoprt("not", Some((L.build_not), int_t), None, Some((L.build_not), bool_t), Some((L.build_not
Loprt("idx", None, None, None, None, Some((build_idx), int_t))
] in

    List.map (fun t -> let bops = List.map (function
       | Oprt(o, i, f, b, c, l) ->
   let tfn = match t with
     | "int" -> i
     | "float" -> f
     | "bool" -> b
     | "char" -> c
     | "list" -> l
              in
              let bop = match tfn with
                      | Some tfn ->
                        let (fn, bd) = build_ctype_fn (t ^ "_" ^ o) ((function
                          | "add" -> ctype_add_t
         | "sub" -> ctype_sub_t
         | "mul" -> ctype_mul_t
         | "div" -> ctype_div_t
         | "exp" -> ctype_exp_t
         | "eq" -> ctype_eq_t
         | "neq" -> ctype_neq_t
         | "lesser" -> ctype_lesser_t
         | "leq" -> ctype_leq_t
         | "greater" -> ctype_greater_t
         | "geq" -> ctype_geq_t
         | "and" -> ctype_and_t
         | "or" -> ctype_or_t) o)
                                in BOprt(Some(((fn, bd), tfn)))
                      | None -> BOprt(None)
   in bop
 | Uoprt(o, i, f, b, c, l) ->
   let tfn = match t with
     | "int" -> i
     | "float" -> f
     | "bool" -> b
     | "char" -> c
     | "list" -> l
   in
              let bop = match tfn with
                      | Some tfn ->
                        let (fn, bd) = build_ctype_fn (t ^ "_" ^ o) ((function
         | "neg" -> ctype_neg_t
         | "not" -> ctype_not_t) o)
                                in BUoprt(Some(((fn, bd), tfn)))
                      | None -> BUoprt(None)
              in bop
       | Loprt(o, i, f, b, c, l) ->
   let tfn = match t with
     | "int" -> i
     | "float" -> f
     | "bool" -> b
```

```
          | "char" -> c
          | "list" -> l
                    in
                    let bop = match tfn with
                            | Some tfn ->
                              let (fn, bd) = build_ctype_fn (t ^ "_" ^ o) ((function
              | "idx" -> ctype_idx_t) o)
                                      in BLoprt(Some(((fn, bd), tfn)))
                            | None -> BLoprt(None)
        in bop) ops
      in (t, bops)) typs
    in

(* Functions! *)
let (func_call_fn,func_call_b) = build_ctype_fn "func_call" ctype_call_t in

let ctype_func = L.define_global "ctype_func" (L.const_named_struct ctype_t [|
      L.const_pointer_null ctype_add_pt; (* ctype_add_pt *)
      L.const_pointer_null ctype_sub_pt; (* ctype_sub_pt *)
      L.const_pointer_null ctype_mul_pt; (* ctype_mul_pt *)
      L.const_pointer_null ctype_div_pt; (* ctype_div_pt *)
      L.const_pointer_null ctype_exp_pt; (* ctype_exp_pt *)
      L.const_pointer_null ctype_eq_pt; (* ctype_eq_pt *)
      L.const_pointer_null ctype_neq_pt; (* ctype_neq_pt *)
      L.const_pointer_null ctype_lesser_pt; (* ctype_lesser_pt *)
      L.const_pointer_null ctype_leq_pt; (* ctype_leq_pt *)
      L.const_pointer_null ctype_greater_pt; (* ctype_greater_pt *)
      L.const_pointer_null ctype_geq_pt; (* ctype_geq_pt *)
      L.const_pointer_null ctype_and_pt; (* ctype_and_pt *)
      L.const_pointer_null ctype_or_pt; (* ctype_or_pt *)
      L.const_pointer_null ctype_neg_pt; (* ctype_neg_pt *)
      L.const_pointer_null ctype_not_pt; (* ctype_not_pt *)
      L.const_pointer_null ctype_idx_pt; (* ctype_not_pt *)
      func_call_fn; (* ctype_call_pt *)
      L.const_pointer_null ctype_heapify_pt; (* ctype_not_pt *)
      L.const_pointer_null ctype_print_pt; (* ctype_not_pt *)
  |]) the_module in



let build_fnptr_of_cfo cobj_p b =
  let x2 = L.build_struct_gep cobj_p cobj_data_idx "x2" b in
  let x3 = L.build_load x2 "x3" b in
  let fnptr = L.build_bitcast x3 userdef_fn_pt "fnptr" b in
  fnptr
in

let build_special_ctype_fn fn ty_str =
    let (name,fn_ty) = (match fn with
      |FPrint -> ("print",ctype_print_t)
      |FHeapify -> ("heapify",ctype_heapify_t)
      |FCall -> ("call",ctype_call_t)
    ) in
    let (the_fn,the_bld) = build_ctype_fn (ty_str^"_"^name) fn_ty in
```

```
    (the_fn,the_bld)
in

(*let x = [
    (FPrint,[Int,Bool,Float]);
    (FHeapify,[Int,Bool,Float]);
    (FCall,[FuncType])
]

let build_fns spec_ty ty_list = List.map (build_special_ctype_fn spec_ty) ty_list in
let all = List.map (fun x -> match x with (spec_ty,ty_list) in build_fns spec_ty ty_list) x in
let get_special_fn spec_ty ty =
  match *)


(* Print *)
let (int_print_fn,int_print_b) = build_special_ctype_fn FPrint "int" in
let (char_print_fn,char_print_b) = build_special_ctype_fn FPrint "char" in
let (float_print_fn,float_print_b) = build_special_ctype_fn FPrint "float" in
let (bool_print_fn,bool_print_b) = build_special_ctype_fn FPrint "bool" in
let get_print_fn_lval = function
  |"int" -> int_print_fn
  |"float" -> float_print_fn
  |"char" -> char_print_fn
  |"bool" -> bool_print_fn
  | _ -> L.const_pointer_null ctype_print_pt
in
let get_print_builder = function
  |"int" -> int_print_b
  |"float" -> float_print_b
  |"char" -> char_print_b
  |"bool" -> bool_print_b
in

(* Heapify *)
let (int_heapify_fn,int_heapify_b) = build_special_ctype_fn FHeapify "int" in
let (float_heapify_fn,float_heapify_b) = build_special_ctype_fn FHeapify "float" in
let (bool_heapify_fn,bool_heapify_b) = build_special_ctype_fn FHeapify "bool" in
let (func_heapify_fn,func_heapify_b) = build_special_ctype_fn FHeapify "func" in
let get_heapify_fn_lval = function
  |"int" -> int_heapify_fn
  |"float" -> float_heapify_fn
  |"bool" -> bool_heapify_fn
  |"func" -> func_heapify_fn
  | _ -> L.const_pointer_null ctype_heapify_pt
in

(* define the default CTypes *)
let [ctype_int; ctype_float; ctype_bool; ctype_char; ctype_list] =
        List.map (fun (t, bops) -> L.define_global ("ctype_" ^ t) (L.const_named_struct ctype_t (A
          | BOprt(o) -> (match o with
            | Some(((fn, bd), tfn)) -> fn
            | None -> L.const_pointer_null ctype_add_pt)
          | BUoprt(o) -> (match o with
```

```
            | Some(((fn, bd), tfn)) -> fn
            | None -> L.const_pointer_null ctype_neg_pt)
          | BLoprt(o) -> (match o with
            | Some(((fn, bd), tfn)) -> fn
      | None -> L.const_pointer_null ctype_idx_pt)) bops) @ ([L.const_pointer_null ctype_call_pt;
            in

let ctype_of_ASTtype = function
  | Int -> Some ctype_int
  | Float -> Some ctype_float
  | Bool -> Some ctype_bool
  | String -> Some ctype_list
  | Dyn -> None
  | IntArr -> Some ctype_list
  | FloatArr -> Some ctype_list
  | BoolArr -> Some ctype_list
  | StringArr -> Some ctype_list
  | FuncType -> Some ctype_func
  | Null -> None
in

let ctype_of_datatype = function
  | dt when dt = int_t -> ctype_int
  | dt when dt = float_t -> ctype_float
  | dt when dt = bool_t -> ctype_bool
  | dt when dt = char_t -> ctype_char
  | dt when dt = clist_t -> ctype_list
in
let ctype_of_typ = function  (* only for optimized Raws hence limited matching *)
    |Int -> ctype_int
    |Float -> ctype_float
    |Bool -> ctype_bool
in

let build_getctypefn_cobj ctype_fn_idx cobj_p b =
  let x2 = L.build_struct_gep cobj_p cobj_type_idx "x2" b in
  let x3 = L.build_load x2 "x3" b in  (* x3: ctype_pt *)
  let x4 = L.build_struct_gep x3 ctype_fn_idx "x4" b in
  let fn_ptr = L.build_load x4 "fn_ptr" b in
  fn_ptr
in

(** define helper functions for commonly used code snippets **)
let build_new_cobj_empty builder =
  let objptr = L.build_malloc cobj_t "__new_objptr" builder in (* objptr: cobj_pt* *)
  let datafieldptr = L.build_struct_gep objptr cobj_data_idx "datafieldptr" builder in  (* datafie
  let ctypefieldptr = L.build_struct_gep objptr cobj_type_idx "ctypefieldptr" builder in
  (objptr,datafieldptr,ctypefieldptr)
in

let build_new_cobj data_type builder =
  (* malloc the new object and its data *)
  let objptr = L.build_malloc cobj_t "__new_objptr" builder in (* objptr: cobj_pt *)
  let dataptr = L.build_malloc data_type "__new_dataptr" builder in
```

```
  let dataptr_as_i8ptr = L.build_bitcast dataptr char_pt "dataptr_as_i8" builder in

  (* store ctypeptr in the struct *)
  let ctypefieldptr = L.build_struct_gep objptr cobj_type_idx "ctypefieldptr" builder in
  ignore(L.build_store (ctype_of_datatype data_type) ctypefieldptr builder);

  (* store dataptr in the struct *)
  let datafieldptr = L.build_struct_gep objptr cobj_data_idx "datafieldptr" builder in  (* datafie
  let datafieldptr_as_i8ptrptr = L.build_bitcast datafieldptr (L.pointer_type char_pt) "datafieldp
  ignore(L.build_store dataptr_as_i8ptr datafieldptr_as_i8ptrptr builder);

  (objptr, dataptr)
in

let build_new_cobj_init data_type value b =
    let (objptr, dataptr) = build_new_cobj data_type b in
    ignore(L.build_store value dataptr b);
    objptr
in

let build_new_clist dataptr_of_cobj elm_pts builder =
  (* len *)
  let length = List.length elm_pts in
  let len = L.const_int int_t length in

  (* cap *)
  let capacity = max length initial_list_size in
  let cap = L.const_int int_t capacity in

  (* dataptr: mallocs empty CObj array *)
  let dataptr = L.build_malloc (L.array_type cobj_pt capacity) "__new_dataptr" builder in
  let dataptr_as_i8ptr = L.build_bitcast dataptr char_pt "dataptr_as_i8" builder in

  let elm_pts_as_cobjptrs = List.map (fun e ->
    let elm_pt_as_cobjptr = L.build_bitcast e cobj_pt "elm_ptr_as_cobjptr" builder
    in elm_pt_as_cobjptr) elm_pts in

  (* null pointers to fill empty capacity *)
  let elms_w_nulls = if List.length elm_pts_as_cobjptrs < capacity
    then elm_pts_as_cobjptrs@(Array.to_list (Array.make (capacity - List.length elm_pts) (L.const_
    else elm_pts_as_cobjptrs in

  (* stores the data *)
  let store_elms elm idx =
    let gep_addr = L.build_gep dataptr [|L.const_int int_t 0; L.const_int int_t idx|] "__elem_ptr"
    ignore(L.build_store elm gep_addr builder); ()
  in
  ignore(List.iter2 store_elms elms_w_nulls (seq capacity));

  (* store dataptr the struct *)
  let datafieldptr = L.build_struct_gep dataptr_of_cobj clist_data_idx "datafieldptr" builder in
  let datafieldptr_as_i8ptrptr = L.build_bitcast datafieldptr (L.pointer_type char_pt) "datafieldp
  ignore(L.build_store dataptr_as_i8ptr datafieldptr_as_i8ptrptr builder);
```

```
  (* store len in the struct *)
  let lenfieldptr = L.build_struct_gep dataptr_of_cobj clist_len_idx "lenfieldptr" builder in  (*
  ignore(L.build_store len lenfieldptr builder);

  (* store cap in the struct *)
  let capfieldptr = L.build_struct_gep dataptr_of_cobj clist_cap_idx "capfieldptr" builder in  (*
  ignore(L.build_store cap capfieldptr builder);
in

(** manually making the ctype_ functions **)
(* does alloca, store, then load *)  (* note you should not use this if youre not using the values
let boilerplate_till_load remote_cobj_p prettyname b =
  ignore(L.set_value_name ("remote_"^prettyname) remote_cobj_p);
  let cobj_pp = L.build_alloca cobj_pt (prettyname^"_p") b in
  ignore(L.build_store remote_cobj_p cobj_pp b);
  let cobj_p = L.build_load cobj_pp (prettyname^"_p") b in
  cobj_p
in

let boilerplate_binop data_type fn b =
  let formals_llvalues = (Array.to_list (L.params fn)) in
  let [ remote_self_p; remote_other_p ] = formals_llvalues in

  (* boilerplate *)
  let self_p = boilerplate_till_load remote_self_p "self_p" b in
  let other_p = boilerplate_till_load remote_other_p "other_p" b in

  (* get data *)
  let self_data = build_getdata_cobj data_type self_p b in
  let other_data = build_getdata_cobj data_type other_p b in
  (self_data, other_data)
in

let boilerplate_uop data_type fn b =
  let formals_llvalues = (Array.to_list (L.params fn)) in
  let [ remote_self_p ] = formals_llvalues in


  let _ = build_gettype_cobj remote_self_p b in

  (* boilerplate *)
  let self_p = boilerplate_till_load remote_self_p "self_p" b in

  (* get data *)
  let self_data = build_getdata_cobj data_type self_p b in
  (self_data)
in

let boilerplate_lop data_type fn b =
    (* TODO: throw error if array bounds exceeded *)
  let formals_llvalues = Array.to_list (L.params fn) in
  let [ remote_self_p; remote_other_p ] = formals_llvalues in

  (* boilerplate *)
```

```
  let self_p = boilerplate_till_load remote_self_p "self_p" b in
  let other_p = boilerplate_till_load remote_other_p "other_p" b in

  (* get data *)
  let self_data = build_getlist_cobj self_p b in
  let other_data = build_getdata_cobj int_t other_p b in
  (self_data, other_data)
in

List.iter (fun (t, bops) -> List.iter (function
  | BOprt(o) -> (match o with
    | Some(((fn, bd), tfn)) ->
      let (tf, tp) = tfn in
      let (self_data, other_data) = boilerplate_binop (get_t t) fn bd in
      let result_data = tf self_data other_data "result_data" bd in
      let result = build_new_cobj_init tp result_data bd in
      ignore(L.build_ret result bd)
    | None -> ())
  | BUoprt(o) -> (match o with
    | Some(((fn, bd), tfn)) ->
      let (tf, tp) = tfn in
      let (self_data) = boilerplate_uop (get_t t) fn bd in
      let result_data = tf self_data "result_data" bd in
      let result = build_new_cobj_init tp result_data bd in
      ignore(L.build_ret result bd)
    | None -> ())
  | BLoprt(o) -> (match o with
    | Some(((fn, bd), tfn)) ->
      let (tf, tp) = tfn in
      let (self_data, other_data) = boilerplate_lop (get_t t) fn bd in
      let result_data = tf self_data other_data "result_data" bd in
      let result = result_data in
      ignore(L.build_ret result bd)
    | None -> ())) bops) built_ops;


  (* Functions! *)
  (* building __call__ for ctype_func *)
      let (fn,b) = (func_call_fn,func_call_b) in
        let formals_llvalues = (Array.to_list (L.params fn)) in
        let [ remote_self_p ; remote_argv ] = formals_llvalues in
        let self_p = boilerplate_till_load remote_self_p "self_p" b in

        (* manual boilerplate for argv since it's not a cobj *)
        ignore(L.set_value_name ("remote_argv") remote_argv);
        let argv_p = L.build_alloca cobj_ppt "argv_p" b in
        ignore(L.build_store remote_argv argv_p b);
        let argv = L.build_load argv_p "argv" b in

        let fn_p = build_fnptr_of_cfo self_p b in
        let result = L.build_call fn_p [|argv|] "result" b in
        ignore(L.build_ret result b);
```

```
(* heapify(self_p) modifies self by copying its data to the heap and pointing to the new heap data
let build_heapify data_type fn b =   (* data_type = int_t etc *)
    let formals_llvalues = (Array.to_list (L.params fn)) in
    let [remote_self_p] = formals_llvalues in
    let box_addr = boilerplate_till_load remote_self_p "self_p" b in
    (* the box dataptr_addr points to the raw data we want to copy *)
    let dataptr_addr_i8pp = L.build_struct_gep box_addr cobj_data_idx "dat" b in
    let dataptr_addr = L.build_bitcast dataptr_addr_i8pp (L.pointer_type (L.pointer_type data_type
    let rawdata_addr = L.build_load dataptr_addr "raw_data_addr" b in
    let rawdata = L.build_load rawdata_addr "raw_data" b in
    let heap_data_p = L.build_malloc data_type "heap_data_p" b in
    ignore(L.build_store rawdata heap_data_p b);
    let heap_data_p = L.build_bitcast heap_data_p char_pt "heap_data_p" b in
    ignore(L.build_store heap_data_p dataptr_addr_i8pp b);
    ignore(L.build_ret (L.const_int int_t 0) b);  (* or can ret void? *)
in

(* build the heapify functions *)
let get_heapify_builder_of_t = function
  |"int" -> int_heapify_b
  |"float" -> float_heapify_b
  |"bool" -> bool_heapify_b
in
ignore(List.iter (fun t -> build_heapify (get_t t) (get_heapify_fn_lval t) (get_heapify_builder_of

let _ =
  let (fn,b) = (func_heapify_fn,func_heapify_b) in
  ignore(L.build_ret (L.const_int int_t 0) b);
in




(* define printf *)
let printf_t : L.lltype =   (* define the type that the printf function should be *)
    L.var_arg_function_type int_t [| char_pt |] in
let printf_func : L.llvalue =   (* now use that type to declare printf (dont fill out the body jus
    L.declare_function "printf" printf_t the_module in

let build_print_fn data_type fn b =   (* data_type = int_t etc *)
    let formals_llvalues = (Array.to_list (L.params fn)) in
    let [remote_self_p] = formals_llvalues in
    let box_addr = boilerplate_till_load remote_self_p "self_p" b in
    (* the box dataptr_addr points to the raw data we want to copy *)
    let dataptr_addr_i8pp = L.build_struct_gep box_addr cobj_data_idx "dat" b in
    let dataptr_addr = L.build_bitcast dataptr_addr_i8pp (L.pointer_type (L.pointer_type data_type
    let rawdata_addr = L.build_load dataptr_addr "raw_data_addr" b in
    let rawdata = L.build_load rawdata_addr "raw_data" b in
    let format_str = (match data_type with
      |t when t = int_t -> L.build_global_stringptr "%d\n" "fmt" b
      |t when t = float_t -> L.build_global_stringptr "%g\n" "fmt" b
      |t when t = bool_t -> L.build_global_stringptr "%d\n" "fmt" b
      |t when t = char_t -> L.build_global_stringptr "%s\n" "fmt" b
    ) in
```

```
    ignore(L.build_call printf_func [| format_str ; rawdata |] "printf" b);
    ignore(L.build_ret (L.const_int int_t 0) b);  (* or can ret void? *)
in
(* build the print functions *)
ignore(List.iter (fun t -> build_print_fn (get_t t) (get_print_fn_lval t) (get_print_builder t)) [

(* define exit *)
let exit_t : L.lltype =   (* define the type that the printf function should be *)
  L.function_type (int_t) [| int_t |] in
let exit_func : L.llvalue =   (* now use that type to declare printf (dont fill out the body just
    L.declare_function "exit" exit_t the_module in


let name_of_bind = function
    |Bind(name,_) -> name
in
let type_of_bind = function
    |Bind(_,ty) -> ty
in
let ltyp_of_typ = function
    |Int -> int_t
    |Float -> float_t
    |Bool -> bool_t
    |_ -> cobj_pt
    (** todo lists and stuff **)
in

(** allocate for all the bindings and put them in a map **)

(* pass in Some(builder) to do local vars alloca() or None to do globals non-alloca *)
let build_binding_list local_builder_opt binds =   (* returns a stringmap Bind -> Addr *)
(* strip out all the FuncTypes from binds *)
    (*let binds = List.rev (List.fold_left (fun binds bind -> if ((type_of_bind bind) = FuncType)
    (** the commented code adds a Dyn version of every var. i wont use it for pure immutable phase
    (**let dynify bind =   (* turns a bind into dynamic. a helper fn *)
       Bind(name,_) = bind in
         Bind(name,Dyn)
    in
    let dyns_list =   (* redundant list where every bind is dynamic *)
        List.map dynify (names_of_bindlist binds)
    in
    binds = List.sort_uniq Pervasives.compare (binds @ dyns_list)
    in   (* now binds has a dyn() version of each variable *) **)
    let prettyname_of_bind bind = (name_of_bind bind)^"_"^(string_of_typ (type_of_bind bind))
    in
    let get_const bind = match (type_of_bind bind) with
      |Int -> L.const_null int_t
      |Float -> L.const_null float_t
      |Bool -> L.const_null bool_t
      |_ -> L.define_global ((prettyname_of_bind bind)^"_obj") (L.const_named_struct cobj_t [|L.co
      (*L.define_global (prettyname_of_bind bind) (the_cobj) the_module*)

      (*|_ -> L.define_global (prettyname_of_bind bind) (L.const_null cobj_t) the_module*)
      (** TODO impl lists and everything! and strings. idk how these will work **)
```

```
      in
      let allocate bind =
        let alloc_result =
          (match local_builder_opt with
            |None -> L.define_global (prettyname_of_bind bind) (get_const bind) the_module
            |Some(builder) -> L.build_alloca (ltyp_of_typ (type_of_bind bind)) (prettyname_of_bind b
          )
        in
        let (res,newbind) = match (type_of_bind bind) with
          |Int|Float|Bool -> (RawAddr(alloc_result),bind)
          |_ -> (BoxAddr(alloc_result,false),Bind((name_of_bind bind),Dyn))
        in (res,newbind)
      in
        List.fold_left (fun map bind ->
            let (res,newbind) = allocate bind in
          BindMap.add newbind res map) BindMap.empty binds
  in


  let globals_map =
      let globals_list = snd prgm  (* snd prgrm is the bind list of globals *) in
        build_binding_list None globals_list
  in
  let lookup_global_binding bind =   (*pbind bind;*)
    try BindMap.find bind globals_map
    with Not_found -> tstp "Not found in globals:";pbind bind;BindMap.find bind globals_map
  in


  (** setup main() where all the code will go **)
  let main_ftype = L.function_type int_t [||] in   (* ftype is the full llvm function signature *)
  let main_function = L.define_function "main" main_ftype the_module in
  let main_builder = L.builder_at_end context (L.entry_block main_function) in
  let int_format_str = L.build_global_stringptr "%d\n" "fmt" main_builder
  and string_format_str = L.build_global_stringptr "%s\n" "fmt" main_builder
  and float_format_str = L.build_global_stringptr "%g\n" "fmt" main_builder in
  let init_state:state = {ret_typ=Int;namespace=BindMap.empty; func=main_function; b=main_builder;op


  (* useful utility functions! *)
  let names_of_bindlist bindlist =
    List.map name_of_bind bindlist
  in
(* helper fn: seq 4 == [0;1;2;3] *)
  let seq len =
    let rec aux len acc =
      if len<0 then acc else aux (len-1) (len::acc)
    in aux (len-1) []
  in

  let lookup namespace bind = (*tstp (string_of_sbind bind);*)
      let bind = match bind with
        |Bind(n,Int)|Bind(n,Float)|Bind(n,Bool) -> bind
        |Bind(n,_) -> Bind(n,Dyn)
```

```
    in
    try BindMap.find bind namespace
      with Not_found -> lookup_global_binding bind
in

let build_temp_box rawval raw_ty b =
    tstp ("Building temp box for "^string_of_typ raw_ty);
    let raw_ltyp = (ltyp_of_typ raw_ty) in
    let temp_box = L.build_alloca cobj_t "temp_box" b in
    let heap_data_p = L.build_malloc raw_ltyp "heap_data_temp_box" b in
    ignore(L.build_store rawval heap_data_p b);
    let heap_data_p = L.build_bitcast heap_data_p char_pt "heap_data_p" b in
    let dataptr_addr = L.build_struct_gep temp_box cobj_data_idx "dat" b in
    let typeptr_addr = L.build_struct_gep temp_box cobj_type_idx "ty" b in
    let typeptr_addr = L.build_bitcast typeptr_addr (L.pointer_type ctype_pt) "ty" b in
    ignore(L.build_store heap_data_p dataptr_addr b);
    ignore(L.build_store (ctype_of_typ raw_ty) typeptr_addr b);
    Box(temp_box)
in
let rec change_state old = function
    | S_rettyp(ret_typ) -> {ret_typ=ret_typ;namespace=old.namespace;func=old.func;b=old.b;optim_fu
    | S_names(namespace) -> {ret_typ=old.ret_typ;namespace=namespace;func=old.func;b=old.b;optim_f
    | S_func(func) -> {ret_typ=old.ret_typ;namespace=old.namespace;func=func;b=old.b;optim_funcs=o
    | S_b(b) -> {ret_typ=old.ret_typ;namespace=old.namespace;func=old.func;b=b;optim_funcs=old.opt
    | S_optimfuncs(optim_funcs) -> {ret_typ=old.ret_typ;namespace=old.namespace;func=old.func;b=ol
    | S_generic_func(boolval) -> {ret_typ=old.ret_typ;namespace=old.namespace;func=old.func;b=old.
  | S_needs_reboxing(name,boolval) ->
    let BoxAddr(addr,_) = lookup (old.namespace) (Bind(name,Dyn)) in
    let new_namespace = BindMap.add (Bind(name,Dyn)) (BoxAddr(addr,boolval)) old.namespace in
    change_state old (S_names(new_namespace))
  | S_list(updates) -> List.fold_left change_state old updates
in
let rebox_if_needed boxaddr name the_state =
  match boxaddr with
    |BoxAddr(addr,true) -> tstp ("Boxing "^name);
      let cobj_p = L.build_load addr name the_state.b in
      let fn_p = build_getctypefn_cobj ctype_heapify_idx cobj_p the_state.b in
      ignore(L.build_call fn_p [|cobj_p|] "heapify_result" the_state.b);
      change_state the_state (S_needs_reboxing(name,false))
    |BoxAddr(_,false) -> the_state  (* do nothing *)
  in

let rec expr the_state typed_e =
    let (namespace,the_function) = (the_state.namespace,the_state.func) in
    let (e,ty) = typed_e in
    match e with
  | SLit lit -> let res = (match lit with
      | IntLit i -> Raw(L.const_int int_t i)
      | BoolLit i -> Raw(L.const_int bool_t (if i then 1 else 0))
      | FloatLit i -> Raw((L.const_float float_t i))
      | StringLit i -> let elements = List.rev (Seq.fold_left (fun l ch ->
          let cobj_of_char_ptr = build_new_cobj_init char_t (L.const_int char_t (Char.code ch)) th
          cobj_of_char_ptr::l) [] (String.to_seq i)) in
        let (objptr, dataptr) = build_new_cobj clist_t the_state.b in
```

```
      let _ = build_new_clist dataptr elements the_state.b in
        (Box(objptr))
    ) in (res,the_state)

| SVar name ->
    (match (lookup namespace (Bind(name, ty))) with
      |RawAddr(addr) -> (Raw(L.build_load addr name the_state.b),the_state)
      |BoxAddr(addr,needs_update) ->
         let the_state = rebox_if_needed (BoxAddr(addr,needs_update)) name the_state in
        (Box(L.build_load addr name the_state.b),the_state)
    )
| SBinop(e1, op, e2) ->
  let (_,ty1) = e1
  and (_,ty2) = e2 in
  let (e1',the_state) = expr the_state e1 in
  let (e2',the_state) = expr the_state e2 in

  let generic_binop box1 box2 =
      let (Box(v1),Box(v2)) = (box1,box2) in
      let fn_idx = (match op with
        | Add      -> ctype_add_idx
        | Sub      -> ctype_sub_idx
        | Mul      -> ctype_mul_idx
        | Div      -> ctype_div_idx
        | Exp      -> ctype_exp_idx
        | Eq       -> ctype_eq_idx
        | Neq      -> ctype_neq_idx
        | Less     -> ctype_lesser_idx
        | Leq      -> ctype_leq_idx
        | Greater  -> ctype_greater_idx
        | Geq      -> ctype_geq_idx
        | And      -> ctype_and_idx
        | Or       -> ctype_or_idx
        | ListAccess -> ctype_idx_idx ) in
    let fn_p = build_getctypefn_cobj fn_idx v1 the_state.b in


    (* exception handling: invalid_op *)
    let bad_op_bb = L.append_block context "bad_op" the_state.func in
    let bad_op_bd = L.builder_at_end context bad_op_bb in

    let proceed_bb = L.append_block context "proceed" the_state.func in

    (* check for op exception *)
    let invalid_op = L.build_is_null fn_p "invalid_op" the_state.b in
      ignore(L.build_cond_br invalid_op bad_op_bb proceed_bb the_state.b);

    (* print message and exit *)
    let err_message =
      let info = "invalid use of " ^ (Utilities.binop_to_string op) ^ " operator" in
        L.build_global_string info "error message" bad_op_bd in
    let str_format_str1 = L.build_global_stringptr  "%s\n" "fmt" bad_op_bd in
      ignore(L.build_call printf_func [| str_format_str1 ; err_message |] "printf" bad_op_bd);
      ignore(L.build_call exit_func [| (L.const_int int_t 1) |] "exit" bad_op_bd);
```

```
  (* return to normal control flow *)
  let the_state = change_state the_state (S_b(L.builder_at_end context proceed_bb)) in
    ignore(L.build_br proceed_bb bad_op_bd);

  (* exception handling: invalid_arg *)
  let bad_arg_bb = L.append_block context "bad_arg" the_state.func in
  let bad_arg_bd = L.builder_at_end context bad_arg_bb in

  let proceed_bb = L.append_block context "proceed" the_state.func in

  (* check for arg exception *)

  let _ = match op with
    | ListAccess ->
      let typ1 = ctype_int in
      let typ2 = build_gettype_cobj v2 the_state.b in
      let typ1_as_int = L.build_ptrtoint typ1 int_t "typ1_as_int" the_state.b in
      let typ2_as_int = L.build_ptrtoint typ2 int_t "typ2_as_int" the_state.b in
      let diff = L.build_sub typ1_as_int typ2_as_int "diff" the_state.b in
      let invalid_arg = L.build_icmp L.Icmp.Ne diff (L.const_int int_t 0) "invalid_arg" the_st
        ignore(L.build_cond_br invalid_arg bad_arg_bb proceed_bb the_state.b);
    | _ ->
      let typ1 = build_gettype_cobj v1 the_state.b in
      let typ2 = build_gettype_cobj v2 the_state.b in
      let typ1_as_int = L.build_ptrtoint typ1 int_t "typ1_as_int" the_state.b in
      let typ2_as_int = L.build_ptrtoint typ2 int_t "typ2_as_int" the_state.b in
      let diff = L.build_sub typ1_as_int typ2_as_int "diff" the_state.b in
      let invalid_arg = L.build_icmp L.Icmp.Ne diff (L.const_int int_t 0) "invalid_arg" the_st
        ignore(L.build_cond_br invalid_arg bad_arg_bb proceed_bb the_state.b);
  in

  (* print message and exit *)
  let err_message =
    let info = "RuntimeError: unsupported operand type(s) for binary " ^ (Utilities.binop_to_s
      L.build_global_string info "error message" bad_arg_bd in
  let str_format_str1 = L.build_global_stringptr  "%s\n" "fmt" bad_arg_bd in
    ignore(L.build_call printf_func [| str_format_str1 ; err_message |] "printf" bad_arg_bd);
    ignore(L.build_call exit_func [| (L.const_int int_t 1) |] "exit" bad_arg_bd);

  (* return to normal control flow *)
  let the_state = change_state the_state (S_b(L.builder_at_end context proceed_bb)) in
    ignore(L.build_br proceed_bb bad_arg_bd);



  let result = L.build_call fn_p [| v1 ; v2 |] "binop_result" the_state.b in
  (Box(result),the_state)
in

let (res,the_state) = (match (e1',e2') with
    |(Raw(v1),Raw(v2)) ->
        let binop_instruction = (match ty1 with
          |Int|Bool -> (match op with
```

```
            | Add     -> L.build_add
            | Sub     -> L.build_sub
            | Mul    -> L.build_mul
            | Div     -> L.build_sdiv
            | And     -> L.build_and
            | Or      -> L.build_or
            | Eq   -> L.build_icmp L.Icmp.Eq
            | Neq     -> L.build_icmp L.Icmp.Ne
            | Less    -> L.build_icmp L.Icmp.Slt
            | Leq     -> L.build_icmp L.Icmp.Sle
            | Greater -> L.build_icmp L.Icmp.Sgt
            | Geq     -> L.build_icmp L.Icmp.Sge
          )
          |Float -> (match op with
            | Add     -> L.build_fadd
            | Sub     -> L.build_fsub
            | Mul    -> L.build_fmul
            | Div     -> L.build_fdiv
            | Eq   -> L.build_fcmp L.Fcmp.Oeq
            | Neq     -> L.build_fcmp L.Fcmp.One
            | Less    -> L.build_fcmp L.Fcmp.Olt
            | Leq     -> L.build_fcmp L.Fcmp.Ole
            | Greater -> L.build_fcmp L.Fcmp.Ogt
            | Geq     -> L.build_fcmp L.Fcmp.Oge
            | And | Or ->
                raise (Failure "internal error: semant should have rejected and/or on float")
          )
      ) in
        (Raw(binop_instruction v1 v2 "binop_result" the_state.b),the_state)
    |(Box(boxval),Raw(rawval)) ->
      generic_binop (Box(boxval)) (build_temp_box rawval ty2 the_state.b)
    |(Raw(rawval),Box(boxval)) ->
      generic_binop (build_temp_box rawval ty1 the_state.b) (Box(boxval))
    |(Box(v1),Box(v2)) -> generic_binop (Box(v1)) (Box(v2))
  ) in (res,the_state)
|SCall(fexpr, arg_expr_list, SNop) -> raise (Failure "Generic scalls partially implemented and
  tstp ("GENERIC SCALL of "^(string_of_int (List.length arg_expr_list))^" args");
  (* eval the arg exprs *)
  let argc = List.length arg_expr_list in

  let eval_arg aggreg e =
      let (the_state,args) = aggreg in
      let (res,the_state) = expr the_state e in
      (the_state,res::args)
  in
  let (the_state,arg_dataunits) = List.fold_left eval_arg (the_state,[]) (List.rev arg_expr_li

  let arg_types = List.map (fun (_,ty) -> ty) arg_expr_list in

  let transform_if_needed raw_ty = function
      |Box(v) -> Box(v)
      |Raw(v) -> build_temp_box v raw_ty the_state.b
  in
```

```
  let boxed_args = List.map2 transform_if_needed arg_types arg_dataunits in
  let llargs = List.map (fun b -> match b with Box(v) -> v) boxed_args in

  let cobj_p_arr_t = L.array_type cobj_pt argc in
  (* allocate stack space for argv *)
  let argv_as_arr = L.build_alloca cobj_p_arr_t "argv_arr" the_state.b in
  (* store llargs values in argv *)

  let store_arg llarg idx =
    let gep_addr = L.build_gep argv_as_arr [|L.const_int int_t 0; L.const_int int_t idx|] "arg
    ignore(L.build_store llarg gep_addr the_state.b);()
  in

  ignore(List.iter2 store_arg llargs (seq argc));
  let argv = L.build_bitcast argv_as_arr cobj_ppt "argv" the_state.b in

  (* now we have argv! so we just need to get the fn ptr and call it *)
  let (Box(caller_cobj_p),the_state) = expr the_state fexpr in
  let call_ptr = build_getctypefn_cobj ctype_call_idx caller_cobj_p the_state.b in
  let result = L.build_call call_ptr [|caller_cobj_p;argv|] "result" the_state.b in
  (Box(result),the_state)


|SCall(fexpr, arg_expr_list, SFunc(sfdecl)) ->
tstp ("OPTIMIZED SCALL of "^sfdecl.sfname^" with binds:"); List.iter pbind sfdecl.sformals; ts
  (*ignore(expr the_state fexpr);*) (* I guess we dont care abt the result of this since we ju
  (*let (_,the_state) = expr the_state fexpr in*)
  let arg_types = List.map (fun (_,ty) -> ty) arg_expr_list in
  let arg_lltypes = List.map ltyp_of_typ arg_types in
  let eval_arg aggreg e =
      let (the_state,args) = aggreg in
      let (res,the_state) = expr the_state e in
      (the_state,res::args)
  in
  let (the_state,arg_dataunits) = List.fold_left eval_arg (the_state,[]) (List.rev arg_expr_li
  let unwrap_if_raw = function  (* just to cause a crash if any Dyn *)
      |Raw(v) ->v
      |Box(v) ->v
  in
  let arg_vals = List.map unwrap_if_raw arg_dataunits in

  let optim_func = (match (SfdeclMap.find_opt sfdecl the_state.optim_funcs) with
    |Some(optim_func) -> optim_func
    |None -> tstp "(no optimized version found, generating new one)";
      (* now lets build the optimized function *)
      let formal_types = (Array.of_list arg_types) in
      let ftype = L.function_type (ltyp_of_typ sfdecl.styp) (Array.of_list arg_lltypes) in  (*
      let optim_func = L.define_function sfdecl.sfname ftype the_module in   (* define_functio

          (* now lets build the body of the optimized function *)
      let fn_builder = L.builder_at_end context (L.entry_block optim_func) in
      let int_format_str = L.build_global_stringptr "%d\n" "fmt" the_state.b
      and string_format_str = L.build_global_stringptr "%d\n" "fmt" the_state.b
      and float_format_str = L.build_global_stringptr "%g\n" "fmt" the_state.b in
```

```
        let fn_namespace = build_binding_list (Some(fn_builder)) (sfdecl.sformals @ sfdecl.sloca
        let vals_to_store = Array.to_list (L.params optim_func) in
        let addr_of_bind bind = match (lookup fn_namespace bind) with
            |RawAddr(addr) -> addr
            |BoxAddr(addr,_) -> addr  (* maybe use the flag! *)
        in
        let addrs = List.map addr_of_bind sfdecl.sformals in

        ignore(List.iter2 (fun addr value -> ignore(L.build_store value addr fn_builder)) addrs
        let fn_state = change_state the_state (S_list([S_names(fn_namespace);S_func(optim_func);
        let fn_state = stmt fn_state sfdecl.sbody in
        let fn_state = add_terminal fn_state (match sfdecl.styp with
            Null -> (fun b -> L.build_ret (build_new_cobj_init int_t (L.const_int int_t 0) b) b)
          | Float -> L.build_ret (L.const_float float_t 0.0)
          | Dyn -> (fun b -> L.build_ret (build_new_cobj_init int_t (L.const_int int_t 0) b) b)
          | t -> L.build_ret (L.const_int int_t 0)
        ) in optim_func
    ) in
    let result = L.build_call optim_func (Array.of_list arg_vals) "result" the_state.b in
    let the_state = change_state the_state (S_optimfuncs(SfdeclMap.add sfdecl optim_func the_sta
    let res=(match sfdecl.styp with
        |Int|Float|Bool -> Raw(result)
        |_ -> Box(result)
    ) in (res,the_state)
| SListAccess(e1,e2)  -> expr the_state (SBinop(e1,ListAccess,e2), ty)
|SUnop(op, e1) ->
  let (_,ty1) = e1 in
  let (e1',the_state) = expr the_state e1 in
  let (res,the_state) = (match e1' with
  |Box(v1) ->
      let fn_idx = match op with
        | Neg         -> ctype_neg_idx
        | Not         -> ctype_not_idx in
      let fn_p = build_getctypefn_cobj fn_idx v1 the_state.b in

      (* exception handling: invalid_op *)
      let bad_op_bb = L.append_block context "bad_op" the_state.func in
      let bad_op_bd = L.builder_at_end context bad_op_bb in

      let proceed_bb = L.append_block context "proceed" the_state.func in

      (* check for op exception *)
      let invalid_op = L.build_is_null fn_p "invalid_op" the_state.b in
        ignore(L.build_cond_br invalid_op bad_op_bb proceed_bb the_state.b);

      (* print message and exit *)
      let err_message =
        let info = "RuntimeError: unsupported operand type for unary " ^ (Utilities.unop_to_stri
          L.build_global_string info "error message" bad_op_bd in
      let str_format_str1 = L.build_global_stringptr  "%s\n" "fmt" bad_op_bd in
        ignore(L.build_call printf_func [| str_format_str1 ; err_message |] "printf" bad_op_bd);
        ignore(L.build_call exit_func [| (L.const_int int_t 1) |] "exit" bad_op_bd);

      (* return to normal control flow *)
```

```
        let the_state = change_state the_state (S_b(L.builder_at_end context proceed_bb)) in
        ignore(L.build_br proceed_bb bad_op_bd);

        let result = L.build_call fn_p [| v1 |] "uop_result" the_state.b in
        (Box(result), the_state)
      |Raw(v1) ->
             let res  = (match op with
        | Neg when ty1=Float         -> L.build_fneg
        | Neg         -> L.build_neg
        | Not         -> L.build_not
        ) v1 "unop_result" the_state.b in
             (Raw(res),the_state)
      ) in (res,the_state)

  | SList(el, t) ->
      let transform_if_needed raw_ty = function
          |Box(v) -> Box(v)
          |Raw(v) -> build_temp_box v raw_ty the_state.b
      in

    let (elements, the_state) = List.fold_left (fun (l, the_state) e ->
      let (element, the_state) = expr the_state e in
        (element::l, the_state)) ([], the_state) (List.rev el) in
    let (objptr, dataptr) = build_new_cobj clist_t the_state.b in
    let raw_ty = (match t with
      |IntArr -> Int
      |FloatArr -> Float
      |BoolArr -> Bool
    ) in
    let elements = List.map (fun elem -> match (transform_if_needed raw_ty elem) with Box(v) -> v)
    let _ = build_new_clist dataptr elements the_state.b in
      (Box(objptr), the_state)

and add_terminal the_state instr =
    (match L.block_terminator (L.insertion_block the_state.b) with
         Some _ -> ()   (* do nothing if terminator is there *)
    | None -> ignore (instr the_state.b)); the_state

and rip_from_inner_state old inner =
  change_state old (S_list([S_names(inner.namespace);S_optimfuncs(inner.optim_funcs)])) (* grab na

and stmt the_state s =   (* namespace comes first bc never gets modified unless descending so it w
    let (namespace,the_function) = (the_state.namespace, the_state.func) in
    match s with
    | SBlock s -> List.fold_left stmt the_state s
    | SExpr e ->  let (_,the_state) = expr the_state e in the_state
    | SAsn (bind_list, e) -> (*L.dump_module the_module;*)
      let (_, tp_rhs) = e in
      let (e', the_state) = expr the_state e in
      let get_bind = function
        | SLVar (Bind (name, explicit_type)) -> (Bind(name, tp_rhs), explicit_type)
        | _ -> raise (Failure "CodegenError: this kind of assignment has not been implemented.")

      in let binds = List.map get_bind bind_list in
```

```
  let addrs = List.map (fun (bind, explicit_type) -> ((lookup namespace bind), explicit_type))
  let do_store lhs rhs the_state =
    let (lbind, tp_lhs) = lhs in
    let the_state = (match rhs with
      | Raw(v) -> (match lbind with
         | RawAddr(addr) -> ignore(L.build_store v addr the_state.b); the_state
         | BoxAddr(_,_) -> tstp "ERROR, assinging Raw to BoxAddr"; the_state) (** shd crash in
      | Box(v) -> (match lbind with
         | RawAddr(_) -> tstp "ERROR, assigning Box to RawAddr"; the_state
         | BoxAddr(addr, _) ->
            let the_state = (match tp_lhs with
              | Dyn -> the_state
              | _ ->
                (* exception handling: invalid assign *)
                let bad_asn_bb = L.append_block context "bad_asn" the_state.func in
                let bad_asn_bd = L.builder_at_end context bad_asn_bb in

                let proceed_bb = L.append_block context "proceed" the_state.func in

                (* check for asn exception *)
                let ctp_lhs = ctype_of_ASTtype tp_lhs in (* type of lefthand expression *)
                let ctp_rhs = build_gettype_cobj v the_state.b in
                let _ = (match ctp_lhs with
                  | None -> ()
                  | Some ctp_lhs ->

                    let lhs_as_int = L.build_ptrtoint ctp_lhs int_t "lhs_as_int" the_state.b i
                    let rhs_as_int = L.build_ptrtoint ctp_rhs int_t "rhs_ rtp_as_int" the_stat
                    let diff = L.build_sub lhs_as_int rhs_as_int "diff" the_state.b in
                    let invalid_asn = L.build_icmp L.Icmp.Ne diff (L.const_int int_t 0) "inval
                      ignore(L.build_cond_br invalid_asn bad_asn_bb proceed_bb the_state.b);)
                in

                (* print message and exit *)
                let err_message =
                  let info = "invalid assignment to object of type " ^ (Utilities.type_to_stri
                    L.build_global_string info "error message" bad_asn_bd in
                let str_format_str1 = L.build_global_stringptr  "%s\n" "fmt" bad_asn_bd in
                  ignore(L.build_call printf_func [| str_format_str1; err_message |] "printf"
                  ignore(L.build_call exit_func [| (L.const_int int_t 1) |] "exit" bad_asn_bd)

                (* return to normal control flow *)
                let the_state = change_state the_state (S_b(L.builder_at_end context proceed_b
                  ignore(L.build_br proceed_bb bad_asn_bd); the_state)
        in ignore(L.build_store v addr the_state.b); the_state))
    in the_state
  in
 let (the_state, _) = List.fold_left (fun (the_state, rhs) lhs ->
    let the_state = do_store lhs rhs the_state in (the_state, e')) (the_state, e') addrs in
  the_state
| SNop -> the_state
| SPrint e ->
     let (_,t) = e in
```

```
      let (res,the_state) = expr the_state e in
      (match res with
          |Raw(v) -> tstp "raw"; (match t with
              | Int -> ignore(L.build_call printf_func [| int_format_str ; v |] "printf" the_s
              | Float -> ignore(L.build_call printf_func [| float_format_str ; v |] "printf" t
              | Bool -> ignore(L.build_call printf_func [| int_format_str ; v |] "printf" the_
              | String -> ignore(L.build_call printf_func [| string_format_str ; v |] "printf"
          )
          |Box(v) ->tstp "box";
              (*let cobjptr = L.build_alloca cobj_t "tmp" b in
              ignore(L.build_store v cobjptr b);*)
              (*ignore(L.build_call printf_func [| int_format_str ; (build_getdata_cobj int_t
              let fn_p = build_getctypefn_cobj ctype_print_idx v the_state.b in
              ignore(L.build_call fn_p [|v|] "print_cob" the_state.b); the_state
      )


|SIf (predicate, then_stmt, else_stmt) ->
  let (e,the_state) = expr the_state predicate in
  let bool_val = (match e with
    |Raw(v) -> v
    |Box(v) -> build_getdata_cobj bool_t v the_state.b
  ) in
  let merge_bb = L.append_block context "merge" the_function in
  let build_br_merge = L.build_br merge_bb in
  let then_bb = L.append_block context "then" the_function in
  let then_state = change_state the_state (S_b(L.builder_at_end context then_bb)) in
  let then_state = add_terminal (stmt then_state then_stmt) build_br_merge in
  let the_state = rip_from_inner_state the_state then_state in
  let else_bb = L.append_block context "else" the_function in
  let else_state = change_state the_state (S_b(L.builder_at_end context else_bb)) in
  let else_state = add_terminal (stmt else_state else_stmt) build_br_merge in  (* same deal as
  let the_state = rip_from_inner_state the_state else_state in
  ignore(L.build_cond_br bool_val then_bb else_bb the_state.b);
  let the_state = change_state the_state (S_b(L.builder_at_end context merge_bb)) in
  the_state

| SWhile (predicate, body) ->
  let pred_bb = L.append_block context "while" the_function in
  ignore(L.build_br pred_bb the_state.b);
  let body_bb = L.append_block context "while_body" the_function in
  let body_state = change_state the_state (S_b(L.builder_at_end context body_bb)) in
  let body_state = add_terminal (stmt body_state body) (L.build_br pred_bb) in
  let the_state = rip_from_inner_state the_state body_state in
  let pred_builder = L.builder_at_end context pred_bb in
  (* eval the boolean predicate *)
  let pred_state = change_state the_state (S_b(L.builder_at_end context pred_bb)) in
  let (e,pred_state) = expr pred_state predicate in
  let the_state = rip_from_inner_state the_state pred_state in
  let bool_val = (match e with
    |Raw(v) -> v
    |Box(v) -> build_getdata_cobj bool_t v pred_state.b
  ) in
  let merge_bb = L.append_block context "merge" the_function in
```

```
    ignore(L.build_cond_br bool_val body_bb merge_bb pred_state.b);
    let merge_state = change_state the_state (S_b(L.builder_at_end context merge_bb)) in
    merge_state
  | SFor(var, lst, body) ->
     (* initialize list index variable and list length *)
     let (Box(objptr), the_state) = expr the_state lst in  (* TODO update box if needed *)
     let listptr = build_getlist_cobj objptr the_state.b in
     let nptr = L.build_alloca int_t "nptr" the_state.b in
       ignore(L.build_store (L.const_int int_t (0)) nptr the_state.b);
     let n = L.build_load nptr "n" the_state.b in
     let ln = build_getlen_clist listptr the_state.b in

     (* iter block *)
     let iter_bb = L.append_block context "iter" the_function in
       ignore(L.build_br iter_bb the_state.b);

     let iter_builder = L.builder_at_end context iter_bb in
     let n = L.build_load nptr "n" iter_builder in
     let nnext = L.build_add n (L.const_int int_t 1) "nnext" iter_builder in
       ignore(L.build_store nnext nptr iter_builder);

     let iter_complete = (L.build_icmp L.Icmp.Sge) n ln "iter_complete" iter_builder in (* true

     (* body of for loop *)
     let body_bb = L.append_block context "for_body" the_function in
     let body_builder = L.builder_at_end context body_bb in

     let elmptr = build_idx listptr n "binop_result" body_builder in
       let BoxAddr(var_addr,_) = (lookup namespace var) in (*assignment so ok to throw away the
       ignore(L.build_store elmptr var_addr body_builder);
     let the_state = change_state the_state (S_b(body_builder)) in
       add_terminal (stmt the_state body) (L.build_br iter_bb);

     let merge_bb = L.append_block context "merge" the_function in
       ignore(L.build_cond_br iter_complete merge_bb body_bb iter_builder);
     let the_state = change_state the_state (S_b(L.builder_at_end context merge_bb)) in
       the_state
| SReturn e -> let (_,ty) = e in
    let (res,the_state) = expr the_state e in
    (match the_state.generic_func with  (* if generic must ret cobject *)
      |false -> (match ty with
        | Null -> L.build_ret (build_new_cobj_init int_t (L.const_int int_t 0) the_state.b) the_
        | _ -> L.build_ret (match res with
            | Raw(v) -> (match the_state.ret_typ with
                |Dyn ->(match (build_temp_box v ty the_state.b) with Box(v) -> v)
                |_ -> v
            )
            |Box(v) -> v
        ) the_state.b
        )
      |true -> L.build_ret (match res with
        |Box(v) -> v
        |Raw(v) -> (match (build_temp_box v ty the_state.b) with Box(v) -> v)
        ) the_state.b
```

```
        );the_state
    | SFunc sfdecl -> the_state (*
        tstp ("CREATING GENERIC FN: " ^ sfdecl.sfname); (* create the generic function object, local
        (* outer scope work: point binding to new cfuncobj *)
        let fname = sfdecl.sfname in
        let the_function = L.define_function fname userdef_fn_t the_module in
        (* manually design the fn object w proper data & type ptrs and put in bind *)
        let _ =
          let (fn_obj,datafieldptr,ctypefieldptr) = build_new_cobj_empty the_state.b in
          let dfp_as_fp = L.build_bitcast datafieldptr (L.pointer_type userdef_fn_pt) "dfp_as_fp" th
          ignore(L.build_store the_function dfp_as_fp the_state.b);  (* store fnptr *)
          ignore(L.build_store ctype_func ctypefieldptr the_state.b);  (* store ctype ptr *)
          (* store new object in appropriate binding *)
          let BoxAddr(boxaddr,_) = (lookup namespace (Bind(fname,Dyn))) in (*ok to throw away need_u
          ignore(L.build_store fn_obj boxaddr the_state.b)
        in
        let fn_b = L.builder_at_end context (L.entry_block the_function) in
        (* update the namespace in this big section *)
        let local_names = names_of_bindlist sfdecl.slocals
        and formal_names = names_of_bindlist sfdecl.sformals in
        let argc = List.length formal_names
        and argv = Array.get (L.params the_function) 0 in (* argv is first/only arg *)
        let cobj_p_arr_pt = L.pointer_type (L.array_type cobj_pt argc) in
        let formals_arr_p = L.build_bitcast argv cobj_p_arr_pt "formals_arr_p" fn_b in
        (* now formals_arr_p is a ptr to an array of cobj_ps which are the formals *)
        let formals_arr = L.build_load formals_arr_p "formals_arr" fn_b in
        (* Very important! the actual extraction of the formals from formals_arr *)
        let formal_vals = List.map (fun idx -> L.build_extractvalue formals_arr idx ("arg"^(string_o
        (* now formal_vals is a list of co_ps *)

        let names_to_dynlist names =
          List.rev (List.fold_left (fun acc n -> (Bind(n,Dyn))::acc) [] names)
        in

(* all formals should be dyns! *)
        (*let fn_namespace = build_binding_list (Some(fn_b)) (names_to_dynlist formal_names) in*)
        let add_formal nspace name cobj_p =  (* alloc a formal *)
            L.set_value_name name cobj_p;  (* cosmetic *)
            let alloca = L.build_alloca cobj_pt name fn_b in
            ignore(L.build_store cobj_p alloca fn_b);
            BindMap.add (Bind(name,Dyn)) (BoxAddr(alloca,false)) nspace
        in
        let fn_namespace = build_binding_list (Some(fn_b)) sfdecl.slocals in
        let fn_namespace = List.fold_left2 add_formal fn_namespace formal_names formal_vals in
        let int_format_str = L.build_global_stringptr "%d\n" "fmt" fn_b
        and float_format_str = L.build_global_stringptr "%f\n" "fmt" fn_b
        and str_format_str = L.build_global_stringptr  "%s\n" "fmt" fn_b in
        (* build function body by calling stmt! *)
        let build_return bld = L.build_ret (build_new_cobj_init int_t (L.const_int int_t 0) bld) bld
        let fn_state = change_state the_state (S_list([S_names(fn_namespace);S_func(the_function);S_
        let fn_state = add_terminal (stmt fn_state sfdecl.sbody) build_return in
        let the_state = change_state the_state (S_optimfuncs(fn_state.optim_funcs)) in (* grab optim
        the_state  (* SFunc() returns the original builder *)
        *)
```

```
  | STransform (name,from_ty,to_ty) ->
    tstp ("Transforming "^name^": "^(string_of_typ from_ty)^" -> "^(string_of_typ to_ty));
    (match (from_ty,to_ty) with
     |(x,y) when x=y -> the_state
     |(FuncType,Dyn)|(Dyn,FuncType) -> the_state
     |(Dyn,raw_ty) when raw_ty=Int || raw_ty=Float || raw_ty=Bool ->
       (* get addresses for raw and boxed versions *)
       let unchecked_boxaddr = lookup namespace (Bind(name,Dyn)) in
       let the_state = rebox_if_needed unchecked_boxaddr name the_state in
       let BoxAddr(box_addr,_) = unchecked_boxaddr
       and RawAddr(raw_addr) = lookup namespace (Bind(name,raw_ty)) in
       (* gep for direct pointers to the type and data fields of box *)
       let cobj_addr = L.build_load box_addr "cobjptr" the_state.b in
       let dataptr_addr = L.build_struct_gep cobj_addr cobj_data_idx "dat_p_p" the_state.b in
       let dataptr = L.build_load dataptr_addr "dat_p" the_state.b in
       let typed_dataptr = L.build_bitcast dataptr (ltyp_of_typ raw_ty) "typd_dat_p" the_state.b i
       let data = L.build_load typed_dataptr "dat" the_state.b in
       ignore(L.build_store data raw_addr the_state.b);
       the_state

     | (raw_ty,Dyn) when raw_ty=Int || raw_ty=Float || raw_ty=Bool ->
       (* get addresses for raw and boxed versions *)
       let BoxAddr(box_addr,_) = lookup namespace (Bind(name,Dyn)) (* no need to check needs_updat
       and RawAddr(raw_addr) = lookup namespace (Bind(name,raw_ty)) in
       (* gep for direct pointers to the type and data fields of box *)
       let cobj_addr = L.build_load box_addr "cobjptr" the_state.b in
       let raw_addr = L.build_bitcast raw_addr char_pt "raw" the_state.b in
       let dataptr_addr = L.build_struct_gep cobj_addr cobj_data_idx "dat_p_p" the_state.b in
       let typeptr_addr = L.build_struct_gep cobj_addr cobj_type_idx "ty_p_p" the_state.b in
       let typeptr_addr = L.build_bitcast typeptr_addr (L.pointer_type ctype_pt) "ty" the_state.b
       (* store raw_addr in the box's dataptr field and update the typeptr *)
       ignore(L.build_store raw_addr dataptr_addr the_state.b);
       ignore(L.build_store (ctype_of_typ raw_ty) typeptr_addr the_state.b);
       let the_state = change_state the_state (S_needs_reboxing(name,true)) in
       the_state
    )
in

let final_state = stmt init_state (SBlock(fst prgm)) in

ignore(L.build_ret (L.const_int int_t 0) final_state.b);
  (* prints module *)

the_module  (* return the resulting llvm module with all code!! *)
```

codegenutils.ml

8.2 src Files

ast.ml (Jacob Austin)

```
1 The Abstract Syntax Tree (AST) for the Coral Programming Language *)
2
```

```
 This AST supports many of the features of modern Python, with the exception of
ceptions, and some built-in data-structures supported by Python. *)

pe operator = Add | Sub | Mul | Div | Exp | Eq | Neq | Less | Leq | Greater | Geq | And | Or | ListA

pe uop = Neg | Not

pe literal =
| IntLit of int
| BoolLit of bool
| FloatLit of float
| StringLit of string

pe typ = Int | Float | Bool | String | Dyn | IntArr | FloatArr | BoolArr | StringArr | FuncType | Nu

pe bind = Bind of string * typ

pe expr =
| Binop of expr * operator * expr
| Lit of literal
| Var of bind
| Unop of uop * expr
| Call of expr * expr list
| Method of expr * string * expr list
| Field of expr * string
| List of expr list
| ListAccess of expr * expr (* expr, entry *)
| ListSlice of expr * expr * expr (* expr, left, right *)

pe stmt =
| Func of bind * bind list * stmt
| Block of stmt list
| Expr of expr
| If of expr * stmt * stmt
| For of bind * expr * stmt
| Range of bind * expr * stmt
| While of expr * stmt
| Return of expr
| Class of string * stmt
| Asn of expr list * expr
| Type of expr
| Print of expr
| Import of string
| Nop


t rec string_of_op = function
| Add -> "+"
| Sub -> "-"
| Mul -> "*"
| Div -> "/"
| Exp -> "**"
| Eq -> "=="
| Neq -> "!="
```

```
| Less -> "<"
| Leq -> "<="
| Greater -> ">"
| Geq -> ">="
| And -> "and"
| Or -> "or"
| ListAccess -> "at index"

t rec string_of_uop = function
| Neg -> "-"
| Not -> "not"

t rec string_of_lit = function
| IntLit(i) -> string_of_int i
| BoolLit(b) -> string_of_bool b
| FloatLit(f) -> string_of_float f
| StringLit(s) -> s

t rec string_of_typ = function
| Int -> "int"
| Float -> "float"
| Bool -> "bool"
| String -> "str"
| Dyn -> "dyn"
| IntArr -> "int[]"
| FloatArr -> "float[]"
| BoolArr -> "bool[]"
| StringArr -> "str[]"
| FuncType -> "func"
| Null -> "null"

t rec string_of_bind = function
| Bind(s, t) -> s ^ ": " ^ string_of_typ t

t rec string_of_expr = function
| Binop(e1, o, e2) -> string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
| Lit(l) -> string_of_lit l
| Var(b) -> string_of_bind b
| Unop(o, e) -> string_of_uop o ^ string_of_expr e
| Call(e, el) -> string_of_expr e ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
| Method(obj, m, el) -> string_of_expr obj ^ "." ^ m ^ "(" ^ String.concat ", " (List.map string_of_
| Field(obj, s) -> string_of_expr obj ^ "." ^ s
| List(el) -> String.concat ", " (List.map string_of_expr el)
| ListAccess(e1, e2) -> string_of_expr e1 ^ "[" ^ string_of_expr e2 ^ "]"
| ListSlice(e1, e2, e3) -> string_of_expr e1 ^ "[" ^ string_of_expr e2 ^ ":" ^ string_of_expr e3 ^ "

t rec string_of_stmt = function
| Func(b, bl, s) -> "def " ^ string_of_bind b ^ "(" ^ String.concat ", " (List.map string_of_bind bl
| Block(sl) -> String.concat "\n" (List.map string_of_stmt sl) ^ "\n"
| Expr(e) -> string_of_expr e
| If(e, s1, s2) ->  "if " ^ string_of_expr e ^ ":\n" ^ string_of_stmt s1 ^ "else:\n" ^ string_of_stm
| For(b, e, s) -> "for " ^ string_of_bind b ^ " in " ^ string_of_expr e ^ ":\n" ^ string_of_stmt s
| Range(b, e, s) -> "for " ^ string_of_bind b ^ " in range (" ^ string_of_expr e ^ "):\n" ^ string_o
| While(e, s) -> "while " ^ string_of_expr e ^ ":\n" ^ string_of_stmt s
```

```
111 | Return(e) -> "return " ^ string_of_expr e ^ "\n"
112 | Class(str, s) -> "class " ^ str ^ ":\n" ^ string_of_stmt s
113 | Asn(el, e) -> String.concat ", " (List.map string_of_expr el) ^ " = "  ^ string_of_expr e
114 | Type(e) -> string_of_expr e
115 | Print(e) -> string_of_expr e
116 | Import(e) -> "import " ^ e
117 | Nop -> ""
118
119 d string_of_program l = String.concat "" (List.map string_of_stmt l) ^ "\n\n"
```

---

codegen.ml - Matthew Bowers and Sanford Miller

```
1      (* Code generation: translate takes a semantically checked AST and
2 VM tutorial: Make sure to read the OCaml version of the tutorial
3 tp://llvm.org/docs/tutorial/index.html
4 tailed documentation on the OCaml LLVM library:
5 tp://llvm.moe/
6 tp://llvm.moe/ocaml/
7
8
9 dule L = Llvm
10 module A = Ast*)
11 en Ast
12 en Sast
13
14 dule StringMap = Map.Make(String)
15 t pt some_lltype = Printf.eprintf "pt: %s%s\n" "---->" (L.string_of_lltype some_lltype)
16 t pv some_llvalue = Printf.eprintf "pv: %s%s\n" "---->" (L.string_of_llvalue some_llvalue)
17 t tst() = Printf.eprintf "!!!!!!!!!!";()
18 t tstp str = Printf.eprintf "%s\n" str;()
19
20
21
22 type state = (L.llvalue StringMap.t) * L.llvalue * L.llbuilder*)
23 pe state = {
24   namespace: L.llvalue StringMap.t;
25   func: L.llvalue;
26   b: L.llbuilder
27
28
29 t seq len =
30 let rec aux len acc =
31   if len < 0 then acc else aux (len - 1) (len::acc)
32 in aux (len - 1) []
33
34 t initial_list_size = 10
35 t list_growth_factor = 2
36
37 pe oprt =
38 | Oprt of
39     string
40   * ((L.llvalue -> L.llvalue -> string -> L.llbuilder -> L.llvalue) * L.lltype) option
41   * ((L.llvalue -> L.llvalue -> string -> L.llbuilder -> L.llvalue) * L.lltype) option
```

```
  * ((L.llvalue -> L.llvalue -> string -> L.llbuilder -> L.llvalue) * L.lltype) option
  * ((L.llvalue -> L.llvalue -> string -> L.llbuilder -> L.llvalue) * L.lltype) option
  * ((L.llvalue -> L.llvalue -> string -> L.llbuilder -> L.llvalue) * L.lltype) option
| Uoprt of
    string
  * ((L.llvalue -> string -> L.llbuilder -> L.llvalue) * L.lltype) option
  * ((L.llvalue -> string -> L.llbuilder -> L.llvalue) * L.lltype) option
  * ((L.llvalue -> string -> L.llbuilder -> L.llvalue) * L.lltype) option
  * ((L.llvalue -> string -> L.llbuilder -> L.llvalue) * L.lltype) option
  * ((L.llvalue -> string -> L.llbuilder -> L.llvalue) * L.lltype) option
| Loprt of
    string
  * ((L.llvalue -> L.llvalue -> string -> L.llbuilder -> L.llvalue) * L.lltype) option
  * ((L.llvalue -> L.llvalue -> string -> L.llbuilder -> L.llvalue) * L.lltype) option
  * ((L.llvalue -> L.llvalue -> string -> L.llbuilder -> L.llvalue) * L.lltype) option
  * ((L.llvalue -> L.llvalue -> string -> L.llbuilder -> L.llvalue) * L.lltype) option
  * ((L.llvalue -> L.llvalue -> string -> L.llbuilder -> L.llvalue) * L.lltype) option

pe built_oprt =
| BOprt of ((L.llvalue * L.llbuilder) * ((L.llvalue -> L.llvalue -> string -> L.llbuilder -> L.llval
| BUoprt of ((L.llvalue * L.llbuilder) * ((L.llvalue -> string -> L.llbuilder -> L.llvalue) * L.llty
| BLoprt of ((L.llvalue * L.llbuilder) * ((L.llvalue -> L.llvalue -> string -> L.llbuilder -> L.llva

 translate : Sast.program -> Llvm.module *)
t translate prgm =   (* note this whole thing only takes two things: globals= list of (typ,name) (bi
let context = L.global_context () in  (* context keeps track of global vars and stuff i think *)

(* Create the LLVM compilation module boolo which
   we will generate code *)
let the_module = L.create_module context "Coral" in  (* the_module will hold all functs + global var
let pm() = L.dump_module the_module in

(* Get types from the context *)
let int_t      = L.i32_type    context
and float_t    = L.double_type context
and bool_t     = L.i1_type     context
and char_t     = L.i8_type     context in

(* ptr types *)
let int_pt = L.pointer_type int_t
and float_pt = L.pointer_type float_t
and bool_pt = L.pointer_type bool_t
and char_pt = L.pointer_type char_t in

(* define cobj and ctype structs *)
let cobj_t = L.named_struct_type context "CObj" in (*define a named struct*)
let cobj_pt = L.pointer_type cobj_t in
let cobj_ppt = L.pointer_type cobj_pt in

(* all generic userdef functions follow this type *)
let userdef_fn_t = L.function_type cobj_pt [| cobj_ppt |] in   (* takes an argv *)
let userdef_fn_pt = L.pointer_type userdef_fn_t in

(* define cobj_list and ctype_list structs *)
```

```
96 let clist_t = L.named_struct_type context "CList" in (*define a named struct*)
97 let clist_pt = L.pointer_type clist_t in
98
99 (* define ctype and ctype structs *)
100 let ctype_t = L.named_struct_type context "CType" in (*define a named struct*)
101 let ctype_pt = L.pointer_type ctype_t in
102
103 (* cobj idxs *)
104 let cobj_data_idx = 0
105 and cobj_type_idx = 1 in
106
107 (* clist idxs *)
108 let clist_data_idx = 0
109 and clist_len_idx = 1
110 and clist_cap_idx = 2
111 in
112
113 (* ctype idx *)
114 let ctype_add_idx = 0
115 and ctype_sub_idx = 1
116 and ctype_mul_idx = 2
117 and ctype_div_idx = 3
118 and ctype_exp_idx = 4
119 and ctype_eq_idx = 5
120 and ctype_neq_idx = 6
121 and ctype_lesser_idx = 7
122 and ctype_leq_idx = 8
123 and ctype_greater_idx = 9
124 and ctype_geq_idx = 10
125 and ctype_and_idx = 11
126 and ctype_or_idx = 12
127 and ctype_neg_idx = 13
128 and ctype_not_idx = 14
129 and ctype_idx_idx = 15
130 and ctype_call_idx = 16
131 and num_ctype_idxs = 17 in (**must update when adding idxs! (tho not used anywhere yet)**)
132
133 (* type sigs for fns in ctype *)
134 let ctype_add_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
135 and ctype_sub_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
136 and ctype_mul_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
137 and ctype_div_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
138 and ctype_exp_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
139 and ctype_eq_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
140 and ctype_neq_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
141 and ctype_lesser_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
142 and ctype_leq_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
143 and ctype_greater_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
144 and ctype_geq_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
145 and ctype_and_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
146 and ctype_or_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
147 and ctype_neg_t = L.function_type cobj_pt [| cobj_pt |]
148 and ctype_not_t = L.function_type cobj_pt [| cobj_pt |]
149 and ctype_idx_t = L.function_type cobj_pt [| cobj_pt; cobj_pt |]
```

```
150 and ctype_call_t = L.function_type cobj_pt [| cobj_pt ; cobj_ppt |] in
151
152 (* type sigs for ptrs to fns in ctype *)
153 let ctype_add_pt = L.pointer_type ctype_add_t
154 and ctype_sub_pt = L.pointer_type ctype_sub_t
155 and ctype_mul_pt = L.pointer_type ctype_mul_t
156 and ctype_div_pt = L.pointer_type ctype_div_t
157 and ctype_exp_pt = L.pointer_type ctype_exp_t
158 and ctype_eq_pt = L.pointer_type ctype_eq_t
159 and ctype_neq_pt = L.pointer_type ctype_neq_t
160 and ctype_lesser_pt = L.pointer_type ctype_lesser_t
161 and ctype_leq_pt = L.pointer_type ctype_leq_t
162 and ctype_greater_pt = L.pointer_type ctype_greater_t
163 and ctype_geq_pt = L.pointer_type ctype_geq_t
164 and ctype_and_pt = L.pointer_type ctype_and_t
165 and ctype_or_pt = L.pointer_type ctype_or_t
166 and ctype_neg_pt = L.pointer_type ctype_neg_t
167 and ctype_not_pt = L.pointer_type ctype_not_t
168 and ctype_idx_pt = L.pointer_type ctype_idx_t
169 and ctype_call_pt = L.pointer_type ctype_call_t in
170
171 let ctype_t = L.named_struct_type context "CType" in (*define a named struct*)
172 let ctype_pt = L.pointer_type ctype_t in
173
174 (* set ctype and cobj struct bodies *)
175 ignore(L.struct_set_body cobj_t [| char_pt; ctype_pt |] false);
176 ignore(L.struct_set_body clist_t [| char_pt; int_t; int_t |] false);
177 ignore(L.struct_set_body ctype_t [|
178         ctype_add_pt;
179         ctype_sub_pt;
180         ctype_mul_pt;
181   ctype_div_pt;
182   ctype_exp_pt;
183   ctype_eq_pt;
184   ctype_neq_pt;
185   ctype_lesser_pt;
186         ctype_leq_pt;
187         ctype_greater_pt;
188         ctype_geq_pt;
189         ctype_and_pt;
190         ctype_or_pt;
191         ctype_neg_pt;
192   ctype_not_pt;
193         ctype_idx_pt;
194         ctype_call_pt |] false);
195
196  let get_t = function
197    | "int" -> int_t
198    | "float" -> float_t
199    | "bool" -> bool_t
200    | "char" -> char_t
201    | "list" -> clist_t
202  in
203
```

```
 let build_ctype_fn fname ftype = (* ftype = "ctype_add_t" etc *)
   let the_function = L.define_function fname ftype the_module in
   let builder = L.builder_at_end context (L.entry_block the_function) in
   (the_function, builder)
 in

(* here's how you go from a cobj to the data value: *)
let build_getdata_cobj data_type cobj_p b =  (* data_type = int_t etc *)
  (*let x1 = L.build_load (lookup_global_binding "a") "x1" b in*)
  let x2 = L.build_struct_gep cobj_p cobj_data_idx "x2" b in
  let x3 = L.build_load x2 "x3" b in
  let x4 = L.build_bitcast x3 (L.pointer_type data_type) "x4" b in
  let data = L.build_load x4 "data" b in
  data
in

let build_getlist_cobj cobj_p b =
  let gep_addr = L.build_struct_gep cobj_p cobj_data_idx "__gep_addr" b in
  let objptr = L.build_load gep_addr "__objptr" b in
  L.build_bitcast objptr clist_pt "__clistptr" b
in

let build_getlen_clist clist_p b =
  let gep_addr = L.build_struct_gep clist_p clist_len_idx "__gep_addr" b in
  let gep_addr_as_intptr = L.build_bitcast gep_addr int_pt "__gep_addr_as_intptr" b in
  let length = L.build_load gep_addr_as_intptr "__length" b in
  length
in

let build_getcap_clist clist_p b =

  let gep_addr = L.build_struct_gep clist_p clist_cap_idx "__gep_addr" b in
  let gep_addr_as_intptr = L.build_bitcast gep_addr int_pt "__gep_addr_as_intptr" b in
  let capacity = L.build_load gep_addr_as_intptr "__capacity" b in
  capacity
in

let build_idx self_p other_p name b =

  (* TODO: throw error if array bounds exceeded *)
  let capacity = build_getcap_clist self_p b in
  let inbounds = L.build_icmp L.Icmp.Slt other_p capacity "__inbounds" b in (* other_p is index bein

  (* get elememnt *)
  let gep_addr = L.build_struct_gep self_p clist_data_idx "__gep_addr" b in
  let gep_addr_as_cobjptrptrptr = L.build_bitcast gep_addr (L.pointer_type (L.pointer_type cobj_pt))
  let gep_addr_as_cobjptrptr = L.build_load gep_addr_as_cobjptrptrptr "__gep_addr_as_cobjptrptr" b i
  let gep_addr_as_cobjptrptr = L.build_gep gep_addr_as_cobjptrptr [| other_p |] "__gep_addr_as_cobjp
  let cobjptr = L.build_load gep_addr_as_cobjptrptr "__cobjptr" b in
  cobjptr
in

let built_ops =
        let typs = ["int"; "float"; "bool"; "char"; "list"] in
```

```
258
259         let ops = [
260           Oprt("add", Some((L.build_add), int_t), Some((L.build_fadd), float_t), None, None, None);
261       Oprt("sub", Some((L.build_sub), int_t), Some((L.build_fsub), float_t), None, None, None);
262       Oprt("mul", Some((L.build_mul), int_t), Some((L.build_fmul), float_t), None, None, None);
263       Oprt("div", Some((L.build_sdiv), int_t), Some((L.build_fdiv), float_t), None, None, None);
264       Oprt("exp", None, None, None, None, None);
265       Oprt("eq", Some((L.build_icmp L.Icmp.Eq), bool_t), Some((L.build_fcmp L.Fcmp.Ueq), bool_t), Som
266       Oprt("neq", Some((L.build_icmp L.Icmp.Ne), bool_t), Some((L.build_fcmp L.Fcmp.Une), bool_t), So
267       Oprt("lesser", Some((L.build_icmp L.Icmp.Slt), bool_t), Some((L.build_fcmp L.Fcmp.Ult), bool_t)
268       Oprt("leq", Some((L.build_icmp L.Icmp.Sle), bool_t), Some((L.build_fcmp L.Fcmp.Ule), bool_t), S
269       Oprt("greater", Some((L.build_icmp L.Icmp.Sgt), bool_t), Some((L.build_fcmp L.Fcmp.Ugt), bool_t
270       Oprt("geq", Some((L.build_icmp L.Icmp.Sge), bool_t), Some((L.build_fcmp L.Fcmp.Uge), bool_t), S
271       Oprt("and", Some((L.build_and), int_t), None, Some((L.build_and), bool_t), Some((L.build_add),
272       Oprt("or", Some((L.build_or), int_t), None, Some((L.build_or), bool_t), Some((L.build_or), char
273       Uoprt("neg", Some((L.build_neg), int_t), Some((L.build_fneg), float_t), Some((L.build_neg), boo
274       Uoprt("not", Some((L.build_not), int_t), None, Some((L.build_not), bool_t), Some((L.build_not),
275       Loprt("idx", None, None, None, None, Some((build_idx), int_t))
276       ] in
277
278           List.map (fun t -> let bops = List.map (function
279              | Oprt(o, i, f, b, c, l) ->
280         let tfn = match t with
281           | "int" -> i
282           | "float" -> f
283           | "bool" -> b
284           | "char" -> c
285           | "list" -> l
286        in
287        let bop = match tfn with
288                | Some tfn ->
289                  let (fn, bd) = build_ctype_fn (t ^ "_" ^ o) ((function
290                    | "add" -> ctype_add_t
291                  | "sub" -> ctype_sub_t
292                  | "mul" -> ctype_mul_t
293                  | "div" -> ctype_div_t
294                  | "exp" -> ctype_exp_t
295                  | "eq" -> ctype_eq_t
296                  | "neq" -> ctype_neq_t
297                  | "lesser" -> ctype_lesser_t
298                  | "leq" -> ctype_leq_t
299                  | "greater" -> ctype_greater_t
300                  | "geq" -> ctype_geq_t
301                  | "and" -> ctype_and_t
302                  | "or" -> ctype_or_t) o)
303                        in BOprt(Some(((fn, bd), tfn)))
304                | None -> BOprt(None)
305         in bop
306       | Uoprt(o, i, f, b, c, l) ->
307         let tfn = match t with
308           | "int" -> i
309           | "float" -> f
310           | "bool" -> b
311           | "char" -> c
```

```
312           | "list" -> l
313         in
314       let bop = match tfn with
315               | Some tfn ->
316                 let (fn, bd) = build_ctype_fn (t ^ "_" ^ o) ((function
317               | "neg" -> ctype_neg_t
318               | "not" -> ctype_not_t) o)
319                       in BUoprt(Some(((fn, bd), tfn)))
320               | None -> BUoprt(None)
321       in bop
322       | Loprt(o, i, f, b, c, l) ->
323         let tfn = match t with
324           | "int" -> i
325           | "float" -> f
326           | "bool" -> b
327           | "char" -> c
328           | "list" -> l
329       in
330       let bop = match tfn with
331               | Some tfn ->
332                 let (fn, bd) = build_ctype_fn (t ^ "_" ^ o) ((function
333               | "idx" -> ctype_idx_t) o)
334                       in BLoprt(Some(((fn, bd), tfn)))
335               | None -> BLoprt(None)
336         in bop) ops
337       in (t, bops)) typs
338     in
339
340 (* Functions! *)
341 let (func_call_fn,func_call_b) = build_ctype_fn "func_call" ctype_call_t in
342
343 let ctype_func = L.define_global "ctype_func" (L.const_named_struct ctype_t [|
344       L.const_pointer_null ctype_add_pt; (* ctype_add_pt *)
345       L.const_pointer_null ctype_sub_pt; (* ctype_sub_pt *)
346       L.const_pointer_null ctype_mul_pt; (* ctype_mul_pt *)
347       L.const_pointer_null ctype_div_pt; (* ctype_div_pt *)
348       L.const_pointer_null ctype_exp_pt; (* ctype_exp_pt *)
349       L.const_pointer_null ctype_eq_pt; (* ctype_eq_pt *)
350       L.const_pointer_null ctype_neq_pt; (* ctype_neq_pt *)
351       L.const_pointer_null ctype_lesser_pt; (* ctype_lesser_pt *)
352       L.const_pointer_null ctype_leq_pt; (* ctype_leq_pt *)
353       L.const_pointer_null ctype_greater_pt; (* ctype_greater_pt *)
354       L.const_pointer_null ctype_geq_pt; (* ctype_geq_pt *)
355       L.const_pointer_null ctype_and_pt; (* ctype_and_pt *)
356       L.const_pointer_null ctype_or_pt; (* ctype_or_pt *)
357       L.const_pointer_null ctype_neg_pt; (* ctype_neg_pt *)
358       L.const_pointer_null ctype_not_pt; (* ctype_not_pt *)
359       L.const_pointer_null ctype_idx_pt; (* ctype_not_pt *)
360       func_call_fn; (* ctype_call_pt *)
361   |]) the_module in
362
363 let build_fnptr_of_cfo cobj_p b =
364   let x2 = L.build_struct_gep cobj_p cobj_data_idx "x2" b in
365   let x3 = L.build_load x2 "x3" b in
```

```
366   let fnptr = L.build_bitcast x3 userdef_fn_pt "fnptr" b in
367   fnptr
368 in
369
370 (* define the default CTypes *)
371 let [ctype_int; ctype_float; ctype_bool; ctype_char; ctype_list] =
372         List.map (fun (t, bops) -> L.define_global ("ctype_" ^ t) (L.const_named_struct ctype_t (Arr
373           | BOprt(o) -> (match o with
374             | Some(((fn, bd), tfn)) -> fn
375             | None -> L.const_pointer_null ctype_add_pt)
376           | BUoprt(o) -> (match o with
377             | Some(((fn, bd), tfn)) -> fn
378             | None -> L.const_pointer_null ctype_neg_pt)
379           | BLoprt(o) -> (match o with
380             | Some(((fn, bd), tfn)) -> fn
381             | None -> L.const_pointer_null ctype_idx_pt)) bops) @ ([L.const_pointer_null ctype_call_
382             in
383
384 let ctype_of_datatype = function
385   | dt when dt = int_t -> ctype_int
386   | dt when dt = float_t -> ctype_float
387   | dt when dt = bool_t -> ctype_bool
388   | dt when dt = char_t -> ctype_char
389   | dt when dt = clist_t -> ctype_list
390 in
391
392 let build_getctypefn_cobj ctype_fn_idx cobj_p b =
393   let x2 = L.build_struct_gep cobj_p cobj_type_idx "x2" b in
394   let x3 = L.build_load x2 "x3" b in  (* x3: ctype_pt *)
395   let x4 = L.build_struct_gep x3 ctype_fn_idx "x4" b in
396   let fn_ptr = L.build_load x4 "fn_ptr" b in
397   fn_ptr
398 in
399
400 (** define helper functions for commonly used code snippets **)
401 let build_new_cobj_empty builder =
402   let objptr = L.build_malloc cobj_t "__new_objptr" builder in (* objptr: cobj_pt* *)
403   let datafieldptr = L.build_struct_gep objptr cobj_data_idx "datafieldptr" builder in  (* datafield
404   let ctypefieldptr = L.build_struct_gep objptr cobj_type_idx "ctypefieldptr" builder in
405   (objptr,datafieldptr,ctypefieldptr)
406 in
407
408 let build_new_cobj data_type builder =
409   (* malloc the new object and its data *)
410   let objptr = L.build_malloc cobj_t "__new_objptr" builder in (* objptr: cobj_pt *)
411   let dataptr = L.build_malloc data_type "__new_dataptr" builder in
412   let dataptr_as_i8ptr = L.build_bitcast dataptr char_pt "dataptr_as_i8" builder in
413
414   (* store ctypeptr in the struct *)
415   let ctypefieldptr = L.build_struct_gep objptr cobj_type_idx "ctypefieldptr" builder in
416   ignore(L.build_store (ctype_of_datatype data_type) ctypefieldptr builder);
417
418   (* store dataptr in the struct *)
419   let datafieldptr = L.build_struct_gep objptr cobj_data_idx "datafieldptr" builder in  (* datafield
```

```
420   let datafieldptr_as_i8ptrptr = L.build_bitcast datafieldptr (L.pointer_type char_pt) "datafieldptr
421   ignore(L.build_store dataptr_as_i8ptr datafieldptr_as_i8ptrptr builder);
422
423   (objptr, dataptr)
424 in
425
426 let build_new_cobj_init data_type value b =
427     let (objptr, dataptr) = build_new_cobj data_type b in
428     ignore(L.build_store value dataptr b);
429     objptr
430 in
431
432 let build_new_clist dataptr_of_cobj elm_pts builder =
433   (* len *)
434   let length = List.length elm_pts in
435   let len = L.const_int int_t length in
436
437   (* cap *)
438   let capacity = max length initial_list_size in
439   let cap = L.const_int int_t capacity in
440
441   (* dataptr: mallocs empty CObj array *)
442   let dataptr = L.build_malloc (L.array_type cobj_pt capacity) "__new_dataptr" builder in
443   let dataptr_as_i8ptr = L.build_bitcast dataptr char_pt "dataptr_as_i8" builder in
444
445   let elm_pts_as_cobjptrs = List.map (fun e ->
446     let elm_pt_as_cobjptr = L.build_bitcast e cobj_pt "elm_ptr_as_cobjptr" builder
447     in elm_pt_as_cobjptr) elm_pts in
448
449   (* null pointers to fill empty capacity *)
450   let elms_w_nulls = if List.length elm_pts_as_cobjptrs < capacity
451     then elm_pts_as_cobjptrs@(Array.to_list (Array.make (capacity - List.length elm_pts) (L.const_po
452     else elm_pts_as_cobjptrs in
453
454   (* stores the data *)
455   let store_elms elm idx =
456     let gep_addr = L.build_gep dataptr [|L.const_int int_t 0; L.const_int int_t idx|] "__elem_ptr" b
457     ignore(L.build_store elm gep_addr builder); ()
458   in
459   ignore(List.iter2 store_elms elms_w_nulls (seq capacity));
460
461   (* store dataptr the struct *)
462   let datafieldptr = L.build_struct_gep dataptr_of_cobj clist_data_idx "datafieldptr" builder in  (*
463   let datafieldptr_as_i8ptrptr = L.build_bitcast datafieldptr (L.pointer_type char_pt) "datafieldptr
464   ignore(L.build_store dataptr_as_i8ptr datafieldptr_as_i8ptrptr builder);
465
466   (* store len in the struct *)
467   let lenfieldptr = L.build_struct_gep dataptr_of_cobj clist_len_idx "lenfieldptr" builder in  (* le
468   ignore(L.build_store len lenfieldptr builder);
469
470   (* store cap in the struct *)
471   let capfieldptr = L.build_struct_gep dataptr_of_cobj clist_cap_idx "capfieldptr" builder in  (* ca
472   ignore(L.build_store cap capfieldptr builder);
473 in
```

```
474
475 (** manually making the ctype_ functions **)
476 (* does alloca, store, then load *)  (* note you should not use this if youre not using the values r
477 let boilerplate_till_load remote_cobj_p prettyname b =
478   ignore(L.set_value_name ("remote_"^prettyname) remote_cobj_p);
479   let cobj_pp = L.build_alloca cobj_pt (prettyname^"_p") b in
480   ignore(L.build_store remote_cobj_p cobj_pp b);
481   let cobj_p = L.build_load cobj_pp (prettyname^"_p") b in
482   cobj_p
483 in
484
485 (*let quick_def_fn fname ret_type formals_types =
486   let ftype = L.function_type ret_type formals_types in
487   let the_function = L.define_function fname ftype the_module in
488   let builder = L.builder_at_end context (L.entry_block the_function) in
489   (the_function, ftype, builder)
490   *)
491 let boilerplate_binop data_type fn b =
492   let formals_llvalues = (Array.to_list (L.params fn)) in
493   let [ remote_self_p; remote_other_p ] = formals_llvalues in
494
495   (* boilerplate *)
496   let self_p = boilerplate_till_load remote_self_p "self_p" b in
497   let other_p = boilerplate_till_load remote_other_p "other_p" b in
498
499   (* get data *)
500   let self_data = build_getdata_cobj data_type self_p b in
501   let other_data = build_getdata_cobj data_type other_p b in
502   (self_data, other_data)
503 in
504
505 let boilerplate_uop data_type fn b =
506   let formals_llvalues = (Array.to_list (L.params fn)) in
507   let [ remote_self_p ] = formals_llvalues in
508
509   (* boilerplate *)
510   let self_p = boilerplate_till_load remote_self_p "self_p" b in
511
512   (* get data *)
513   let self_data = build_getdata_cobj data_type self_p b in
514   (self_data)
515 in
516
517 let boilerplate_lop data_type fn b =
518     (* TODO: throw error if array bounds exceeded *)
519   let formals_llvalues = Array.to_list (L.params fn) in
520   let [ remote_self_p; remote_other_p ] = formals_llvalues in
521
522   (* boilerplate *)
523   let self_p = boilerplate_till_load remote_self_p "self_p" b in
524   let other_p = boilerplate_till_load remote_other_p "other_p" b in
525
526   (* get data *)
527   let self_data = build_getlist_cobj self_p b in
```

```
528   let other_data = build_getdata_cobj int_t other_p b in
529   (self_data, other_data)
530 in
531
532 List.iter (fun (t, bops) -> List.iter (function
533   | BOprt(o) -> (match o with
534     | Some(((fn, bd), tfn)) ->
535       let (tf, tp) = tfn in
536       let (self_data, other_data) = boilerplate_binop (get_t t) fn bd in
537       let result_data = tf self_data other_data "result_data" bd in
538       let result = build_new_cobj_init tp result_data bd in
539       ignore(L.build_ret result bd)
540     | None -> ())
541   | BUoprt(o) -> (match o with
542     | Some(((fn, bd), tfn)) ->
543       let (tf, tp) = tfn in
544       let (self_data) = boilerplate_uop (get_t t) fn bd in
545       let result_data = tf self_data "result_data" bd in
546       let result = build_new_cobj_init tp result_data bd in
547       ignore(L.build_ret result bd)
548     | None -> ())
549   | BLoprt(o) -> (match o with
550     | Some(((fn, bd), tfn)) ->
551       let (tf, tp) = tfn in
552       let (self_data, other_data) = boilerplate_lop (get_t t) fn bd in
553       let result_data = tf self_data other_data "result_data" bd in
554       let result = result_data in
555       ignore(L.build_ret result bd)
556     | None -> ())) bops) built_ops;
557
558
559   (* Functions! *)
560   (* building __call__ for ctype_func *)
561       let (fn,b) = (func_call_fn,func_call_b) in
562         let formals_llvalues = (Array.to_list (L.params fn)) in
563         let [ remote_self_p ; remote_argv ] = formals_llvalues in
564         let self_p = boilerplate_till_load remote_self_p "self_p" b in
565
566         (* manual boilerplate for argv since it's not a cobj *)
567         ignore(L.set_value_name ("remote_argv") remote_argv);
568         let argv_p = L.build_alloca cobj_ppt "argv_p" b in
569         ignore(L.build_store remote_argv argv_p b);
570         let argv = L.build_load argv_p "argv" b in
571
572         let fn_p = build_fnptr_of_cfo self_p b in
573         let result = L.build_call fn_p [|argv|] "result" b in
574         ignore(L.build_ret result b);
575
576
577
578 (** allocate for all the bindings and put them in a map **)
579 let build_binding_list_global coral_names =   (* coral_names: string list *)  (* returns a stringmap
580       List.fold_left (fun map name -> StringMap.add name (L.define_global name (L.const_pointer_null
581 and build_binding_list_local coral_names builder =   (* coral_names: string list *)  (* returns a st
```

```
582       List.fold_left (fun map name -> StringMap.add name (L.build_malloc cobj_pt name builder) map)
583 (*let example_binding = L.declare_global cobj_ppt "example_binding" the_module in*)
584 in
585
586 let name_of_bind = function
587     |Bind(name,_) -> name
588 in
589
590 let globals_map =
591     let globals_list = List.map name_of_bind (snd prgm)  (* snd prgrm is the bind list of globals *)
592 (*let globals_map = build_binding_list_global ["a";"b";"c"] in*)
593     build_binding_list_global globals_list
594 in
595 let lookup_global_binding coral_name =
596   StringMap.find coral_name globals_map
597 in
598
599 (* define printf *)
600 let printf_t : L.lltype =   (* define the type that the printf function shd be *)
601     L.var_arg_function_type int_t [| char_pt |] in
602 let printf_func : L.llvalue =   (* now use that type to declare printf (dont fill out the body just
603     L.declare_function "printf" printf_t the_module in
604
605 (** setup main() where all the code will go **)
606 let main_ftype = L.function_type int_t [||] in   (* ftype is the full llvm function signature *)
607 let main_function = L.define_function "main" main_ftype the_module in
608 let main_builder = L.builder_at_end context (L.entry_block main_function) in
609 let int_format_str = L.build_global_stringptr "%d\n" "fmt" main_builder
610 and float_format_str = L.build_global_stringptr "%g\n" "fmt" main_builder in
611 let init_state:state = {namespace=StringMap.empty; func=main_function; b=main_builder} in
612
613 (*
614 (* basic object creation test *)
615 let (objptr,dataptr) = build_new_cobj int_t main_builder in
616 ignore(L.build_store objptr (lookup_global_binding "a") main_builder);
617 ignore(L.build_store (L.const_int int_t 69) dataptr main_builder);
618 let x5 = build_getdata_cobj int_t objptr main_builder in
619 (**we'll need one of these per-fn actually**)
620 let int_format_str = L.build_global_stringptr "%d\n" "fmt" main_builder
621 and float_format_str = L.build_global_stringptr "%g\n" "fmt" main_builder in
622 let retval = L.build_call printf_func [| int_format_str ; (L.const_int int_t 10) |] "printf" main_bu
623 let r2 = L.build_call printf_func [| int_format_str ; x5 |] "printf" main_builder in
624
625
626 (* useful utility functions! *)
627 let names_of_bindlist bindlist =
628   List.map name_of_bind bindlist
629 in
630  helper fn: seq 4 == [0;1;2;3] *)
631 let seq len =
632   let rec aux len acc =
633     if len<0 then acc else aux (len-1) (len::acc)
634   in aux (len-1) []
635 in
```

```
636
637 let lookup coral_name namespace =
638     try StringMap.find coral_name namespace
639       with Not_found -> lookup_global_binding coral_name
640 in
641
642
643 let rec expr the_state typed_e =
644     let (namespace,the_function,b) = (the_state.namespace,the_state.func,the_state.b) in
645     let (e,ty) = typed_e in
646     match e with
647   | SLit lit -> (match lit with
648       | IntLit i -> build_new_cobj_init int_t (L.const_int int_t i) b
649       | BoolLit i -> build_new_cobj_init bool_t (L.const_int bool_t (if i then 1 else 0)) b
650       | FloatLit i -> build_new_cobj_init float_t (L.const_float float_t i) b
651       | StringLit i -> let elements = List.rev (Seq.fold_left (fun l ch ->
652           let cobj_of_char_ptr = build_new_cobj_init char_t (L.const_int char_t (Char.code ch)) b in
653           cobj_of_char_ptr::l) [] (String.to_seq i)) in
654         let (objptr, dataptr) = build_new_cobj clist_t b in
655         let _ = build_new_clist dataptr elements b in
656         objptr
657   )
658   | SVar x -> let coral_name = x in
659       L.build_load (lookup coral_name namespace) coral_name b
660   | SCall(fexpr, arg_expr_list, sfdecl) ->
661           tstp "entering SCALL";
662       let argc = List.length arg_expr_list
663       (* eval the arg exprs *)
664       and llargs = List.map (expr the_state) (List.rev arg_expr_list) in
665       let cobj_p_arr_t = L.array_type cobj_pt argc in
666       (* allocate stack space for argv *)
667       let argv_as_arr = L.build_alloca cobj_p_arr_t "argv_arr" b in
668       (* store llargs values in argv *)
669
670       let store_arg llarg idx =
671         let gep_addr = L.build_gep argv_as_arr [|L.const_int int_t 0; L.const_int int_t idx|] "arg"
672         ignore(L.build_store llarg gep_addr b);()
673       in
674
675       ignore(List.iter2 store_arg llargs (seq argc));
676       (*let argv_as_arr_filled = List.fold_left2 (fun arr llarg idx -> L.build_insertvalue arr llarg
677       *)
678       (*ignore(L.build_store (L.const_array cobj_pt (Array.of_list llargs)) argv_as_arr b);*)
679       let argv = L.build_bitcast argv_as_arr cobj_ppt "argv" b in
680
681       (* now we have argv! so we just need to get the fn ptr and call it *)
682       (*let fname = name_of_bind fname_sbind in
683           tstp ("SCALL of "^fname);
684       let caller_cobj_p = L.build_load (lookup fname namespace) fname b in
685 *)
686       let caller_cobj_p = expr the_state fexpr in
687       let call_ptr = build_getctypefn_cobj ctype_call_idx caller_cobj_p b in
688       let result = L.build_call call_ptr [|caller_cobj_p;argv|] "result" b in
689       result
```

```
690   | SListAccess(e1,e2)  -> expr the_state (SBinop(e1,ListAccess,e2),ty)
691   | SBinop(e1, op, e2) ->
692     let e1' = expr the_state e1
693     and e2' = expr the_state e2 in
694     let fn_idx = match op with
695       | Add       -> ctype_add_idx
696       | Sub       -> ctype_sub_idx
697       | Mul       -> ctype_mul_idx
698       | Div       -> ctype_div_idx
699       | Exp       -> ctype_exp_idx
700       | Eq        -> ctype_eq_idx
701       | Neq       -> ctype_neq_idx
702       | Less      -> ctype_lesser_idx
703       | Leq       -> ctype_leq_idx
704       | Greater   -> ctype_greater_idx
705       | Geq       -> ctype_geq_idx
706       | And       -> ctype_and_idx
707       | Or        -> ctype_or_idx
708       | ListAccess -> ctype_idx_idx in
709     let fn_p = build_getctypefn_cobj fn_idx e1' b in
710       L.build_call fn_p [| e1'; e2' |] "binop_result" b
711   | SUnop(op, e) ->
712     let e' = expr the_state e in
713     let fn_idx = match op with
714       | Neg          -> ctype_neg_idx
715       | Not          -> ctype_not_idx in
716     let fn_p = build_getctypefn_cobj fn_idx e' b in
717       L.build_call fn_p [| e' |] "uop_result" b
718   | SList(el, t) ->
719     let elements = List.map (fun e ->
720       expr the_state e) el in
721     let (objptr, dataptr) = build_new_cobj clist_t b in
722     let _ = build_new_clist dataptr elements b in
723     objptr
724   (* | Snoexper *)
725 in
726
727   let add_terminal the_state instr =
728     match L.block_terminator (L.insertion_block the_state.b) with
729    Some _ -> ()   (* do nothing if terminator is there *)
730     | None -> ignore (instr the_state.b)
731   in
732
733 let change_builder_state old_state b =
734     {namespace=old_state.namespace;func=old_state.func;b=b}
735 in
736
737 let rec stmt the_state s =   (* namespace comes first bc never gets modified unless descending so it
738     let (namespace,the_function,b) = (the_state.namespace,the_state.func,the_state.b) in
739     match s with
740     |SBlock s -> List.fold_left stmt the_state s
741     |SExpr e ->  ignore(expr the_state e); the_state
742     |SAsn (lvalue_list, e) -> (*L.dump_module the_module;*) tstp "entering asn";
743       let e' = expr the_state e in tstp "passing checkpoint";
```

```
744       let get_name = function
745         | SLVar (Bind (name, explicit_t)) -> name
746         | _ -> raise (Failure "CodegenError: this kind of assignment has not been implemented.")
747
748       in let names = List.map get_name lvalue_list in
749       List.iter (fun name -> ignore (L.build_store e' (lookup name namespace) b)) names ; tstp "leav
750     (*|SReturn  (* later *)*)
751     |SNop -> the_state
752     |SIf (predicate, then_stmt, else_stmt) ->
753        let bool_val = build_getdata_cobj bool_t (expr the_state predicate) b in
754        (*let bool_val = (L.const_bool bool_t 1) in*)
755          let merge_bb = L.append_block context "merge" the_function in
756            let build_br_merge = L.build_br merge_bb in
757              let then_bb = L.append_block context "then" the_function in
758              let new_state = change_builder_state the_state (L.builder_at_end context then_bb) in
759                add_terminal (stmt new_state then_stmt) build_br_merge;
760              let else_bb = L.append_block context "else" the_function in
761              let new_state = change_builder_state the_state (L.builder_at_end context else_bb) in
762                add_terminal (stmt new_state else_stmt) build_br_merge;  (* same deal as with 'then'
763                  ignore(L.build_cond_br bool_val then_bb else_bb b);
764                  let new_state = change_builder_state the_state (L.builder_at_end context merge_bb )
765     | SWhile (predicate, body) ->
766       let pred_bb = L.append_block context "while" the_function in
767         ignore(L.build_br pred_bb b);
768       let body_bb = L.append_block context "while_body" the_function in
769       let new_state = change_builder_state the_state (L.builder_at_end context body_bb) in
770         add_terminal (stmt new_state body) (L.build_br pred_bb);
771       let pred_builder = L.builder_at_end context pred_bb in
772       let new_state = change_builder_state the_state (L.builder_at_end context pred_bb) in
773       let bool_val = build_getdata_cobj bool_t (expr new_state predicate) pred_builder in
774       let merge_bb = L.append_block context "merge" the_function in
775         ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
776       let new_state = change_builder_state the_state (L.builder_at_end context merge_bb) in
777         new_state
778     | SFor(var, lst, body) ->
779        (* initialize list index variable and list length *)
780        let objptr = expr the_state lst in
781        let listptr = build_getlist_cobj objptr b in
782        let nptr = L.build_alloca int_t "nptr" b in
783          ignore(L.build_store (L.const_int int_t (0)) nptr b);
784        let n = L.build_load nptr "n" b in
785        let ln = build_getlen_clist listptr b in
786
787        (* iter block *)
788        let iter_bb = L.append_block context "iter" the_function in
789          ignore(L.build_br iter_bb b);
790
791        let iter_builder = L.builder_at_end context iter_bb in
792        let n = L.build_load nptr "n" iter_builder in
793        let nnext = L.build_add n (L.const_int int_t 1) "nnext" iter_builder in
794          ignore(L.build_store nnext nptr iter_builder);
795
796        let iter_complete = (L.build_icmp L.Icmp.Sge) n ln "iter_complete" iter_builder in (* true if
797
```

```
798         (* body of for loop *)
799         let body_bb = L.append_block context "for_body" the_function in
800         let body_builder = L.builder_at_end context body_bb in
801
802         let name = name_of_bind var  in
803         let elmptr = build_idx listptr n "binop_result" body_builder in
804           ignore(L.build_store elmptr (lookup name namespace) body_builder);
805         let new_state = change_builder_state the_state body_builder in
806           add_terminal (stmt new_state body) (L.build_br iter_bb);
807
808         let merge_bb = L.append_block context "merge" the_function in
809           ignore(L.build_cond_br iter_complete merge_bb body_bb iter_builder);
810         let new_state = change_builder_state the_state (L.builder_at_end context merge_bb) in
811           new_state
812     | SPrint e -> (* TODO make this depend on runtime for dynamic types, implement strings *)
813       let (_, ty) = e in (match ty with
814         | Int -> ignore(L.build_call printf_func [| int_format_str ; (build_getdata_cobj int_t (expr t
815         | Float -> ignore(L.build_call printf_func [| float_format_str ; (build_getdata_cobj float_t (
816         | _ -> ignore(L.build_call printf_func [| int_format_str ; (build_getdata_cobj int_t (expr the
817       )
818
819     | STransform(a, b, c) -> the_state
820
821     | SFunc sfdecl ->
822         (* outer scope work: point binding to new cfuncobj *)
823         let fname = sfdecl.sfname in
824         let the_function = L.define_function fname userdef_fn_t the_module in
825
826         (* manually design the fn object w proper data & type ptrs and put in bind *)
827         let _ =
828           let (fn_obj,datafieldptr,ctypefieldptr) = build_new_cobj_empty b in
829           let dfp_as_fp = L.build_bitcast datafieldptr (L.pointer_type userdef_fn_pt) "dfp_as_fp" b in
830           ignore(L.build_store the_function dfp_as_fp b);  (* store fnptr *)
831           ignore(L.build_store ctype_func ctypefieldptr b);  (* store ctype ptr *)
832           (* store new object in appropriate binding *)
833           ignore(L.build_store fn_obj (lookup fname namespace) b)
834         in
835
836         let fn_b = L.builder_at_end context (L.entry_block the_function) in
837         (* update the namespace *)
838         let fn_namespace =
839           let local_names = names_of_bindlist sfdecl.slocals
840           and formal_names = names_of_bindlist sfdecl.sformals in
841           (*List.iter print_endline local_names;*)
842           let argc = List.length formal_names
843           and argv = Array.get (L.params the_function) 0 in (* argv is first/only arg *)
844           let cobj_p_arr_pt = L.pointer_type (L.array_type cobj_pt argc) in
845           let formals_arr_p = L.build_bitcast argv cobj_p_arr_pt "formals_arr_p" fn_b in
846           (* now formals_arr_p is a ptr to an array of cobj_ps which are the formals *)
847           let formals_arr = L.build_load formals_arr_p "formals_arr" fn_b in
848           (*let formal_co_pp_list = List.map (fun idx -> L.build_gep formals_arr_p [|0;idx|] ("arg"^(s
849
850
851           (* Very important! the actual extraction of the formals from formals_arr *)
```

```
852         let formal_vals = List.map (fun idx -> L.build_extractvalue formals_arr idx ("arg"^(string_o
853           (* now formal_vals is a list of co_ps *)
854 
855         let add_formal namespace_wip name cobj_p =  (* alloc a formal *)
856             L.set_value_name name cobj_p;  (* cosmetic *)
857             let alloca = L.build_alloca cobj_pt name fn_b in
858             ignore(L.build_store cobj_p alloca fn_b);
859             StringMap.add name alloca namespace_wip
860         and add_local namespace_wip name =  (* alloc a local *)
861             let alloca = L.build_alloca cobj_pt name fn_b in
862             StringMap.add name alloca namespace_wip
863         in   (* pull in add_formal and add_local *)
864         let added_locals = List.fold_left add_local namespace local_names in
865         let added_formals = List.fold_left2 add_formal added_locals formal_names formal_vals
866         in added_formals  (* fn_namespace is now equal to this *)
867       in
868 
869       let int_format_str = L.build_global_stringptr "%d\n" "fmt" fn_b
870       and float_format_str = L.build_global_stringptr "%f\n" "fmt" fn_b in
871 
872       (* build function body by calling stmt! *)
873       let build_return some_b = L.build_ret (build_new_cobj_init int_t (L.const_int int_t 69) some_b
874       let fn_state:state = {namespace=fn_namespace;func=the_function;b=fn_b} in
875       add_terminal (stmt fn_state sfdecl.sbody) build_return;the_state  (* SFunc() returns the origi
876   | SReturn e ->
877           L.build_ret (expr the_state e) b; the_state
878 
879 
880 in
881 
882 (*
883 let ex = SIf(SLit(BoolLit(true)),SAsn([WeakBind("a",Dyn)],SLit(boolLit(42))),SNop) in
884 let main_builder = stmt StringMap.empty main_builder ex in
885 
886 
887 let final_state = stmt init_state (SBlock(fst prgm)) in
888 (*
889 let f = (L.define_function "myfn" (L.function_type (L.void_type context) [||]) the_module) in
890 let myb = L.builder_at_end context (L.entry_block f) in
891 ignore(L.build_ret_void myb);
892 let main_builder = stmt StringMap.empty main_builder (SBlock(fst prgm)) in
893 let f = (L.define_function "myfn" (L.function_type (L.void_type context) [||]) the_module) in
894 let myb = L.builder_at_end context (L.entry_block f) in
895 ignore(L.build_ret_void myb);
896 *)
897 
898 
899 (*
900 let ex = SAsn([WeakBind("a",Dyn)],SLit(BoolLit(42))) in
901 ignore(stmt StringMap.empty main_builder ex);
902 *)
903 
904 (*
905 let objptr = L.build_load (lookup_global_binding "a") "objptr" main_builder in
```

```
let x5 = build_getdata_cobj int_t objptr main_builder in
(**we'll need one of these per-fn actually**)
(*
let objptr = L.build_load (lookup_global_binding "b") "objptr" main_builder in
let x6 = build_getdata_cobj int_t objptr main_builder in

let retval = L.build_call printf_func [| int_format_str ; (L.const_int int_t 10) |] "printf" main_bu
let r2 = L.build_call printf_func [| int_format_str ; x5 |] "printf" main_builder in
(*let r2 = L.build_call printf_func [| int_format_str ; x6 |] "printf" main_builder in*)


ignore(L.build_ret (L.const_int int_t 0) final_state.b);
(* pm(); *)(* prints module *)
the_module  (* return the resulting llvm module with all code!! *)
```

coral.ml

```
1
```

interpret.ml (Jacob Austin)

```
     * open Ast
en Sast
en Getopt (* package used to handle command line arguments *)
en Utilities
 Interpret defined a minimal interpreter for a very small subset of our language. This
terpreter is capable of handling floating point types and a subset of the allowed expressions
d statements *)
 expr -> (float, map), used to evaluate expressions *)
t rec eval_expr map = function
| Lit(x) -> (match x with
   | FloatLit(y) -> (y, map)
   | IntLit(y) -> (float_of_int y, map)
   | BoolLit(y) -> (float_of_bool y, map)
   | StringLit(_) -> raise (Runtime "NotImplementedError: Strings have not yet been implemented!");
 )
| List(x) -> (raise (Runtime "NotImplementedError: Lists have not yet been implemented!"); )
| Var(Bind(x, _)) ->
  (try
     let Expr(v) = (StringMap.find x map) in
     eval_expr map v
    with Not_found -> Printf.printf "NameError: name '%s' is not defined!\n" x; flush stdout; raise
  )
| Unop(op, v) ->
  let (v1, m1) = eval_expr map v in
  (match op with
    | Neg -> (-.v1, m1)
    | Not -> if v1 = 0.0 then (1.0, m1) else (0.0, m1)
  )
| Call(x, args) -> match x with
  | Var(Bind(name, typ)) ->
    (try let Func(_, a, ex) = StringMap.find name map in
```

```
32        let zipped = zip a args in
33        let m1 = List.fold_left add_to_map map zipped in
34        let (v2, _) = eval_stmt m1 ex in (v2, map)
35      with
36        Not_found -> Printf.printf "NameError: name '%s' is not defined!\n" name; flush stdout; raise
37      )
38    | _ -> raise (Runtime "NotImplementedError: Anonymous functions have not been implemented!");
39  | Method(_, _, _) -> raise (Runtime "NotImplementedError: Methods have not yet been implemented!");
40  | Field(_, _) -> raise (Runtime "NotImplementedError: Fields have not yet been implemented!");
41
42  | Binop(e1, op, e2) ->
43    let (v1, m1) = eval_expr map e1 in
44    let (v2, m2) = eval_expr m1 e2 in
45    (match op with
46      | Add -> (v1 +. v2, m2)
47      | Sub -> (v1 -. v2, m2)
48      | Mul -> (v1 *. v2, m2)
49      | Div -> (v1 /. v2, m2)
50      | Exp -> (v1 ** v2, m2)
51      | Greater -> (float_of_bool (v1 > v2), m2)
52      | Less -> (float_of_bool (v1 < v2), m2)
53      | Geq -> (float_of_bool (v1 >= v2), m2)
54      | Leq -> (float_of_bool (v1 <= v2), m2)
55      | Neq -> (float_of_bool (v1 <> v2), m2)
56      | And -> (float_of_bool (v1 <> 0.0 && v2 <> 0.0), m2)
57      | Or -> (float_of_bool (v1 <> 0.0|| v2 <> 0.0), m2)
58      | Eq -> (float_of_bool (v1 = v2), m2)
59    )
60  | _ -> raise (Runtime "NotImplementedError: This expression has not yet been implemented!");
61   helper function used to add a list of function arguments to the map of local variables *)
62  d add_to_map map = function
63  | (Bind(a, t), b) ->
64    let (v1, m1) = eval_expr map b in
65    let m2 = (StringMap.add a (Expr (Lit (FloatLit(v1)))) m1) in m2
66   takes a statement and evaluates it, returning a float and a map, used to evaluate all expressions *
67  d eval_stmt map = function
68  | Block(a) -> main map 0.0 a
69  | Func(Bind(a, t), b, c) -> let m1 = (StringMap.add a (Func(Bind(a, t), b, c)) map) in (0.0, m1)
70  | Class(a, b) -> raise (Runtime "NotImplementedError: Classes have not yet been implemented!");
71  | Expr(a) -> let (x, m1) = eval_expr map a in (x, m1)
72  | If(a, b, c) -> let (x, m1) = eval_expr map a in if x = 1.0 then eval_stmt m1 b else eval_stmt m1 c
73  | For(Bind(a, t), b, c) -> raise (Runtime "NotImplementedError: For loops have not yet been implemen
74  | While(a, b) -> let rec recurse map = let (x, m1) = eval_expr map a in if x = 1.0 then let (x1, m2)
75  | Return(a) ->  eval_expr map a;       (* expr *)
76  | Asn(names, v) -> let (v1, m1) = eval_expr map v in let m2 = List.fold_left (fun m (Var(Bind(name,
77  | Type(_) -> (0.0, map)
78  | Print(_) -> (0.0, map)
79  | Nop -> (0.0, map)
80   takes a stmt list, iterates through the list and evaluates it in order *)
81  d main map value = function
82  | [] -> (value, map)
83  | a :: t -> let (v1, m1) = eval_stmt map a in main m1 v1 t *)
```

---

parser.mly (Jacob Austin

```
    %{
pen Ast


 A simple LR parser in OcamlYacc implementing a minimal Python syntax with some
mitations on exceptions, generators, importing modules, and some other features.

e first parsing target, tokenize, is a helper marser used to simply parse the
xing stream into a list that can be used to extract indentation/tabs for the
thon-style indentation based parsing scheme. The second is the full parser */

oken NOELSE ASN EQ NEQ LT GT LEQ GEQ PLUS MINUS TIMES DIVIDE PLUSEQ MINUSEQ TIMESEQ DIVIDEEQ EXPEQ
oken EXP NOT NEG SEP AND OR ARROW NOP TYPE PRINT FUNC
oken TAB COLON EOF EOL IF ELSE FOR WHILE COMMA DEF IN TRUE FALSE IS RETURN NONE DOT
oken BOOL INT FLOAT STRING BOOLARR INTARR FLOATARR STRINGARR
oken CLASS IMPORT CEND RANGE
oken INDENT DEDENT
oken LPAREN RPAREN
oken LBRACK RBRACK
oken LBRACE RBRACE
oken <string> VARIABLE

oken <float> FLOAT_LITERAL
oken <string> STRING_LITERAL
oken <int> INT_LITERAL
oken <bool> BOOL_LITERAL

onassoc NOFIELD /* handles shift-reduce error for fields and methods in classes */
onassoc FIELD

onassoc NOELSE /* handles shift-reduce error for else and noelse clauses */
onassoc ELSE

ight ASN PLUSEQ MINUSEQ DIVIDEEQ TIMESEQ
eft DOT
eft OR
eft AND
eft EQ NEQ
eft LT GT LEQ GEQ
eft PLUS MINUS
eft TIMES DIVIDE
ight EXP EXPEQ
ight NOT NEG
eft SEP

onassoc LPAREN LBRACK LBRACE
onassoc RPAREN RBRACK RBRACE

 this tokenize target is simply used to take the input lexbuf stream and produce
list of Parser.tokens for use by the indentation method in Coral.ml */

tart tokenize
ype <token list> tokenize
```

```
54
55
56  this program target is the main target used to parse the entire program */
57
58 tart program
59 ype <Ast.stmt list> program
60
61
62
63 kenize:
64 | seq EOL { $1 @ [EOL] }
65 | seq CEND { $1 @ [EOL] }
66 | CEND { CEND :: [EOL] }
67 | EOL { NOP :: [EOL] }
68
69  seq: an auxillary target used to handle shift reduce errors in tokenize */
70
71 q:
72 | token { [$1] }
73 | token seq { $1 :: $2 }
74
75 ken:
76 | COLON { COLON }
77 | TAB { TAB }
78 | ARROW { ARROW }
79 | RETURN { RETURN }
80 | NOT { NOT }
81 | IF { IF }
82 | ELSE { ELSE }
83 | FOR { FOR }
84 | WHILE { WHILE }
85 | DEF { DEF }
86 | COMMA { COMMA }
87 | NEQ { NEQ }
88 | LT { LT }
89 | GT { GT }
90 | LEQ { LEQ }
91 | GEQ { GEQ }
92 | AND { AND }
93 | OR { OR }
94 | IN { IN }
95 | TRUE { TRUE }
96 | FALSE { FALSE }
97 | IS { IS }
98 | PLUS { PLUS }
99 | MINUS { MINUS }
100 | TIMES { TIMES }
101 | DIVIDE { DIVIDE }
102 | EXP { EXP }
103 | PLUSEQ { PLUSEQ }
104 | MINUSEQ { MINUSEQ }
105 | TIMESEQ { TIMESEQ }
106 | DIVIDEEQ { DIVIDEEQ }
107 | EXPEQ { EXPEQ }
```

```
| LPAREN { LPAREN }
| RPAREN { RPAREN }
| LBRACK { LBRACK }
| RBRACK { RBRACK }
| LBRACE { LBRACE }
| RBRACE { RBRACE }
| EQ { EQ }
| ASN { ASN }
| SEP { SEP }
| BOOL { BOOL }
| INT { INT }
| FLOAT { FLOAT }
| FUNC { FUNC }
| STRING { STRING }
| INTARR { INTARR }
| FLOATARR { FLOATARR }
| STRINGARR { STRINGARR }
| BOOLARR { BOOLARR }
| INDENT { INDENT }
| DEDENT { DEDENT }
| VARIABLE { VARIABLE($1) }
| FLOAT_LITERAL { FLOAT_LITERAL($1) }
| INT_LITERAL { INT_LITERAL($1) }
| BOOL_LITERAL { BOOL_LITERAL($1) }
| STRING_LITERAL { STRING_LITERAL($1) }
| EOF { EOF }
| CLASS { CLASS }
| NONE { NONE }
| DOT { DOT }
| TYPE { TYPE }
| PRINT { PRINT }
| IMPORT { IMPORT }
| RANGE { RANGE }

 program: the main program parser target. read a list of statements until EOF is reached.
nstructed backwards per the usual OCaml functional list syntax. */

ogram: stmt_list EOF { List.rev $1 }

 stmt_list: a list of statements in the global scope or a function body. modified from Micro C. */

mt_list:
| { [] }
| stmt_list stmt { $2 :: $1 }

 stmt: this defines all possible statements in the Coral language. Those possible statements are:

 an expression
 another statement (in case of unusual behavior in the parser
 a class with a name and block of statements
 a function with optional typed arguments
 a function with explicit return type
 a for loop
```

```
 a while loop
 a list of names or other valid lvalue expressions. will be expanded as more lvalues are supported
 hard-coded type statements
 hard-coded print statements
 no operation statements

her statements can be added by defining the appropriate syntax, and adding a new class of statements
 the ast.ml stmt type.


mt:
| expr SEP { Expr $1 }
| stmt SEP { $1 }
| IMPORT VARIABLE SEP { Import($2) }
| CLASS VARIABLE COLON stmt_block { Class($2, $4) }
| DEF VARIABLE LPAREN formals_opt RPAREN COLON stmt_block { Func(Bind($2, Dyn), $4, $7) }
| DEF VARIABLE LPAREN formals_opt RPAREN ARROW typ COLON stmt_block { Func(Bind($2, $7), $4, $9) }
| RETURN expr SEP { Return $2 }
| IF expr COLON stmt_block %prec NOELSE { If($2, $4, Block([])) }
| IF expr COLON stmt_block ELSE COLON stmt_block { If($2, $4, $7) } /* to do figure out (Block) */
| FOR bind_opt IN expr COLON stmt_block { For($2, $4, $6) }
| FOR bind_opt IN RANGE LPAREN expr RPAREN COLON stmt_block { Range($2, $6, $9) }
| WHILE expr COLON stmt_block { While($2, $4) }
| formal_asn_list ASN expr { Asn(List.rev $1, $3) }
| lvalue PLUSEQ expr { Asn([$1], Binop($1, Add, $3)) }
| lvalue MINUSEQ expr { Asn([$1], Binop($1, Sub, $3)) }
| lvalue TIMESEQ expr { Asn([$1], Binop($1, Mul, $3)) }
| lvalue DIVIDEEQ expr { Asn([$1], Binop($1, Div, $3)) }
| lvalue EXPEQ expr { Asn([$1], Binop($1, Exp, $3)) }
| TYPE LPAREN expr RPAREN { Type($3) }
| PRINT LPAREN expr RPAREN { Print($3) }
| NOP { Nop }

rmal_asn_list:
| lvalue { [$1] }
| formal_asn_list ASN lvalue { $3 :: $1 }

alue:
| bind_opt { Var $1 }
| list_access { $1 }

 inplace_op:
| lvalue PLUSEQ expr { Asn([$1], Binop($1, Add, $3)) }
| lvalue MINUSEQ expr { Asn([$1], Binop($1, Sub, $3)) }
| lvalue TIMESEQ expr { Asn([$1], Binop($1, Mul, $3)) }
| lvalue DIVIDEEQ expr { Asn([$1], Binop($1, Div, $3)) }
| lvalue EXPEQ expr { Asn([$1], Binop($1, Exp, $3)) } */

 bind_opt: optional type target, for variables of the form x or x : type */

```

```
211 nd_opt:
212 | VARIABLE { Bind($1, Dyn) }
213 | VARIABLE COLON typ { Bind($1, $3) }
214
215  list_access: list access of the form name[expr] or name[expr : expr].
216 is permits invalid access and needs to be checked in semant and at runtime. */
217
218 st_access:
219 | expr LBRACK expr RBRACK { ListAccess($1, $3) }
220 | expr LBRACK expr COLON expr RBRACK { ListSlice($1, $3, $5) }
221
222  stmt_block: a statement block contained within a function, class, loop, or conditional.
223 limited by an indent and dedent block introduced by the indentation parser in coral.ml */
224
225 mt_block:
226 | INDENT SEP stmt_list DEDENT { Block(List.rev $3) }
227
228  formals_opt: an optional formal name in a function declaration */
229
230 rmals_opt:
231 | { [] }
232 | formal_list { List.rev $1 }
233
234  formal_list: a list of optional formal names in a function declaration */
235
236 rmal_list:
237 | bind_opt { [$1] }
238 | formal_list COMMA bind_opt { $3 :: $1 }
239
240  actuals_opt: an optional expression in a function call */
241
242 tuals_opt:
243 | { [] }
244 | actuals_list { List.rev $1 }
245
246  actuals_list: an optional list of expressions in a function call */
247
248 tuals_list:
249 | expr { [$1] }
250 | actuals_list COMMA expr { $3 :: $1 }
251
252  typ: the possible type attributes that can be attached to an optionally typed variable. */
253
254 p:
255 | FLOAT { Float }
256 | INT { Int }
257 | BOOL { Bool }
258 | STRING { String }
259 | FLOATARR { FloatArr }
260 | INTARR { IntArr }
261 | BOOLARR { BoolArr }
262 | STRINGARR { StringArr }
263 | FUNC { FuncType }
```

```
 expr: these are all possible expressions allowed in the Coral language. each
pression corresponds to an expr object in the ast.ml file. Expressions are anything
at return a value, i.e. can be assigned to a variable. These include lists, list access
d list slice, methods and fields, function calls, binary and unary operations, and literals. */

pr:
list_access { $1 }
VARIABLE { Var(Bind($1, Dyn)) }
expr LPAREN actuals_opt RPAREN { Call($1, $3) }
expr DOT VARIABLE LPAREN actuals_opt RPAREN { Method($1, $3, $5) }
expr DOT VARIABLE %prec FIELD { Field($1, $3) }
MINUS expr %prec NEG { Unop(Neg, $2) }
NOT expr %prec NOT { Unop(Not, $2) }
LPAREN expr RPAREN { $2 }
FLOAT_LITERAL { Lit(FloatLit($1)) }
BOOL_LITERAL { Lit(BoolLit($1)) }
INT_LITERAL { Lit(IntLit($1)) }
STRING_LITERAL { Lit(StringLit($1)) }
LBRACK actuals_opt RBRACK { List($2) }
expr EQ expr { Binop($1, Eq, $3) }
expr NEQ expr { Binop($1, Neq, $3) }
expr LT expr { Binop($1, Less, $3) }
expr GT expr { Binop($1, Greater, $3) }
expr LEQ expr { Binop($1, Leq, $3) }
expr GEQ expr { Binop($1, Geq, $3) }
expr AND expr { Binop($1, And, $3) }
expr OR expr { Binop($1, Or, $3) }
expr PLUS expr { Binop($1, Add, $3) }
expr MINUS expr { Binop($1, Sub, $3) }
expr TIMES expr { Binop($1, Mul, $3) }
expr DIVIDE expr { Binop($1, Div, $3) }
expr EXP expr { Binop($1, Exp, $3) }
```

---

sast.ml (Jacob Austin)

```
    open Ast


at semant does:
1. undeclared identifiers (get a list of globals and locals for each function, check to make sure th
2. correct number of arguments in function call
3. correct types of explicitly declared variables
4. return types in the right place
5. duplicate formal arguments


pe sprogram = sstmt list * bind list

```

```
14 d sfunc_decl = {
15 styp : typ;
16 sfname : string;
17 sformals : bind list;
18 slocals : bind list;
19 sbody : sstmt
20
21
22 d sexp =
23 | SBinop of sexpr * operator * sexpr (* (left sexpr, op, right sexpr) *)
24 | SLit of literal (* literal *)
25 | SVar of string (* see above *)
26 | SUnop of uop * sexpr (* (uop, sexpr ) *)
27 | SCall of sexpr * sexpr list * sstmt (* SVar or SCall, list of args, SFunc) *) (* sstmt is SNop if
28 | SMethod of sexpr * string * sexpr list (* not implemented *)
29 | SField of sexpr * string (* not implemented *)
30 | SList of sexpr list * typ (* (list of expressions, inferred type) *)
31 | SNoexpr (* no expression *)
32 | SListAccess of sexpr * sexpr (* not implemented *)
33 | SListSlice of sexpr * sexpr * sexpr (* not implemented *)
34
35 d sexpr = sexp * typ
36
37 d sstmt = (* this can be refactored using Blocks, but I haven't quite figured it out yet *)
38 | SFunc of sfunc_decl (* (name, return type), list of formals, list of locals, body) *)
39 | SBlock of sstmt list (* block found in function body or for/else/if/while loop *)
40 | SExpr of sexpr (* see above *)
41 | SIf of sexpr * sstmt * sstmt (* condition, if, else *)
42 | SFor of bind * sexpr * sstmt (* (variable, list, body (block)) *)
43 | SWhile of sexpr * sstmt (* (condition, body (block)) *)
44 | SReturn of sexpr (* return statement *)
45 | SClass of bind * sstmt (* not implemented *)
46 | SAsn of lvalue list * sexpr (* x : int = sexpr, (Bind(x, int), sexpr) *)
47 | STransform of string * typ * typ
48 | SPrint of sexpr
49 | SNop
50
51 d lvalue =
52 | SLVar of bind
53 | SLListAccess of sexpr * sexpr
54 | SLListSlice of sexpr * sexpr * sexpr
55 | SLExpr of sexpr
56
57 t concat_end delim = List.fold_left (fun a c -> a ^ delim ^ c) ""
58 t append_list v = List.map (fun c -> c ^ v)
59
60 t rec string_of_sexpr (e, t) = "(" ^ string_of_sexp e ^ ": " ^ string_of_typ t ^ ")"
61
62 d string_of_sbind = function
63 | Bind(s, t) -> s ^ ": " ^ string_of_typ t
64
65 d string_of_sexp = function
66 | SBinop(e1, o, e2) -> string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
67 | SLit(l) -> string_of_lit l
```

```
| SVar(str) -> str
| SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
| SCall(e, el, s) -> string_of_sexpr e ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")
| SMethod(obj, m, el) -> string_of_sexpr obj ^ "." ^ m ^ "(" ^ String.concat ", " (List.map string_o
| SField(obj, s) -> string_of_sexpr obj ^ "." ^ s
| SList(el, t) -> string_of_typ t ^ " list : " ^ String.concat ", " (List.map string_of_sexpr el)
| SListAccess(e1, e2) -> string_of_sexpr e1 ^ "[" ^ string_of_sexpr e2 ^ "]"
| SListSlice(e1, e2, e3) -> string_of_sexpr e1 ^ "[" ^ string_of_sexpr e2 ^ ":" ^ string_of_sexpr e3
| SNoexpr -> ""

d string_of_sstmt depth = function
| SFunc({ styp; sfname; sformals; slocals; sbody }) -> "def " ^ sfname ^ "(" ^ String.concat ", " (L
| SBlock(sl) -> concat_end (String.make (2 * depth) ' ') (append_list "\n" (List.map (string_of_sstm
| SExpr(e) -> string_of_sexpr e
| SIf(e, s1, s2) ->  "if " ^ string_of_sexpr e ^ ":\n" ^ string_of_sstmt depth s1 ^ (String.make (2
| SFor(b, e, s) -> "for " ^ string_of_sbind b ^ " in " ^ string_of_sexpr e ^ ":\n" ^ string_of_sstmt
| SWhile(e, s) -> "while " ^ string_of_sexpr e ^ ":\n" ^ string_of_sstmt depth s
| SReturn(e) -> "return " ^ string_of_sexpr e
| SClass(b, s) -> "class " ^ string_of_sbind b ^ ":\n" ^ string_of_sstmt depth s
| SAsn(lvalues, e) -> String.concat ", " (List.map string_of_lvalue lvalues) ^ " = "  ^ string_of_se
| STransform(s, t1, t2) -> "transform " ^ s ^ ": " ^ string_of_typ t1 ^ " -> " ^ string_of_typ t2
| SPrint(e) -> "print(" ^ string_of_sexpr e ^ ")"
| SNop -> ""

d string_of_lvalue = function
| SLVar(sbind) -> string_of_sbind sbind
| SLListAccess(e1, e2) ->  string_of_sexpr e1 ^ "[" ^ string_of_sexpr e2 ^ "]"
| SLListSlice(e1, e2, e3) -> string_of_sexpr e1 ^ "[" ^ string_of_sexpr e2 ^ ":" ^ string_of_sexpr e
| SLExpr(e) -> string_of_sexpr e

d string_of_sprogram (sl, bl) = String.concat "\n" (List.map (string_of_sstmt 1) sl) ^ "\n\nGlobals:

pe flag = {
noeval : bool;
cond : bool;
forloop : bool;

```

scanner.mll (Jacob Austin)

```
     {
pen Parser
xception Eof

let strip_quotes str =
match String.length str with
| 0 | 1 | 2 -> ""
| len -> String.sub str 1 (len - 2)


 a simple lexer implemented in ocamllex for lexing a Coral/Python program, with a few minor limitati
veral Python features have not yet been implemented and will throw errors in the lexing stage if
countered. Coral generally follows the K&R C model for floats and string literals *)
```

```
14
15 t letter = ['a'-'z''A'-'Z''_''0'-'9']
16 t number = ['0'-'9']+('.')?['0'-'9']*
17 t stringliteral = ('"'[^'"''\\']*('\\'_[^'"''\\']*)*'"')
18 t digit = ['0'-'9']
19 t exp = ('e'|'E')('+'|'-')?['0'-'9']+
20 t cstylefloat = ('.'['0'-'9']+exp?|['0'-'9']+('.'['0'-'9']*exp?|exp))
21
22 le token = parse
23 | [' ' '\r'] { token lexbuf }
24 | ':' { COLON }
25 | '\t' { TAB }
26 | '\n' { EOL }
27 | "not" { NOT }
28 | "if" { IF }
29 | "else" { ELSE }
30 | "elif" { raise (Failure("NotImplementedError: elif has not been implemented!" )) }
31 | "assert" { raise (Failure("NotImplementedError: assert has not been implemented!" )) }
32 | "pass" { raise (Failure("NotImplementedError: pass has not been implemented!" )) }
33 | "continue" { raise (Failure("NotImplementedError: continue has not been implemented!" )) }
34 | "break" { raise (Failure("NotImplementedError: break has not been implemented!" )) }
35 | "class" { CLASS }
36 | "for" { FOR }
37 | "while" { WHILE }
38 | "def" { DEF }
39 | "int" { INT }
40 | "float" { FLOAT }
41 | "str" { STRING }
42 | "bool" { BOOL }
43 | "func" { FUNC }
44 | "int[]" { INTARR }
45 | "float[]" { FLOATARR }
46 | "str[]" { STRINGARR }
47 | "bool[]" { BOOLARR }
48 | ',' { COMMA }
49 | '.' { DOT }
50 | "!=" { NEQ }
51 | '<' { LT }
52 | '>' { GT }
53 | "<=" { LEQ }
54 | ">=" { GEQ }
55 | "and" { AND }
56 | "or" { OR }
57 | "in" { IN }
58 | "return" { RETURN }
59 | "range" { RANGE }
60 | "is" { IS }
61 | "None" { NONE }
62 | "range" { RANGE }
63 | '#' { comment lexbuf }
64 | '+' { PLUS }
65 | '-' { MINUS }
66 | '*' { TIMES }
67 | '/' { DIVIDE }
```

```
| "**" { EXP }
| "+=" { PLUSEQ }
| "-=" { MINUSEQ }
| "*=" { TIMESEQ }
| "/=" { DIVIDEEQ}
| "**=" { EXPEQ }
| '(' { LPAREN }
| ')' { RPAREN }
| '{' { LBRACE }
| '}' { RBRACE }
| '[' { LBRACK }
| ']' { RBRACK }
| "==" { EQ }
| '=' { ASN }
| ';' { SEP }
| "->" { ARROW }
| "type" { TYPE }
| "print" { PRINT }
| "import" { IMPORT }
| ("global"|"await"|"from"|"as"|"nonlocal"|"async"|"yield"|"raise"|"except"|"finally"|"is"|"lambda"|
| stringliteral as id { STRING_LITERAL(strip_quotes id) }
| ("True"|"False") as id { if id = "True" then BOOL_LITERAL(true) else BOOL_LITERAL(false) }
| cstylefloat as lit { FLOAT_LITERAL(float_of_string lit) }
| ['0'-'9']+ as id { INT_LITERAL(int_of_string id) }
| (['a'-'z''A'-'Z''_']letter*) as id { VARIABLE(id) }
| eof { raise Eof }
| _ as char { raise (Failure("SyntaxError: invalid character in identifier " ^ Char.escaped char)) }

d comment = parse
| '\n' { CEND }
| _ { comment lexbuf }


| "\"\"\"" { multiline lexbuf }
d multiline = parse
| "\"\"\"" { EOL }
| _ { multiline lexbuf }

```

---

semant.ml (Jacob Austin)

```
     open Ast
en Sast
en Utilities

 Semant takes an Abstract Syntax Tree and returns a Syntactically Checked AST with partial type infe
ntax checking, and other features. expr objects are converted to sexpr, and stmt objects are convert
 sstmts. *)

 binop: evaluate types of two binary operations and check if the binary operation is valid.
is currently is quite restrictive and does not permit automatic type casting like in Python.
is may be changed in the future. The commented-out line would allow that feature *)

```

```
t binop t1 t2 op =
let except = (Failure ("STypeError: unsupported operand type(s) for binary " ^ binop_to_string op ^
match (t1, t2) with
| (Dyn, Dyn) | (Dyn, _) | (_, Dyn) -> Dyn
| _ -> let same = t1 = t2 in (match op with
  | Add | Sub | Mul | Exp when same && t1 = Int -> Int
  | Add | Sub | Mul | Div | Exp when same && t1 = Float -> Float
  | Add | Sub | Mul | Div | Exp when same && t1 = Bool -> Bool
  | Add when same && t1 = String -> String
  (* | Add | Sub | Mul | Div | Exp when t1 = Int || t1 = FloatArr || t1 = Bool && t2 = Int || t2 = F
  | Less | Leq | Greater | Geq when not same && t1 = String || t2 = String -> raise except
  | Eq | Neq | Less | Leq | Greater | Geq | And | Or when same -> Bool
  | And | Or when same && t1 = Bool -> Bool
  | Mul when is_arr t1 && t2 = Int -> t1
  | Mul when is_arr t2 && t1 = Int -> t2
  | Mul when t1 = String && t2 = Int -> String
  | Mul when t2 = String && t1 = Int -> String
  | Add when same && is_arr t1 -> t1
  | Div when same && t1 = Int -> Int
  | _ -> raise except
)

 unop: evalues the type of a unary operation to check if it is valid. Currently is less restrictive
an binop and this may need to be modified depending on how codegen is implemented. *)

t unop t1 op = match t1 with
| Dyn -> Dyn
| _ -> (match op with
  | Neg when t1 = Int || t1 = Float || t1 = Bool -> t1
  | Not -> t1
  | _ -> raise (Failure ("STypeError: unsupported operand type for unary " ^ unop_to_string op ^ ":
)

 convert: takes the triple tuple returned by exp (type, expression, data) and converts
 to (type, sexpr, data) *)

t convert (t, e, d) = (t, (e, t), d)

 expr: syntactically check an expression, returning its type, the sexp object, and any relevant data
t rec expr map x = convert (exp map x)

 exp: evaluate expressions, return their types, a partial sast, and any relevant data *)

d exp map = function
| Lit(x) ->
  let typ = match x with
    | IntLit(x) -> Int
    | BoolLit(x) -> Bool
    | StringLit(x) -> String
    | FloatLit(x) -> Float
  in (typ, SLit(x), None) (* convert lit to type, return (type, SLit(x)), check if defined in map *)

| List(x) -> (* parse Lists to determine if they have uniform type, evaluate each expression separat
  let rec aux typ out = function
```

```
67      | [] -> (type_to_array typ, SList(List.rev out, type_to_array typ), None) (* replace Dyn with ty
68      | a :: rest ->
69        let (t, e, _) = expr map a in
70        if t = typ then aux typ (e :: out) rest
71        else aux Dyn (e :: out) rest in
72      (match x with
73        | a :: rest -> let (t, e, _) = expr map a in aux t [e] rest
74        | [] -> (Dyn, SList([], Dyn), None) (* TODO: maybe do something with this special case of empt
75      )
76
77  | ListAccess(e, x) -> (* parse List access to determine the LHS is a list and the RHS is an int if p
78    let (t1, e1, _) = expr map e in
79    let (t2, e2, _) = expr map x in
80    if t1 <> Dyn && not (is_arr t1) || t2 <> Int && t2 <> Dyn
81      then raise (Failure ("STypeError: invalid types for list access"))
82    else (array_to_type t1, SListAccess(e1, e2), None)
83
84  | ListSlice(e, x1, x2) -> (* parse List Slice to determine the LHS is a list and the two RHS element
85    let (t1, e1, _) = expr map e in
86    let (t2, e2, _) = expr map x1 in
87    let (t3, e3, _) = expr map x2 in
88    if t1 <> Dyn && not (is_arr t1) || t2 <> Int && t2 <> Dyn || t3 <> Int && t3 <> Dyn
89      then raise (Failure ("STypeError: invalid types for list access"))
90    else (array_to_type t1, SListSlice(e1, e2, e3), None)
91
92  | Var(Bind(x, t)) -> (* parse Variables, throwing an error if they are not found in the global looku
93    if StringMap.mem x map then
94    let (t', typ, data) = StringMap.find x map in
95    (typ, SVar(x), data)
96    else raise (Failure ("SNameError: name '" ^ x ^ "' is not defined"))
97
98  | Unop(op, e) -> (* parse Unops, making sure the argument and ops have valid type combinations *)
99    let (t1, e', _) = expr map e in
100   let t2 = unop t1 op in (t2, SUnop(op, e'), None)
101
102 | Binop(a, op, b) -> (* parse Binops, making sure the two arguments and ops have valid type combinat
103   let (t1, e1, _) = expr map a in
104   let (t2, e2, _) = expr map b in
105   let t3 = binop t1 t2 op in (t3, SBinop(e1, op, e2), None)
106
107 | Call(exp, args) -> (* parse Call, checking that the LHS is a function, matching arguments and type
108   let (t, e, data) = expr map exp in
109   if t <> Dyn && t <> FuncType
110     then raise (Failure ("STypeError: cannot call objects of type " ^ type_to_string t))
111   else (match data with
112     | Some(x) -> (* if evaluating the expression returns a function *)
113       (match x with
114         | Func(name, formals, body) -> if t <> FuncType && t <> Dyn then (* if that function is full
115           (raise (Failure ("STypeError: cannot call variable"))) else
116           let param_length = List.length args in
117           if List.length formals <> param_length
118           then raise (Failure ("SSyntaxError: unexpected number of arguments in function call"))
119
120           else let rec aux (map, map', bindout, exprout) v1 v2 = match v1, v2 with
```

```
121             | b, e -> let data = expr map e in let (t', e', _) = data in
122               let (map1, bind, _) = assign map' data b in (map, map1, (bind :: bindout), (e' :: expr
123
124           in let map' = StringMap.map (fun (a, b, c) -> (Dyn, b, c)) map (* ignore dynamic types whe
125           in let (_, map1, bindout, exprout) = (List.fold_left2 aux (map, map', [], []) formals args
126
127           let (_, types) = split_sbind bindout in
128           let stack = TypeMap.empty in
129           let stack' = TypeMap.add (x, types) true stack in
130           let (map2, block, data, locals) = (func_stmt map map1 stack' {cond = false; forloop = fals
131
132           (match data with (* match return type with *)
133             | Some (typ2, e', d) -> (* it did return something *)
134                 let Bind(n1, btype) = name in
135                 if btype <> Dyn && btype <> typ2 then if typ2 <> Dyn
136                 then raise (Failure ("STypeError: invalid return type"))
137                 else let func = { styp = btype; sfname = n1; sformals = (List.rev bindout); slocals
138                   (btype, (SCall(e, (List.rev exprout), SFunc(func))), d)
139                 else let func = { styp = typ2; sfname = n1; sformals = (List.rev bindout); slocals =
140                 (typ2, (SCall(e, (List.rev exprout), SFunc(func))), d)
141
142             | None -> (* function didn't return anything, null function *)
143                 let Bind(n1, btype) = name in if btype <> Dyn then
144                 raise (Failure ("STypeError: invalid return type")) else
145                 let func = { styp = Null; sfname = n1; sformals = (List.rev bindout); slocals = loca
146                 (Null, (SCall(e, (List.rev exprout), SFunc(func))), None))
147
148         | _ -> raise (Failure ("SCriticalFailure: unexpected type encountered internally in Call eva
149
150     | None -> print_endline "SWarning: called unknown/undefined function"; (* TODO probably not nece
151         let eout = List.rev (List.fold_left (fun acc e' -> let (_, e', _) = expr map e' in e' :: acc
152         (Dyn, (SCall(e, eout, SNop)), None)
153       )
154
155 | _ as temp -> print_endline ("SNotImplementedError: '" ^ (expr_to_string temp) ^
156     "' semantic checking not implemented"); (Dyn, SNoexpr, None)
157
158  func_expr: checks expressions within functions. differs from expr in how it handles function calls
159
160 d func_expr globals locals stack flag x = convert (func_exp globals locals stack flag x)
161
162 d func_exp globals locals stack flag = function (* evaluate expressions, return types and add to map
163 | Unop(op, e) ->
164   let (t1, e', _) = func_expr globals locals stack flag e in
165   let t2 = unop t1 op in (t2, SUnop(op, e'), None)
166
167 | Binop(a, op, b) ->
168   let (t1, e1, _) = func_expr globals locals stack flag a in
169   let (t2, e2, _) = func_expr globals locals stack flag b in
170   let t3 = binop t1 t2 op in (t3, SBinop(e1, op, e2), None)
171
172 | Var(Bind(x, t)) ->
173   if StringMap.mem x locals then
174   let (typ, t', data) = StringMap.find x locals in
```

```
175   (t', SVar(x), data)
176   else if flag.noeval then (t, SVar(x), None) else
177   raise (Failure ("SNameError: name '" ^ x ^ "' is not defined"))
178
179 | ListAccess(e, x) ->
180   let (t1, e1, _) = func_expr globals locals stack flag e in
181   let (t2, e2, _) = func_expr globals locals stack flag x in
182   if t1 <> Dyn && not (is_arr t1) || t2 <> Int && t2 <> Dyn then raise (Failure ("STypeError: invali
183   else (array_to_type t1, SListAccess(e1, e2), None)
184
185 | ListSlice(e, x1, x2) ->
186   let (t1, e1, _) = func_expr globals locals stack flag e in
187   let (t2, e2, _) = func_expr globals locals stack flag x1 in
188   let (t3, e3, _) = func_expr globals locals stack flag x2 in
189   if t1 <> Dyn && not (is_arr t1) || t2 <> Int && t2 <> Dyn || t3 <> Int && t3 <> Dyn then raise (Fa
190   else (array_to_type t1, SListSlice(e1, e2, e3), None)
191
192 | Call(exp, args) ->
193   let (t, e, data) = func_expr globals locals stack flag exp in
194   if t <> Dyn && t <> FuncType then raise (Failure ("STypeError: cannot call objects of type " ^ typ
195   (match data with (* data is either the Func info *)
196     | Some(x) ->
197       (match x with
198         | Func(name, formals, body) -> if t <> FuncType && t <> Dyn then
199           (raise (Failure ("STypeError: cannot call variable"))) else
200           let param_length = List.length args in
201           if List.length formals <> param_length
202           then raise (Failure ("SSyntaxError: unexpected number of arguments in function call"))
203
204           else let rec aux (globals, locals, bindout, exprout) v1 v2 = match v1, v2 with
205             | b, e -> let data = func_expr globals locals stack flag e in let (t', e', _) = data in
206             let (map1, _, bind2) = assign globals data b in (map1, locals, (bind2 :: bindout), (e' :
207
208           let map' = StringMap.map (fun (a, b, c) -> (Dyn, b, c)) locals in
209           let (map1, _, bindout, exprout) = (List.fold_left2 aux (globals, map', [], []) formals arg
210           let (map'', _, _) = assign map1 (Dyn, (SCall (e, [], SNop), Dyn), data) name in
211
212           let (_, types) = split_sbind bindout in (* avoid recursive calls by checking if the type h
213           if TypeMap.mem (x, types) stack then (Dyn, SCall(e, (List.rev exprout), SNop), None)
214           else let stack' = TypeMap.add (x, types) true stack in
215
216           let (map2, block, data, locals) = (func_stmt globals map'' stack' flag body) in
217           (match data with
218             | Some (typ2, e', d) -> let Bind(n1, btype) = name in if btype <> Dyn && btype <> typ2 t
219                 if typ2 <> Dyn then raise (Failure ("STypeError: invalid return type")) else
220                 let func = { styp = btype; sfname = n1; sformals = (List.rev bindout); slocals = loc
221                 (btype, (SCall(e, (List.rev exprout), SFunc(func))), d) else (* case where definite
222                 let func = { styp = typ2; sfname = n1; sformals = (List.rev bindout); slocals = loca
223                 (typ2, (SCall(e, (List.rev exprout), SFunc(func))), d) (* TODO fix this somehow *)
224
225             | None -> let Bind(n1, btype) = name in if btype <> Dyn
226               then raise (Failure ("STypeError: invalid return type"))
227               else let func = { styp = Null; sfname = n1; sformals = (List.rev bindout); slocals = l
228               (Null, (SCall(e, (List.rev exprout), SFunc(func))), None)) (* TODO fix this somehow *)
```

```
229
230         | _ -> raise (Failure ("SCriticalFailure: unexpected type encountered internally in Call eva
231
232     | None -> if not flag.noeval then print_endline "SNotImplementedError: calling weakly defined fu
233         let eout = List.rev (List.fold_left (fun acc e' -> let (_, e'', _) = func_expr globals local
234         (Dyn, (SCall(e, eout, SNop)), None)
235       )
236
237   | _ as other -> exp locals other
238
239
240  assign: function to check if a certain assignment can be performed with inferred/given types,
241 es assignment if possible, returns two appropriate binds. the return type is:
242 ew map, bind needed for locals list, bind needed for runtime-checking)
243 e second bind will generally be dynamic, except when type inferred cannot determine if the
244 eration is valid and runtime checks must be inserted.
245 p is type inferred type of data being assigned. t' is the explicit type previously assigned to the
246 riable being assigned to. t is the type (optionally) bound to the variable in this assignment.
247
248
249 d assign map data bind =
250 let (typ, _, data) = data in
251 let Bind(n, t) = bind in
252 if StringMap.mem n map then
253 let (t', _, _) = StringMap.find n map in
254   (match typ with
255     | Dyn -> (match (t', t) with (* todo deal with the Bind thing *)
256       | (Dyn, Dyn) -> let map' = StringMap.add n (Dyn, Dyn, data) map in (map', Bind(n, Dyn), Bind(n
257       | (Dyn, _) -> let map' = StringMap.add n (t, t, data) map in (map', Bind(n, t), Bind(n, t)) (*
258       | (_, Dyn) -> let map' = StringMap.add n (t', t', data) map in (map', Bind(n, t'), Bind(n, t')
259       | (_, _) -> let map' = StringMap.add n (t, t, data) map in (map', Bind(n, t), Bind(n, t)))  (*
260     | _ -> (match t' with
261       | Dyn -> (match t with
262         | Dyn -> let map' = StringMap.add n (Dyn, typ, data) map in (map', Bind(n, typ), Bind(n, Dyn
263         | _ when t = typ -> let m' = StringMap.add n (t, t, data) map in (m', Bind(n, t), Bind(n, Dy
264         | _ -> raise (Failure ("STypeError: invalid type assigned to " ^ n)))
265       | _ -> (match t with
266         | Dyn when t' = typ -> let m' = StringMap.add n (t', typ, data) map in (m', Bind(n, t'), Bin
267         | _ when t = typ -> let m' = StringMap.add n (t, t, data) map in (m', Bind(n, t), Bind(n, Dy
268         | _ -> raise (Failure ("STypeError: invalid type assigned to " ^ n)))
269       | _ -> raise (Failure ("STypeError: invalid type assigned to " ^ n))))
270 else if t = typ
271 then let m' = StringMap.add n (t, t, data) map in (m', Bind(n, t), Bind(n, Dyn)) (*  *)
272 else if t = Dyn then let m' = StringMap.add n (Dyn, typ, data) map in (m', Bind(n, typ), Bind(n, Dyn
273 else if typ = Dyn then let m' = StringMap.add n (t, t, data) map in (m', Bind(n, t), Bind(n, t)) (*
274 else raise (Failure ("STypeError: invalid type assigned to " ^ n))
275
276  makes sure an array type can be assigned to a given variable. used for for loops mostly *)
277 d check_array map e b =
278 let (typ, e', data) = expr map e in
279 match typ with
280 | IntArr | FloatArr | BoolArr | StringArr -> assign map (array_to_type typ, e', data) b
281 | Dyn -> assign map (typ, e', data) b (* (t, e', data) *)
282 | _ -> raise (Failure ("STypeError: invalid array type in for loop."))
```

```
283
284
285  check_func: checks an entire function.
286 obals and locals are the globals and locals maps (locals contains all globals).
287 t is a sstmt list containing the semanting checked stmts.
288 ta is a (typ, e', sstmt) tuple containing return information for the function.
289 cal_vars is a list of sbinds containing the local variables.
290 ack is a TypeMap containing the function call stack.
291 DO distinguish between outer and inner scope return statements to stop evaluating when definitely
292 turned. *)
293
294 d check_func globals locals out data local_vars stack flag = (function
295 | [] -> ((List.rev out), data, locals, List.sort_uniq compare (List.rev local_vars))
296 | a :: t -> let (m', value, d, loc) = func_stmt globals locals stack flag a in
297   (match (data, d) with
298     | (None, None) -> check_func globals m' (value :: out) None (loc @ local_vars) stack flag t
299     | (None, _) -> check_func globals m' (value :: out) d (loc @ local_vars) stack flag t
300     | (_, None) -> check_func globals m' (value :: out) data (loc @ local_vars) stack flag t
301     | (_, _) when d = data -> check_func globals m' (value :: out) data (loc @ local_vars) stack fla
302     | _ -> check_func globals m' (value :: out) (Some (Dyn, (SNoexpr, Dyn), None)) (loc @ local_vars
303
304  match_data: when reconciling branches in a conditional branch, this function
305 checks what return types can still be inferred. If both return the same type,
306 that will be preserved. If only one returns, a generic dynamic object will be returned.
307 If both return the same object, that will be preserved. If both return None, that will
308 be returned. Used in for, if, and while statements.
309
310
311 d match_data d1 d2 = match d1, d2 with
312 | (None, None) -> None
313 | (None, _) | (_, None) -> (Some (Dyn, (SNoexpr, Dyn), None))
314 | (Some x, Some y) ->
315   if x = y then d1
316   else let (t1, _, _) = x and (t2, _, _) = y in
317   (Some ((if t1 = t2 then t1 else Dyn), (SNoexpr, Dyn), None))
318
319  func_stmt: syntactically checkts statements inside functions. Exists mostly to handle
320 function calls which recurse and to redirect calls to expr to func_expr. We may be able
321 to simplify the code by merging this with stmt, but it will be challenging to do.
322
323
324 d func_stmt globals locals stack flag = function
325 | Return(e) -> (* for closures, match t with FuncType, attach local scope *)
326   let data = func_expr globals locals stack flag e in
327   let (typ, e', d) = data in
328   (locals, SReturn(e'), (Some data), [])
329
330 | Block(s) ->
331   let (value, data, map', out) = check_func globals locals [] None [] stack flag s in
332   (map', SBlock(value), data, out)
333
334 | Asn(exprs, e) ->
335   let data = func_expr globals locals stack flag e in
336   let (typ, e', d) = data in
```

```
337
338   let rec aux (m, out, binds) = function
339     | [] -> (m, List.rev out, List.rev binds)
340     | Var x :: t ->
341       let Bind (x1, t1) = x in
342       if flag.cond && t1 <> Dyn then
343       raise (Failure ("SSyntaxError: cannot explicitly type variables in conditional branches"))
344       else let (m, b1, b2) = assign locals data x in
345       (aux (m, (SLVar b2) :: out, b1 :: binds) t)
346
347     | ListAccess(e, index) :: t ->
348       let (t1, e1, _) = func_expr globals locals stack flag e in
349       let (t2, e2, _) = func_expr globals locals stack flag index in
350       if t1 <> Dyn && not (is_arr t1) || t2 <> Int && t2 <> Dyn then raise (Failure ("STypeError: in
351       else (aux (m, SLListAccess(e1, e2) :: out, binds) t)
352
353     | ListSlice(e, low, high) :: t ->
354       let (t1, e1, _) = func_expr globals locals stack flag e in
355       let (t2, e2, _) = func_expr globals locals stack flag low in
356       let (t3, e3, _) = func_expr globals locals stack flag high in
357       if t1 <> Dyn && not (is_arr t1) || t2 <> Int && t2 <> Dyn || t3 <> Int && t3 <> Dyn then raise
358       else (aux (m, SLListSlice(e1, e2, e3) :: out, binds) t)
359
360     | Field(a, b) :: t -> raise (Failure "NotImplementedError: Fields are not implemented")
361     | _ -> raise (Failure ("STypeError: invalid types for assignment."))
362
363   in let (m, out, locals) = aux (locals, [], []) exprs in (m, SAsn(out, e'), None, locals)
364
365 | Expr(e) ->
366     let (t, e', data) = func_expr globals locals stack flag e in (locals, SExpr(e'), None, [])
367
368 | Func(a, b, c) ->
369   let rec dups = function (* check duplicate argument names *)
370     | [] -> ()
371     | (Bind(n1, _) :: Bind(n2, _) :: _) when n1 = n2 -> raise (Failure ("SSyntaxError: duplicate arg
372     | _ :: t -> dups t
373   in let _ = dups (List.sort (fun (Bind(a, _)) (Bind(b, _)) -> compare a b) b) in
374
375   let Bind(name, btype) = a in
376
377   (* we assign Bind(name, Dyn) because we want to allow reassignment of functions. this weakens the
378   exchange for reasonable flexibility *)
379
380   let (map', _, _) = assign locals (FuncType, (SNoexpr, FuncType), Some(Func(a, b, c))) (Bind(name,
381   let (semantmap, _, _) = assign StringMap.empty (FuncType, (SNoexpr, FuncType), Some(Func(a, b, c))
382
383   let (map'', bind) = List.fold_left
384       (fun (map, out) (Bind (x, t)) ->
385         let (map', _, b2) = assign map (t, (SNoexpr, t), None) (Bind (x, t)) in
386         (map', b2 :: out)
387       ) (semantmap, []) b in
388
389   let bindout = List.rev bind in
390   let (map2, block, data, locals) = (func_stmt StringMap.empty map'' TypeMap.empty {flag with noeval
```

```
391
392   (match data with
393     | Some (typ2, e', d) ->
394       if btype <> Dyn && btype <> typ2 then if typ2 <> Dyn then
395       raise (Failure ("STypeError: invalid return type"))
396       else let func = { styp = btype; sfname = name; sformals = (List.rev bindout); slocals = locals
397         (map', SFunc(func), None, [Bind(name, FuncType)])
398       else let func = { styp = typ2; sfname = name; sformals = (List.rev bindout); slocals = locals;
399       (map', SFunc(func), None, [Bind(name, FuncType)])
400
401     | None ->
402       if btype <> Dyn then
403       raise (Failure ("STypeError: invalid return type")) else
404       let func = { styp = Null; sfname = name; sformals = (List.rev bindout); slocals = locals; sbod
405       (map', SFunc(func), None, [Bind(name, FuncType)]))
406
407 | If(a, b, c) -> let (typ, e', _) = func_expr globals locals stack flag a in
408       if typ <> Bool && typ <> Dyn then raise (Failure ("STypeError: invalid boolean type in 'if'"))
409       else let (map', value, data, out) = func_stmt globals locals stack {flag with cond = true;} b
410       let (map'', value', data', out') = func_stmt globals locals stack {flag with cond = true;} c i
411       if equals map' map'' then (map', SIf(e', value, value'), match_data data data', out)
412       else let merged = transform map' map'' in
413       let slist = from_sblock value in let slist' = from_sblock value' in
414       (merged, SIf(e', SBlock(slist @ !rec1), SBlock(slist' @ !rec2)), match_data data data', out @
415
416 | For(a, b, c) -> let (m, b1, b2) = check_array locals b a in
417       let (m', x', d, out) = func_stmt globals m stack {flag with cond = true; forloop = true;} c in
418       let (typ, e', _) = func_expr globals m' stack flag b in
419       if equals locals m' then (m', SFor(b2, e', x'), d, b1 :: out)
420       else let merged = transform locals m' in
421       let slist = from_sblock x' in
422       (merged, SFor(b2, e', SBlock(slist @ !rec2)), match_data d None, b1 :: out @ !binds)
423
424 | Range(a, b, c) ->
425       let a1 = Asn([Var a], Lit(IntLit(0))) in
426       let a2 = Asn([Var a], Binop(Var a, Add, Lit(IntLit(1)))) in
427       let a3 = While(Binop(Var a, Less, b), Block(from_block c @ [a2])) in
428       let a4 = If(Binop(b, Greater, Lit(IntLit(0))), Block(a1 :: [a3]), Block([])) in
429       func_stmt globals locals stack flag a4
430
431 | While(a, b) -> let (typ, e, data) = func_expr globals locals stack flag a in
432       if typ <> Bool && typ <> Dyn then raise (Failure ("STypeError: invalid boolean type in 'if'"))
433       else let (m', x', d, out) = func_stmt globals locals stack {flag with cond = true;} b in
434       if equals locals m' then (m', SWhile(e, x'), d, out) else
435       let merged = transform locals m' in
436       let slist = from_sblock x' in
437       (merged, SWhile(e, SBlock(slist @ !rec2)), match_data d None, out @ !binds)
438
439 | Nop -> (locals, SNop, None, [])
440
441 | Type(a) ->  let (t, e, _) = func_expr globals locals stack flag a in
442   print_endline (type_to_string t);
443   (locals, SNop, None, [])
444
```

```
445 | Print(e) -> let (t, e', _) = func_expr globals locals stack flag e in
446   (locals, SPrint(e'), None, [])
447
448 | _ as s -> let (map', value, out) = stmt locals flag s in (map', value, None, [])
449
450  stmt: the regular statement function used for evaluating statements outside of functions. *)
451
452 d stmt map flag = function (* evaluates statements, can pass it a func *)
453 | Asn(exprs, e) ->
454   let data = expr map e in
455   let (typ, e', d) = data in
456
457   let rec aux (m, binds, locals) = function
458     | [] -> (m, List.rev binds, List.rev locals)
459     | Var x :: t ->
460       let Bind (x1, t1) = x in
461       if flag.cond && t1 <> Dyn then
462       raise (Failure ("SSyntaxError: cannot explicitly type variables in conditional branches"))
463       else let (m', b1, b2) = assign m data x in
464       (aux (m', SLVar b2 :: binds, b1 :: locals) t)
465
466     | ListAccess(e, index) :: t ->
467       let (t1, e1, _) = expr map e in
468       let (t2, e2, _) = expr map index in
469       if t1 <> Dyn && not (is_arr t1) || t2 <> Int && t2 <> Dyn then raise (Failure ("STypeError: in
470       else (aux (m, SLListAccess(e1, e2) :: binds, locals) t)
471
472     | ListSlice(e, low, high) :: t ->
473       let (t1, e1, _) = expr map e in
474       let (t2, e2, _) = expr map low in
475       let (t3, e3, _) = expr map high in
476       if t1 <> Dyn && not (is_arr t1) || t2 <> Int && t2 <> Dyn || t3 <> Int && t3 <> Dyn then raise
477       else (aux (m, SLListSlice(e1, e2, e3) :: binds, locals) t)
478
479     | Field(a, b) :: t -> raise (Failure "NotImplementedError: Fields are not implemented")
480     | _ -> raise (Failure ("STypeError: invalid types for assignment."))
481
482   in let (m, binds, locals) = aux (map, [], []) exprs in (m, SAsn(binds, e'), locals)
483
484 | Expr(e) -> let (t, e', _) = expr map e in (map, SExpr(e'), [])
485 | Block(s) -> let ((value, globals), map') = check map [] [] flag s in (map', SBlock(value), globals
486 | Return(e) -> raise (Failure ("SSyntaxError: return statement outside of function"))
487 | Func(a, b, c) -> let rec dups = function (* check duplicate argument names *)
488     | [] -> ()
489     | (Bind(n1, _) :: Bind(n2, _) :: _) when n1 = n2 -> raise (Failure ("SSyntaxError: duplicate arg
490     | _ :: t -> dups t
491   in let _ = dups (List.sort (fun (Bind(a, _)) (Bind(b, _)) -> compare a b) b) in let Bind(name, bty
492
493   let (map', _, _) = assign map (FuncType, (SNoexpr, FuncType), Some(Func(a, b, c))) (Bind(name, Dyn
494   let (semantmap, _, _) = assign StringMap.empty (FuncType, (SNoexpr, FuncType), Some(Func(a, b, c))
495
496   let (map'', binds) = List.fold_left
497     (fun (map, out) (Bind(x, t)) ->
498       let (map', bind, _) = assign map (t, (SNoexpr, t), None) (Bind(x, t)) in
```

```
499       (map', bind :: out)
500     ) (semantmap, []) b in
501
502   let bindout = List.rev binds in
503   let (map2, block, data, locals) = (func_stmt StringMap.empty map'' TypeMap.empty {flag with noeval
504     (match data with
505       | Some (typ2, e', d) ->
506           if btype <> Dyn && btype <> typ2 then if typ2 <> Dyn then
507           raise (Failure ("STypeError: invalid return type")) else
508           let func = { styp = btype; sfname = name; sformals = (List.rev bindout); slocals = locals;
509             (map', SFunc(func), [Bind(name, FuncType)]) else
510             let func = { styp = typ2; sfname = name; sformals = (List.rev bindout); slocals = locals
511           (map', SFunc(func), [Bind(name, FuncType)])
512
513       | None ->
514         if btype <> Dyn then
515         raise (Failure ("STypeError: invalid return type")) else
516         let func = { styp = Null; sfname = name; sformals = (List.rev bindout); slocals = locals; sb
517         (map', SFunc(func), [Bind(name, FuncType)]))
518
519 | If(a, b, c) ->
520   let (typ, e', _) = expr map a in
521   if typ <> Bool && typ <> Dyn then raise (Failure ("STypeError: invalid boolean type in 'if'"))
522   else let (map', value, out) = stmt map {flag with cond = true;} b in
523   let (map'', value', out') = stmt map {flag with cond = true;} c in
524   if equals map' map'' then (map', SIf(e', value, value'), out')
525   else let merged = transform map' map'' in
526   let slist = from_sblock value in let slist' = from_sblock value' in
527   (merged, SIf(e', SBlock(slist @ !rec1), SBlock(slist' @ !rec2)), out @ out' @ !binds)
528
529 | For(a, b, c) ->
530   let (m, b1, b2) = check_array map b a in
531   let (m', x', out) = stmt m {flag with cond = true; forloop = true; } c in
532   let (typ, e', _) = expr m' b in
533   if equals map m' then (m', SFor(b2, e', x'), b1 :: out)
534   else let merged = transform m m' in
535   let slist = from_sblock x' in
536   (merged, SFor(b2, e', SBlock(slist @ !rec2)), b1 :: out @ !binds)
537
538 | Range(a, b, c) ->
539       let a1 = Asn([Var a], Lit(IntLit(0))) in
540       let a2 = Asn([Var a], Binop(Var a, Add, Lit(IntLit(1)))) in
541       let a3 = While(Binop(Var a, Less, b), Block(from_block c @ [a2])) in
542       let a4 = If(Binop(b, Greater, Lit(IntLit(0))), Block(a1 :: [a3]), Block([])) in
543       stmt map flag a4
544
545 | While(a, b) ->
546   let (t, e, _) = expr map a in
547   if t <> Bool && t <> Dyn then raise (Failure ("STypeError: invalid boolean type in 'if'"))
548   else let (m', x', out) = stmt map {flag with cond = true; }b in
549   if equals map m' then (m', SWhile(e, x'), out)
550   else let merged = transform map m' in
551   let slist = from_sblock x' in
552   (merged, SWhile(e, SBlock(slist @ !rec2)), out @ !binds)
```

```
553
554 | Nop -> (map, SNop, [])
555 | Print(e) -> let (t, e', _) = expr map e in (map, SPrint(e'), [])
556 | Type(a) ->
557   let (t, e, _) = expr map a in
558   print_endline (type_to_string t);
559   (map, SNop, [])
560
561 | _ as temp ->
562   print_endline ("NotImplementedError: '" ^ (stmt_to_string temp) ^ "' semantic checking not impleme
563
564  check: master function to check the entire program by iterating over the list of
565 atements and returning a list of sstmts, a list of globals, and the updated map *)
566
567 d check map out globals flag = function
568 | [] -> ((List.rev out, List.sort_uniq compare (List.rev globals)), map)
569 | a :: t -> let (m', value, g) = stmt map flag a in check m' (value :: out) (g @ globals) flag t
```

---

utilities.ml

```
1       open Ast
2 en Sast
3
4  Runtime: exception type used for the interpreter *)
5 ception Runtime of string
6
7  float_of_bool: converts floats to bools, so we can hack in support for if statements. 1.0 is true,
8 t float_of_bool b = if b then 1.0 else 0.0
9
10  zip: combine two lists, used to match the argument names and their values *)
11 t rec zip lst1 lst2 = match lst1,lst2 with
12 | [], [] -> []
13 | [],_::_-> raise (Runtime "TypeError: invalid arguments passed to function!")
14 | _::_, []-> raise (Runtime "TypeError: invalid arguments passed to function!")
15 | (x::xs),(y::ys) -> (x, y) :: (zip xs ys)
16
17  explode: converts a string to a list of chars *)
18 t explode s =
19 let rec exp i l =
20   if i < 0 then l else exp (i - 1) (s.[i] :: l) in
21 exp (String.length s - 1) []
22
23
24  implode: does the reverse of explode, i.e. converts a list of chars to a string *)
25 t implode s = String.concat "" (List.map (String.make 1) s)
26
27  print: utility function used for printing parsed tokens. can be replaced by menhir mostly. not exha
28 t print = function
29 | Parser.COLON -> "COLON"
30 | Parser.TAB -> "TAB"
31 | Parser.NOT -> "NOT"
32 | Parser.IF -> "IF"
33 | Parser.ELSE -> "ELSE"
```

```
| Parser.FOR -> "FOR"
| Parser.WHILE -> "WHILE"
| Parser.DEF -> "DEF"
| Parser.COMMA -> "COMMA"
| Parser.NEQ -> "NEQ"
| Parser.LT -> "LT"
| Parser.GT -> "GT"
| Parser.LEQ -> "LEQ"
| Parser.GEQ -> "GEQ"
| Parser.AND -> "AND"
| Parser.OR -> "OR"
| Parser.IN -> "IN"
| Parser.TRUE -> "TRUE"
| Parser.FALSE -> "FALSE"
| Parser.IS -> "IS"
| Parser.PLUS -> "PLUS"
| Parser.MINUS -> "MINUS"
| Parser.TIMES -> "TIMES"
| Parser.DIVIDE -> "DIVIDE"
| Parser.EXP -> "EXP"
| Parser.RETURN -> "RETURN"
| Parser.LPAREN -> "LPAREN"
| Parser.RPAREN -> "RPAREN"
| Parser.LBRACK -> "LBRACK"
| Parser.RBRACK -> "RBRACK"
| Parser.EQ -> "EQ"
| Parser.ASN -> "ASN"
| Parser.CLASS -> "CLASS"
| Parser.SEP -> "SEP"
| Parser.EOF -> "EOF"
| Parser.EOL -> "EOL"
| Parser.DOT -> "DOT"
| Parser.INDENT -> "INDENT"
| Parser.DEDENT -> "DEDENT"
| Parser.VARIABLE(_) -> "VARIABLE" (*Printf.sprintf "Var(%s)" x *)
| Parser.STRING_LITERAL(x) -> Printf.sprintf "STRING_LITERAL(%s)" x
| Parser.FLOAT_LITERAL(_) -> "FLOAT_LITERAL" (*Printf.sprintf "Lit(%f)" x *)
| Parser.BOOL_LITERAL(_) -> "BOOL_ITERAL" (*Printf.sprintf "Lit(%f)" x *)
| Parser.INT_LITERAL(_) -> "INT_LITERAL" (*Printf.sprintf "Lit(%f)" x *)
| Parser.INT -> "INT"
| Parser.BOOL -> "BOOL"
| Parser.FLOAT -> "FLOAT"
| Parser.STRING -> "STRING"
| Parser.INTARR -> "INTARR"
| Parser.FLOATARR -> "FLOATARR"
| Parser.STRINGARR -> "STRINGARR"
| Parser.BOOLARR -> "BOOLARR"
| Parser.NOP -> "NOP"
| Parser.TYPE -> "TYPE"
| Parser.PLUSEQ -> "PLUSEQ"
| Parser.MINUSEQ -> "MINUSEQ"
| Parser.TIMESEQ -> "TIMESEQ"
| Parser.DIVIDEEQ -> "DIVIDEEQ"
| Parser.EXPEQ -> "EXPEQ"
```

```
88  | Parser.PRINT -> "PRINT"
89  | Parser.IMPORT -> "IMPORT"
90  | Parser.RANGE -> "RANGE"
91  | Parser.NOELSE -> "NOELSE"
92  | Parser.NEG -> "NEG"
93  | Parser.ARROW -> "ARROW"
94  | Parser.FUNC -> "FUNC"
95  | Parser.NONE -> "NONE"
96  | Parser.CEND -> "CEND"
97  | Parser.LBRACE -> "LBRACE"
98  | Parser.RBRACE -> "RBRACE"
99
100  stmt_to_string converts stmt to string for error handling *)
101 t stmt_to_string = function
102 | Func(_, _, _) -> "func"
103 | Block(_) -> "block"
104 | Expr(_) -> "expr"
105 | If(_, _, _) -> "if"
106 | For(_, _, _) -> "for"
107 | Range(_, _, _) -> "range"
108 | While(_, _) -> "while"
109 | Return(_) -> "return"
110 | Class(_, _) -> "class"
111 | Asn(_, _) -> "asn"
112 | Type(_) -> "type"
113 | Print(_) -> "print"
114 | Import(_) -> "import"
115 | Nop -> "nop"
116
117  expr_to_string: converts expr to string for error handling *)
118 t expr_to_string = function
119 | Binop(_, _, _) -> "binop"
120 | Lit(_) -> "lit"
121 | Var(_) -> "var"
122 | Unop(_, _) -> "unop"
123 | Call(_, _) -> "call"
124 | Method(_, _, _) -> "method"
125 | Field(_, _) -> "field"
126 | List(_) -> "list"
127 | ListAccess(_, _) -> "list access"
128 | ListSlice(_, _, _) -> "list slice"
129
130  type_to_string converts type to string for error handling *)
131 t type_to_string = function
132 | Dyn -> "dyn"
133 | Int -> "int"
134 | Float -> "float"
135 | Bool -> "bool"
136 | String -> "string"
137 | IntArr -> "int list"
138 | FloatArr -> "float list"
139 | BoolArr -> "bool list"
140 | StringArr -> "string list"
141 | FuncType -> "func"
```

```
| Null -> "null"

 unop_to_string: converts unop to string for error handling *)
t unop_to_string = function
| Neg -> "-"
| Not -> "not"

 binop_to_string converts binop to string for error handling *)
t binop_to_string = function
| Add -> "+"
| Sub -> "-"
| Mul -> "*"
| Div -> "/"
| Exp -> "**"
| Eq -> "=="
| Neq -> "!="
| Less -> "<"
| Leq -> "<="
| Greater -> ">"
| Geq -> ">="
| And -> "and"
| Or -> "or"
| ListAccess -> "at index"

 type_to_array converts type to corresponding array type for array handling *)
t type_to_array = function
| Int -> IntArr
| Bool -> BoolArr
| String -> StringArr
| Float -> FloatArr
| _ as x -> x

 array_to_type converts array types to their corresponding non-array types *)
t array_to_type = function
| IntArr -> Int
| BoolArr -> Bool
| FloatArr -> Float
| StringArr -> String
| _ as x -> x

 is_arr checks if a given type is an array type. Strings are currently treated as arrays *)
t is_arr = function
| StringArr | BoolArr | IntArr | FloatArr -> true
| String -> true
| _ -> false

 splits a list of sbinds into a list of strings and a list of types, uses for TypeMap/recursion *)
t split_sbind x =
let rec aux a1 a2 = function
  | [] -> (List.rev a1, List.rev a2)
  | a :: t -> match a with
    | Bind(c, d) -> aux (c :: a1) (d :: a2) t
    | _ -> raise (Failure "unknown failure in argument matching")
in aux [] [] x
```

```
 compare two lists for use in maps with list keys *)
t comp x y =  match List.length x, List.length y with
| a, b when a < b -> -1
| a, b when a > b -> 1
| a, b when a = b -> let rec aux = function
  | [], [] -> 0
  | x :: t, y :: q when x = y -> aux (t, q)
  | x :: t, y :: q when x <> y -> compare x y
  | _ -> raise (Failure "unknown failure in argument matching")
in aux (x, y)
| _ -> raise (Failure "unknown failure in argument matching")


 merge function used to compare and combine two maps for type inference. handles scoping and undefin
t compare_types a b = if a = b then a else Dyn
t compare_decl a b = if a = b then a else false
t compare_data a b = if a = b then a else None

 map with SFunc and argument type list used to check recursive calls *)
dule TypeMap = Map.Make(struct type t = stmt * typ list let compare = Pervasives.compare end);;

 map with string keys, used for variable lookup *)
dule StringMap = Map.Make(String)

 merge: merge function used to reconcile the global lookup map after a conditional branch. *)
t merge m1 m2 = StringMap.merge (fun key v1 v2 -> match v1, v2 with (* merge two lists while keeping
  | Some (a, b, c), Some (d, e, f) -> Some (compare_types a d, compare_types b e, compare_data c f)
  | Some (a, b, c), None -> Some(Dyn, Dyn, c)
  | None, Some(a, b, c) -> Some(Dyn, Dyn, c)
  | None, None -> None
) m1 m2

t rec1 = ref [] (* these are used to extract Transform objects for use in codegen from merge *)
t rec2 = ref []

t binds = ref []

 transform: merge function used to reconcile the global lookup map after a conditional branch.
tracts objects with transformed type for use in codegen. *)

t transform m1 m2 = rec1 := []; rec2 := []; binds := []; StringMap.merge (fun key v1 v2 -> match v1,
  | Some (a, b, c), Some (d, e, f) ->
      let t = compare_types b e in
      let () = if b <> t then (rec1 := ((STransform(key, b, Dyn) :: !rec1)); binds := ((Bind(key, Dy
      let () = if e <> t then (rec2 := ((STransform(key, e, Dyn)) :: !rec2); binds := ((Bind(key, Dy
      Some (compare_types a d, compare_types b e, compare_data c f)

  | Some (a, b, c), None -> let () = if b <> Dyn then (rec1 := (STransform(key, b, Dyn) :: !rec1); b
  | None, Some(a, b, c) -> let () = if b <> Dyn then (rec2 := (STransform(key, b, Dyn) :: !rec2); bi
  | None, None -> None
) m1 m2

```

```
 from_block: used to extract the slist from an SBlock in codegen *)
t from_block block = match block with
| Block(x) -> x
| _ -> raise (Failure ("SCriticalFailure: unexpected type encountered internally in branch evaluatio

 from_sblock: used to extract the slist from an SBlock in codegen *)
t from_sblock block = match block with
| SBlock(x) -> x
| _ -> raise (Failure ("SCriticalFailure: unexpected type encountered internally in branch evaluatio


 check if two maps are equal *)
t equals m1 m2 = (StringMap.equal (fun x y -> (compare x y) = 0) m1 m2) (* check if two maps are equ
```

8.3 test Files

sfail-func1

```
def foo(x):
        def bar(y):
                return y
        return bar

y = foo(3)
y()
```

sfail-func2

```
foo : str = "hello"

def foo(x):
        return x
```

sfail-func3

```
x : str = "hello"

def foo(arr : str):
        arr = 3.0
        return arr
```

sfail-func4

```
def foo(x):
        return x + y

def bar(y):
        foo(y)

bar(3)
```

sfail-func5

```
def foo(y):
        return bez(y)

def bez(x):
        return x + y

def bar(y):
        foo(y)

bar(3)
```

sfail-list3

```
def add(x : int[]):
        acc = 0
        for i in acc:
                acc = acc + i
        return acc

type(add([1, 2, 3]))
```

sfail-type1

```
def foo(x : str, y : int):
        return x + y

x = 4.0
y = 4

foo(x, y)

```

sfail-type2

```
def foo(x : float, y : float):
        return x + y

x = 4
y = 4.0

foo(x, y)

```

sfail-type3

```
def foo(x, y):
        return x + y

foo(3, 3.0)
```

stest-add1

```
x = 4
y = 10

x + y
```

stest-comment1

```
x = 3 # single line comments
type(x) # int

# it turns out multiline comments are hard.

def foo():
        x = 3
#       x = 5
        x = 4
```

stest-exper1

```
x = 3

if x:
        type(x)
```

stest-exper2

```
x = 3

if x == 3:
        y : int = 4
```

stest-func

```
def foo(x, y):
        return x + y

x = 3
y = 4

foo(x, y)
```

stest-func10

```
l = [1, 0, 5, 12]
a = l[2]

for i in l:
        type(i)

type(a)

def foo():
        if x == 3:
                return x
        else:
                return 4.0

```

stest-func11

```
def foo(x : int[]) -> float:
        def bar(x : int[]) -> int:
                return 3
        type(bar([4, 2])) # int
        def bar():
                return 3.0
        type(bar()) # float
        def bar(x) -> float:
                return 3.0
        bar("hello") # float
        type(bar) # func
        bar("hello")
        return 3.0

type(foo([1, 2, 3])) # float

```

stest-func2

```
def foo(x : str, y : int):
        return x

x = "hello"
y = 4

foo(x, y)

```

stest-func3

```
def foo(x):
        def bar(y):
                return y
        return bar

y = foo(3)
y(4)
```

stest-func4

```
def foo(a, b):
        return a

def call(f):
        k = 66
        z = f(k, 1)
        return z

result = call(foo)
type(result)

```

stest-func5

```
def foo():
        def bar(x):
                return x
        return bar

type(foo()(3))
```

stest-func6

```
def add(x):
        if x == 3:
                return 3
        else:
                return 4

type(add(3))

```

stest-func7

```
def add(x):
        if x == 3:
                return 3
        else:
                return 3.0

type(add(3))

```

stest-func8

```
def add(x):
        if x == 3:
                return 3

        return 3.0

type(add(3))

```

stest-func9

```
def foo(x, y):
        x = 3.0
        return x + y

type(foo("hi", 4.0)) # float

def bar(x : int[]):
        acc = 0
        for i in x:
                acc += i
        return acc

type(bar([1, 2, 3, 4])) # int

def gcd(a : int, b : int) -> int:
        while a != b:
                if a > b:
                        a = a - b
                else:
                        b = b - a
        return a

type(gcd(534, 2534)) # int


def gcd(a, b):
        while a != b:
                if a > b:
                        a = a - b
                else:
                        b = b - a
        return a

type(gcd(534, 2534)) # int
type(gcd(534.0, 2534.0)) # float
```

stest-gcd

```
a=1234342213
b=334232

while a != b:
        if a > b:
                a = a - b
        else:
                b = b - a

```

stest-globals1

```
x = 3

def foo():
        a = x
        return a

type(foo())
x = "hello"
type(foo())
```

stest-globals2

```
 x : int = 3

 def foo():
         a = x
         x = "hello"
         return a

 type(foo())

 x : str = "hello"

 type(foo())
```

stest-import1

```
import stestlib

type(gcd(5, 4))
```

stest-list1

```
x = [1, 2, 3]
type(x)
type(x[2])

x = [1, 1.0, 2]
type(x)
type(x[2])
```

stest-list2

```
def add(x : int[]):
        acc = 0
        for i in x:
                acc = acc + i
        return acc

type(add([1, 2, 3]))
```

stest-range1

```
for i in range(10):
        x = 3

type(i)
type(x)
```

stest-recurse1

```
def foo(n):
        if n <= 1:
                return n
        else:
                return foo(n - 1) + foo(n - 2)

type(foo(3))
```

stest-recurse2

```
def foo(x):
        def bar(x):
                if x == 0:
                        return x
                else:
                        return bar(x - 1)
        return bar(x)

type(foo(5))
```

stest-type1

```
def gcd(a, b):
        while a != b:
                if a > b:
                        a = a - b
                else:
                        b = b - a
        return a

y = gcd(100253, 135053)
type(y)
```

stest-type2

```
def concat(a, b):
        return a + b

a = ""
for i in [1, 2, 3]:
        a = concat(a, a * i)

type(a)
```

stest-type3

```
z = 3
z = 2.

if False:
        x = 4
else:
        def x():
                return 1

type(x+1)
```

stest-type4

```
x = 3
type(x)

x = "hello"
type(x)

x = 3.0
type(x)

x = True
type(x)

x = False
type(x)

x : str = "hello"
type(x)

x : float = 3.0
type(x)

x : bool = True
type(x)

x : bool = False
type(x)

x : str = ""
type(x)
```

stestlib

```
def gcd(a : int, b : int) -> int:
        while a != b:
                if a > b:
                        a = a - b
                else:
                        b = b - a

        return a

```

test-add1

```
x = 4
y = 10
print (x + y + 6)
```

test-bool

```
if True:
        print(1)
if True == False:
        print(0)
x = 3
if x == 3:
        print(1)
if x != 3:
        print(0)

```

test-forloops

```
l = [7, 9, 5]

x = "b"

for i in l:
        a = i
        x = i
        print(a)
        print(x)
        print(i)

print(a)
print(i)
print(x)
```

test-func1

```
def foo():
        return 8


print(foo())

```

test-func2

```


def one_please():
        return 1

def call_it(f):
        return f()

z = call_it(one_please)
print(z)

```

test-func3

```

def cond(a):
        if a>0:
                print(a)
        else:
                print(-1)
        return 0

cond(10)
cond(-10)

```

test-func4

```

def no_return():
        print(70)


no_return()


```

test-func5

```
def foo():
        def bar():
                return 4
        return bar

a = foo()
b = a()
print(b)

```

test-func6

```
def sum(a,b):
        return a+b

def one():
        return 1


def do_wild_things(f,a,b):
        return (f(a,b) + f(a,b)) * f(a,b)

z = do_wild_things(sum,2*one(),4)


print(z)

```

test-func7

```
def has_locals(a):
        n = 5-a
        return n


print(has_locals(2))

```

test-gcd

```
a=1234342213
b=334232

while a != b:
        if a > b:
                a = a-b
        else:
                b = b-a

print(a)
```

test-idx

```
l = [1, 2, 3, 4]
a = l[2]

print(a)
```

test-str

```
l = "abc"
a = l[1]

print(a)
```

9 Git Logs

```
commit ddfd28311b0a920db78ecb89c44608f93a3768d8
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Wed Dec 19 23:27:03 2018 -0500

    disabled codegen debug messages

 src/codegenutils.ml | 16 ++++++++--------
 1 file changed, 8 insertions(+), 8 deletions(-)

commit bb3ee2ad8c95364edce035043247ce3e837db154
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Dec 19 23:14:34 2018 -0500

    refactored the code

 Makefile     |  4 ++--
 src/coral.ml | 13 ++++++------
 testall.sh   |  4 ++--
 3 files changed, 11 insertions(+), 10 deletions(-)

commit 5988a60aded7012a2c871821f006063f8096d5de
Merge: 294d024 f87e3f4
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Dec 19 23:08:24 2018 -0500

    Merge branch 'optimization' of https://github.com/ja3067/Coral into optimization

commit 294d0244bafc05238a954b36ca78acad0c12d563
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Dec 19 23:08:21 2018 -0500

    merged

 llvm-test.cl          | 43 -----------------------------------------
```

```
 src/codegen.ml        |  2 +-
 src/coral.ml          |  3 +--
 src/semant.ml         |  8 ++++----
 tests/test-func10.out |  3 +--
 tests/test-func11.out |  3 +--
 tests/test-func12.out |  2 +-
 tests/test-func9.out  |  3 +--
 8 files changed, 10 insertions(+), 57 deletions(-)

commit f87e3f4a0b74b8d97abe02e207112804b6421561
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Wed Dec 19 23:07:58 2018 -0500

    more functions

 src/codegen.ml      | 28 ++++++++++++++++-----------
 src/codegenutils.ml |  2 ++
 2 files changed, 19 insertions(+), 11 deletions(-)

commit 521e0f29c4c7e5eebb466a45dcca7868c271334d
Merge: ea57cc9 c04accb
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Dec 19 22:22:45 2018 -0500

    merged

commit ea57cc93b364c2272edf79da60d0225db6d6104f
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Wed Dec 19 22:20:29 2018 -0500

    fixed bug

 src/codegen.ml | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit cb89934befc39ad107233a60511b2e08398c816f
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Wed Dec 19 22:19:25 2018 -0500

    worked on generic funcitons

 src/codegen.ml      | 299 ++++++++++++++++++++++++-----------------------------
 src/codegenutils.ml |   2 +
 2 files changed, 142 insertions(+), 159 deletions(-)

commit c04accb58e699ee8d168347f97a94a9c57a81206
Merge: e4eaaf6 3568121
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Dec 19 22:18:21 2018 -0500

    Merge branch 'SAST' into llvm

commit 35681212addf97a392a65c54cc260645c19c9998
Author: Jacob Austin <ja3067@columbia.edu>
```

```
Date:   Wed Dec 19 22:18:13 2018 -0500

    made weak functions illegal

 src/semant.ml | 9 ++++++---
 1 file changed, 6 insertions(+), 3 deletions(-)

commit e4eaaf6c08c21d7ec62a3fdcff9e4c7a02620749
Merge: 607cd90 8d2a739
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Dec 19 17:08:12 2018 -0500

    Merge branch 'SAST' into llvm

commit 8d2a739c334c8ad42009641925f567396ac16782
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Dec 19 17:07:48 2018 -0500

    updated tests to reflect reduced type inference on lists

 tests/stest-func10.out | 4 ++--
 tests/stest-list1.out  | 4 ++--
 tests/stest-type2.out  | 2 +-
 3 files changed, 5 insertions(+), 5 deletions(-)

commit 10cf20f666125fd64d0a75a5c0ab679505b78fbb
Merge: 56b1a92 a43e5bc
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Dec 19 17:04:48 2018 -0500

    fixed merge bug

commit 56b1a9262d3b99584b2b0b63b4a806e88aa95614
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Dec 19 17:01:02 2018 -0500

    updated to new lvalue syntax

 src/codegen.ml | 6 +++++-
 1 file changed, 5 insertions(+), 1 deletion(-)

commit d02823f74d28a98ad639959c125e15313bfcad3a
Merge: bbd96ba 3a8eaf4
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Dec 19 16:59:15 2018 -0500

    Merge branch 'SAST' into optimization

commit a43e5bc3b36f93f9dc81b65c6bf4c14be13b5125
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Wed Dec 19 16:57:19 2018 -0500

    refactoring / prettification
```

```
 src/codegen.ml      | 109 +++++++++++++++++----------------------------------
 src/codegenutils.ml |  77 +++++++++++++++++++++++++++++++++++++
 2 files changed, 114 insertions(+), 72 deletions(-)

commit 607cd90173d4805566f61677cb472b251f78e332
Merge: e7b85bf 3a8eaf4
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Dec 19 16:53:44 2018 -0500

    merged with SAST, fixed bug in codegen

commit 3a8eaf457228294472d24b39545b605cc322a6e5
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Dec 19 16:48:05 2018 -0500

    updated lvalue behavior

 .gitignore     |  3 ++-
 src/parser.mly |  2 +-
 src/sast.ml    | 18 +++++++++++---
 src/semant.ml  | 83 +++++++++++++++++++++++++++++++++++---------------------
 4 files changed, 68 insertions(+), 38 deletions(-)

commit 2b24041bd94c29a325d67f6586357e095f2b7c17
Merge: bbd96ba 242fc1f
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Wed Dec 19 15:49:15 2018 -0500

    Merge branch 'exceptions2' into optimization

commit bbd96ba180bf8f29c2a1baa4265551578789fca4
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Dec 19 12:28:02 2018 -0500

    added many more tests, fixed parser indentation for files

 src/coral.ml          | 18 +++++++++++++++++-
 tests/test-func10.cl  | 33 +++++++++++++++++++++++++++++++++
 tests/test-func10.out |  2 ++
 tests/test-func11.cl  | 29 +++++++++++++++++++++++++++++
 tests/test-func11.out |  2 ++
 tests/test-func12.cl  | 12 ++++++++++++
 tests/test-func12.out |  1 +
 tests/test-func8.cl   | 16 ++++++++++++++++
 tests/test-func8.out  |  1 +
 tests/test-func9.cl   | 22 ++++++++++++++++++++++
 tests/test-func9.out  |  2 ++
 11 files changed, 137 insertions(+), 1 deletion(-)

commit 242fc1f68b51eff041bd5cccbf27167513a7c9b2
Merge: d6a3e23 e85e65e
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Wed Dec 19 04:36:34 2018 -0500
```

```
    assignment exceptions

commit e7b85bfb4f33c5bc1743aa19581a4e341006245d
Merge: 85dfeff 3bcf5c3
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Dec 19 01:34:20 2018 -0500

    Merge branch 'SAST' into llvm

commit 3bcf5c366925b668da724dda7abad3acd03aa5c6
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Dec 19 01:34:06 2018 -0500

    fixed bug with valid identifiers

 src/ast.ml      |  1 +
 src/scanner.mll |  2 +-
 src/semant.ml   | 18 ++++++++++++++--
 3 files changed, 18 insertions(+), 3 deletions(-)

commit 85dfefff59ec9a109892f9460d1e168e24b9053a
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Dec 19 00:08:58 2018 -0500

    small fixed

 src/semant.ml | 6 +++---
 1 file changed, 3 insertions(+), 3 deletions(-)

commit 57df14148716a47c7033bac1ac4e84ad005240b2
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Tue Dec 18 22:39:30 2018 -0500

    updated range handling

 src/coral.ml  | 40 ++++++------------------------------
 src/semant.ml | 18 ++++++++++++++--
 2 files changed, 22 insertions(+), 36 deletions(-)

commit e85e65e5edd2c47016286145f6b2e0c43f200a02
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Tue Dec 18 22:20:19 2018 -0500

    assignment exceptions

 src/codegen.ml         | 61 ++++++++++++++++++++++++++++++++++++++++++++++++++
 tests/fail-badasn1.cl  | 13 ++++++++++
 tests/fail-badasn1.out |  0
 tests/fail-badasn2.cl  | 11 +++++++++
 tests/fail-badasn2.out |  0
 5 files changed, 83 insertions(+), 2 deletions(-)

commit d6a3e23d6128dce555b3e5f181972a249d4f9c5d
Author: Jacob Austin <ja3067@columbia.edu>
```

```
Date:   Tue Dec 18 21:39:48 2018 -0500

    refactored Range handling to avoid extra preprocessing function

 src/coral.ml  | 31 +++----------------------------
 src/semant.ml | 14 ++++++++++++++
 2 files changed, 17 insertions(+), 28 deletions(-)

commit dafc01109a79efe8d85021895b67f71fb6bc181a
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Tue Dec 18 13:37:00 2018 -0500

    working merge

 src/semant.ml | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)

commit 46ec3246b63af432cecdc08b2ebe04c2620b6cf4
Merge: 99be831 d032acd
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Tue Dec 18 13:32:05 2018 -0500

    Merge branch 'llvm' into optimization

commit 99be831144d94a249d2cd424443b62504e6899df
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Tue Dec 18 13:31:58 2018 -0500

    fixed bug

 src/coral.ml | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)

commit df6b3e3febabe29e8e2510e0b81a09ffe8b153f8
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Tue Dec 18 11:29:36 2018 -0500

    unops work

 src/codegen.ml | 105 ++++++++++++++++++++++++++++++++++++++-------------------
 1 file changed, 70 insertions(+), 35 deletions(-)

commit 292c85671e2646226923a2b7a009642691cfb48e
Merge: 60238ef cc75261
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Tue Dec 18 10:39:43 2018 -0500

    MERGED exceptions into optimization

commit 60238ef62000ccae92e5af27647a33da203fa515
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Tue Dec 18 09:30:45 2018 -0500

    removed commented code
```

```
 src/codegen.ml | 206 ----------------------------------------------------------
 1 file changed, 206 deletions(-)

commit e392097e20cc84bb42000fff82572791d5ee2859
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Tue Dec 18 09:13:06 2018 -0500

    first class funcs done other than func6

 llvm-test.cl   |  19 ++-
 src/codegen.ml | 391 ++++++++++++++++++++++++++++++++++++++-----------------
 2 files changed, 293 insertions(+), 117 deletions(-)

commit d032acdba33fe7bdb2db23c684f602106b6eb6aa
Merge: 2897033 ffdd347
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 22:51:58 2018 -0500

    merged with SAST

commit ffdd3474ab36327b318382e17d66a529b4318745
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 22:50:17 2018 -0500

    fixed behavior of Range statement

 src/ast.ml             |  1 +
 src/coral.ml           | 30 ++++++++++++++++++++++++++--
 src/parser.mly         |  4 +++-
 src/scanner.mll        |  1 +
 src/utilities.ml       | 11 ++++++++++-
 tests/stest-range1.cl  |  5 +++++
 tests/stest-range1.out |  2 ++
 7 files changed, 50 insertions(+), 4 deletions(-)

commit 2897033f75b39b98ac5656c8c2cf5fa50e504d9b
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 22:35:25 2018 -0500

    small fix for range

 src/coral.ml | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)

commit 48fc11ac14bc018e3a0ae640f012dc96de864137
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 22:33:28 2018 -0500

    updated for loops

 src/coral.ml | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

```
commit 43c3c8f299731118376d444b6c714c2a76b523fb
Merge: 10be8ea 58dddb1
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 22:29:39 2018 -0500

    Merge branch 'llvm' into optimization

commit 58dddb1bfaa88fe80abce3f489d9075d7ac359dc
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 22:29:27 2018 -0500

    added for loops as syntactic sugar

 src/ast.ml       |  1 +
 src/coral.ml     | 23 +++++++++++++++++++++--
 src/parser.mly   |  4 +++-
 src/scanner.mll  |  1 +
 src/utilities.ml |  2 ++
 5 files changed, 28 insertions(+), 3 deletions(-)

commit 10be8ea87197d053de4f00963de81e9c71132dd7
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Mon Dec 17 22:29:12 2018 -0500

    now any mix of raw+immutables work with binops

 llvm-test.cl   | 15 +++++++++--
 src/codegen.ml | 80 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-------------------
 2 files changed, 73 insertions(+), 22 deletions(-)

commit 3cf429baf4338138e0255e2c0ac97b2b358f10d0
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Mon Dec 17 21:22:12 2018 -0500

    switched to Box=cobj_pt, and added prints

 llvm-test.cl   |  29 +++++--------
 src/codegen.ml | 128 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++-------------------
 src/sast.ml    |   3 +-
 3 files changed, 107 insertions(+), 53 deletions(-)

commit c85d978c71f62b6fab3eeeb8f69cb42a8a70164c
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 18:57:12 2018 -0500

    added func type

 src/parser.mly  | 4 +++-
 src/scanner.mll | 2 ++
 2 files changed, 5 insertions(+), 1 deletion(-)

commit 36008476fb7d1b68944cc4794a2c9f576387a575
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 17:16:14 2018 -0500
```

```
    fixed binds

 src/sast.ml   | 1 -
 src/semant.ml | 4 ++--
 2 files changed, 2 insertions(+), 3 deletions(-)

commit 42ef9501a1df5fb021869aba1d19fdfe409c1e9e
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 17:07:30 2018 -0500

    fixed bugs

 src/utilities.ml | 10 +++++-----
 1 file changed, 5 insertions(+), 5 deletions(-)

commit eb4e7c70ea1939d5b968cfc45616681909ca4fc4
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 17:06:26 2018 -0500

    fixed bugs

 src/utilities.ml | 8 ++++----
 1 file changed, 4 insertions(+), 4 deletions(-)

commit 1da26bfdb7eecd194efa88e2894b0e7ae13ee93b
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 17:05:49 2018 -0500

    fixed bugs

 src/utilities.ml | 8 ++++----
 1 file changed, 4 insertions(+), 4 deletions(-)

commit 3727c78a58c623be83e7d7ede7e154f1c02c5da8
Merge: a432bf9 c611f5c
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 16:54:25 2018 -0500

    Merge branch 'llvm' into optimization

commit c611f5c94dd8b3a8a1d7136b938d4d6f74a5b6e0
Merge: db1664c d0bed67
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 16:53:50 2018 -0500

    Merge branch 'SAST' into llvm

commit d0bed67640658aa42f526f0c6960807864ddb856
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 16:53:40 2018 -0500

    fixed gitignore
```

```
 .gitignore |    2 +
 main       |  Bin 15040 -> 0 bytes
 source.ll  | 1559 ---------------------------------------------
 source.s   | 2020 ------------------------------------------------------------
 4 files changed, 2 insertions(+), 3579 deletions(-)

commit 352d23caaa44c90a94b8db486ac7469d253cf36a
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 16:53:21 2018 -0500

    fixed a bug

 main             |  Bin 0 -> 15040 bytes
 source.ll        | 1559 +++++++++++++++++++++++++++++++++++++++++
 source.s         | 2020 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 src/utilities.ml |    2 +-
 4 files changed, 3580 insertions(+), 1 deletion(-)

commit db1664c8845418d09deacb0620af009158236fba
Merge: 6d987a4 f7b20c3
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 16:52:56 2018 -0500

    fixed a bunch of bugs

commit f7b20c390184a1d99fbd1a605f76e8817784a50b
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 16:50:36 2018 -0500

    added dyn when merging branches

 src/semant.ml    | 14 +++++++-------
 src/utilities.ml | 17 ++++++++++-------
 2 files changed, 17 insertions(+), 14 deletions(-)

commit a432bf9306edd1256d336db9eef85390c7b753e0
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Mon Dec 17 16:49:56 2018 -0500

    broken, wip

 llvm-test.cl   | 38 ++++++++++++++++------
 src/codegen.ml | 96 ++++++++++++++++++++++++++++++++++++++++++++++++++++++--------
 2 files changed, 112 insertions(+), 22 deletions(-)

commit cc7526149c84ab941b09f57926d7ca4069b46822
Merge: d144593 6d987a4
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 16:24:11 2018 -0500

    Merge branch 'llvm' into exceptions

commit 6d987a49f27abc4b979c59b951d2fa76993d2f06
Merge: b5e2cf5 d8d7598
```

```
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 16:20:06 2018 -0500

    merged with latest SAST

commit d8d75985483faf71d66088ea212af2b35850ba25
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 16:18:16 2018 -0500

    deleted codegen. not sure why it was there

 src/codegen.ml | 1047 ------------------------------------------------------
 1 file changed, 1047 deletions(-)

commit 38ce99914fd53f0ec2af746d68b09bbd73e5cf70
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 16:14:01 2018 -0500

    fixed lots of bugs. single-line comments now work properly, function evaluation works right, pri

 src/codegen.ml          | 1047 ++++++++++++++++++++++++++++++++++++++++++++++++
 src/coral.ml            |    1 +
 src/parser.mly          |    4 +-
 src/scanner.mll         |   13 +-
 src/semant.ml           |   37 +-
 tests/stest-comment1.cl |    5 +
 tests/stest-func11.cl   |   16 +
 tests/stest-func11.out  |    7 +
 8 files changed, 1110 insertions(+), 20 deletions(-)

commit 45f242dacd784c71c67ca369b1b4f65bdca99368
Merge: c6c2b45 b5e2cf5
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 14:20:46 2018 -0500

    Merge branch 'llvm' into optimization

commit d144593403952589a0de7e66ea40f9683fc7c554
Merge: ca13390 b5e2cf5
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 14:20:36 2018 -0500

    Merge branch 'llvm' into exceptions

commit b5e2cf5d529dfc81e9feb8d545a9a4ccb2806159
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 13:04:06 2018 -0500

    updated error messages to separate codegen from semant

 src/coral.ml | 16 ++++++++++-----
 1 file changed, 11 insertions(+), 5 deletions(-)

commit c6c2b453c93c67afcfb828aacb02ecb12e6ec9f9
```

```
Merge: 5f0dac4 a56e17d
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 12:30:21 2018 -0500

    merged with llvm, modified everything

commit ca133905adb2a3856ff650b3e66ceb4fc21494e5
Merge: f1d94c3 a56e17d
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 12:29:14 2018 -0500

    Merge branch 'llvm' into exceptions

commit a56e17d210c2809525a9a5a6efa80c6a110bdb78
Merge: a0332d2 2e839ab
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 12:28:45 2018 -0500

    Merge branch 'SAST' into llvm

commit 2e839ab395c4cc6d5ac5db5ed692ea800719f263
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 12:28:34 2018 -0500

    removed getopt

 src/getopt/getopt.ml  | 106 ------------------------------------------
 src/getopt/getopt.mli | 120 ------------------------------------------------
 2 files changed, 226 deletions(-)

commit f1d94c38325f7d106a0a036ad959d639000370a0
Merge: 4d98cc0 a0332d2
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 12:27:51 2018 -0500

    small merge

commit a0332d2e854648d9ccbb238e93a98436e2bbe477
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 12:27:19 2018 -0500

    fixed small bug

 src/coral.ml | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 4d98cc0a8e4aee762d43b58305c39971bc056c91
Merge: 2cc5f82 c81f920
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 12:26:53 2018 -0500

    merged with llvm

commit c81f9207f29c538f018ce371575e6bf35e8465eb
```

```
Merge: 7fba649 4e203e0
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 12:24:27 2018 -0500

    removed getopt

commit 4e203e03ce808e7bfe9f6635d5d6d8de098bbe45
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 12:17:47 2018 -0500

    added comments to SAST

 src/semant.ml | 80 +++++++++++++++++++++++++++++++++--------------------------
 1 file changed, 45 insertions(+), 35 deletions(-)

commit f7e902068e5249ae59309222b972fef94eb44bab
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 12:17:20 2018 -0500

    refactored code

 Makefile     |   2 +-
 src/coral.ml | 138 ++++++++++++++++++++++++++++++++++++++++-------------------
 testall.sh   |   2 +-
 3 files changed, 95 insertions(+), 47 deletions(-)

commit 7fba64948e9a55b7f8a668110539b2ced2b1ad77
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 12:05:41 2018 -0500

    updated compilation

 Makefile      |   4 +--
 src/coral.ml  | 108 +++++++++++++++++++++++++++++++---------------------------
 src/semant.ml |  54 +++++++++++++++++------------
 testall.sh    |   4 +--
 4 files changed, 94 insertions(+), 76 deletions(-)

commit 2cc5f82ba7482e14e5935b1893c1c3154a408795
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 12:02:39 2018 -0500

    refactored to allow Python style compilation, added comments

 Makefile      |   4 +--
 src/coral.ml  | 108 +++++++++++++++++++++++++++++++---------------------------
 src/semant.ml |  52 +++++++++++++++++-----------
 testall.sh    |   4 +--
 4 files changed, 93 insertions(+), 75 deletions(-)

commit 5f0dac4d81296fd22595c8304d5f6e0e43fec197
Merge: 863f926 ce5aa4f
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 01:54:20 2018 -0500
```

```
    Merge branch 'llvm' into optimization

commit a786a513539171d32c24731b7684b5a78cd7dc31
Merge: 875f805 ce5aa4f
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 01:54:09 2018 -0500

    Merge branch 'llvm' into exceptions

commit ce5aa4f72fb1612a2867f6d7fbb448b57ea4cfb6
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 01:53:50 2018 -0500

    updated gitignore

 .gitignore | 2 ++
 1 file changed, 2 insertions(+)

commit e8ee386ab43f2b3c2f3290ec431be01d5d77bede
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 01:52:57 2018 -0500

    removed temp files

 main      |  Bin 14816 -> 0 bytes
 source.ll | 1409 ------------------------------------------
 source.s  | 1883 ------------------------------------------------------------
 3 files changed, 3292 deletions(-)

commit ec545a8ed00f307ececdadcf3980a4e80f1ca55e
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 01:52:47 2018 -0500

    merged updated comments with exceptions

 main         |  Bin 0 -> 14816 bytes
 source.ll    | 1409 ++++++++++++++++++++++++++++++++++++++++++++
 source.s     | 1883 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 src/coral.ml |   30 +-
 4 files changed, 3317 insertions(+), 5 deletions(-)

commit 863f9263159b3a0a4191af0707c56c736b427850
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 01:51:27 2018 -0500

    refactored to remove bash scripts

 src/coral.ml | 6 ++----
 1 file changed, 2 insertions(+), 4 deletions(-)

commit ddf57070cba3672c342ce35b3793c032db7372b0
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 01:50:28 2018 -0500
```

```
    refactored compilation process to remove bash scripts

 inter.sh     | 2 --
 src/coral.ml | 4 ++--
 2 files changed, 2 insertions(+), 4 deletions(-)

commit 875f80589700d1d38bd2b4fd22fcee72c7284050
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 01:49:36 2018 -0500

    refactored compilation process to remove bash scripts

 inter.sh     |  2 --
 src/coral.ml | 33 ++++++++++++++++++++++++++++++------
 2 files changed, 27 insertions(+), 8 deletions(-)

commit 3406ae6bde389f66f1ef25a08a199d535c67cd98
Merge: 35d9f3a 2f421db
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Dec 17 01:23:53 2018 -0500

    merged with llvm

commit 894f96362bf9f611d015a713e90c55c4689db2e4
Merge: 5d23b58 2f421db
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Dec 16 20:45:38 2018 -0500

    merged with llvm

commit 2f421dbba00855d56743eb3abcb73f5fe011b0bf
Merge: 53eef61 99a92d7
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Dec 16 20:43:42 2018 -0500

    merged with SAST

commit 99a92d76dedf8eef317d207e281b68e0c07db747
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Dec 16 20:42:54 2018 -0500

    small changes to error messages

 src/coral.ml | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)

commit 53eef61b5f8aa5c9ab19eec4113913af3567f4d7
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Dec 16 20:40:06 2018 -0500

    fixed makefile

 Makefile     | 2 +-
```

```
 _tags        | 2 +-
 src/coral.ml | 4 ++--
 3 files changed, 4 insertions(+), 4 deletions(-)

commit abea7f54f97fa0f36e358e4b25974b16ffd21389
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Dec 16 20:37:45 2018 -0500

    modified interpreter. use ./coral.native -r -c [file], not comp.sh

 Makefile     | 6 ++----
 comp.sh      | 2 --
 src/coral.ml | 8 +++++++-
 testall.sh   | 7 +------
 4 files changed, 10 insertions(+), 13 deletions(-)

commit 5d23b5852d067d1603fbe16fd0e351af68dcfe7e
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Dec 16 20:35:13 2018 -0500

    updated how the compiler works

 Makefile       | 6 ++----
 comp.sh        | 2 --
 llvm-test.cl   | 2 +-
 src/codegen.ml | 4 ++--
 src/coral.ml   | 8 +++++++-
 testall.sh     | 7 +------
 6 files changed, 13 insertions(+), 16 deletions(-)

commit 9d59363d38988705ea8b853e5d5a84adf979b40d
Merge: 8d254c9 1e6d231
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Dec 16 20:11:18 2018 -0500

    merged with SAST

commit 1e6d23163ab23e522cb4b754a20cace5f9b3913a
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Dec 16 20:06:26 2018 -0500

    added new tests

 src/sast.ml            |  6 ++++--
 src/utilities.ml       |  8 ++++----
 tests/stest-func10.cl  | 14 ++++++++++++++
 tests/stest-func10.out |  2 ++
 4 files changed, 24 insertions(+), 6 deletions(-)

commit 12846a6918037664c586be414aefa3d42221377c
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Dec 16 20:01:25 2018 -0500

    fixed interpreter
```

```
 inter.sh     | 2 +-
 src/coral.ml | 8 ++++----
 2 files changed, 5 insertions(+), 5 deletions(-)

commit badca947ad948620ddae4c6a2de778c17e5618c1
Merge: 7bc5e39 81dc2d4
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Dec 16 19:52:31 2018 -0500

    merged with SAST, updated tests

commit 81dc2d40cfa8ec16f7da9812f2dffdad884dd370
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Dec 16 19:28:32 2018 -0500

    updated gitignore

 .gitignore  |   3 +-
 testall.log | 175 ------------------------------------------------------------
 2 files changed, 2 insertions(+), 176 deletions(-)

commit 6c1dc8f4b601ed6eff444274925cbcc4b729b850
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Dec 16 19:28:01 2018 -0500

    added import statements

 src/ast.ml              |   2 +
 src/coral.ml            | 115 ++++++++++++++++++++++++----------
 src/parser.mly          |   4 +-
 src/scanner.mll         |   3 +-
 src/semant.ml           |  12 ++--
 src/utilities.ml        |  10 ++-
 testall.log             | 175 ++++++++++++++++++++++++++++++++++++++++++++++++
 tests/stest-import1.cl  |   3 +
 tests/stest-import1.out |   1 +
 tests/stestlib.cl       |   9 +++
 10 files changed, 291 insertions(+), 43 deletions(-)

commit 7bc5e39d08a244103107506269f403a8dffecb54
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Sun Dec 16 19:18:04 2018 -0500

    binary and unary op exceptions

 inter.sh                   |   2 +-
 src/codegen.ml             | 293 ++++++++++++++++++++++++++++++++-------------
 src/utilities.ml           |   2 +-
 tests/fail-badarg.cl       |  12 ++
 tests/fail-badarg.out      |   2 +
 tests/fail-badop-unary.cl  |  11 ++
 tests/fail-badop-unary.out |   2 +
 tests/fail-badop.cl        |  11 ++
```

```
 tests/fail-badop.out       |   2 +
 9 files changed, 247 insertions(+), 90 deletions(-)

commit 7f2eac3113128a9689f76091fcaf5a45d1e4738a
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Dec 16 17:35:28 2018 -0500

    added hacky Transform type for use in codegen

 src/sast.ml      |  2 ++
 src/semant.ml    | 26 +++++++++++++++++---------
 src/utilities.ml | 23 +++++++++++++++++++++++
 3 files changed, 42 insertions(+), 9 deletions(-)

commit 35d9f3ac5de41872864760d5e82e532373c2df90
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Sat Dec 15 23:53:16 2018 -0500

    made it so all non int/float/bool will be alloca'd as cobj_t, rather than only Dyn doing so

 src/codegen.ml | 6 +++---
 1 file changed, 3 insertions(+), 3 deletions(-)

commit fca824e1309d2b4973493df1171ad1190ce86441
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Sat Dec 15 20:57:43 2018 -0500

    functions work. Nonfirstclass. Hoooooray

 llvm-test.cl   | 28 ++++++----------------
 src/codegen.ml | 83 ++++++++++++++++++++++++++++++++++++++++++++++++++++++---------------

 2 files changed, 72 insertions(+), 39 deletions(-)

commit 2f866d9fc19ec0fa3311e81da3c7d4b26d2b01d5
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Sat Dec 15 19:08:06 2018 -0500

    SWhile added

 llvm-test.cl   |  5 +++--
 src/codegen.ml | 17 ++++++++++++-----
 2 files changed, 15 insertions(+), 7 deletions(-)

commit 759c423aed59ee160fa6108fa83acf6aa1dc1095
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Sat Dec 15 18:55:07 2018 -0500

    sprint for floats added

 llvm-test.cl   |  4 ++--
 src/codegen.ml | 11 +++++++++--
 2 files changed, 11 insertions(+), 4 deletions(-)

commit 3af75ec95a33ad3b963ca12202a1dee619c8ac30
```

```
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Sat Dec 15 18:50:36 2018 -0500

    added SIf

 llvm-test.cl   |  6 ++++++
 src/codegen.ml | 14 ++++++++++----
 2 files changed, 16 insertions(+), 4 deletions(-)

commit fa3c5b2db8a7af77a9f237b91bba40446f0b0104
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 15 18:42:58 2018 -0500

    added tests

 src/semant.ml            |  4 ++--
 tests/stest-comment1.cl  |  4 ++++
 tests/stest-comment1.out |  1 +
 tests/stest-func9.cl     | 35 +++++++++++++++++++++++++++++++++++
 tests/stest-func9.out    |  5 +++++
 5 files changed, 47 insertions(+), 2 deletions(-)

commit 6fde56d97cd29b0af3b91ecc0e617e1e701556c7
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Sat Dec 15 18:40:28 2018 -0500

    added binops for Raws

 llvm-test.cl   |  3 ++-
 src/codegen.ml | 39 +++++++++++++++++++++++++++++++++++++++
 2 files changed, 41 insertions(+), 1 deletion(-)

commit 742260ea887310f6fd3d7eb72f7c7f75b9dca679
Merge: bd1a364 8d254c9
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 15 17:36:14 2018 -0500

    final merge

commit 8d254c97c29e0fdba3bc84f3e2592d5440c9cac2
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 15 17:35:06 2018 -0500

    fixed some bugs

 llvm-test.cl           | 10 ++++++++++
 src/codegen.ml         | 10 ++++++++--
 src/coral.ml           | 33 +++++++++++++++++++++++++++++++++++----------------
 src/semant.ml          | 19 +++++++++++++++++++
 src/utilities.ml       |  1 +
 tests/test-forloops.cl |  2 +-
 6 files changed, 51 insertions(+), 24 deletions(-)

commit bd1a3649673822c62f2633b88695a39ee11d6166
```

```
Merge: 5f676e0 22af91c
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 15 15:35:54 2018 -0500

    Merge branch 'llvm' into optimization

commit 22af91cc14521c57c644f06345c5b224a6027196
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 15 15:35:40 2018 -0500

    removed unnecessary variable

 src/coral.ml | 3 ---
 1 file changed, 3 deletions(-)

commit f89e661c4d862895d100d790ad9fda41c5975277
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 15 15:25:36 2018 -0500

    removed temp files

 .ll         | 1408 ------------------------------------------
 main        |  Bin 14816 -> 0 bytes
 source.ll   | 1462 ------------------------------------------
 source.s    | 1883 ------------------------------------------------------------
 testall.log |  351 -----------
 5 files changed, 5104 deletions(-)

commit 5f676e097e8435e22c49246fbe8ea42ecc6a66ba
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 15 15:25:10 2018 -0500

    removed temp files, merged

 .ll         | 1408 ------------------------------------------
 main        |  Bin 14816 -> 0 bytes
 source.ll   | 1462 ------------------------------------------
 source.s    | 1883 ------------------------------------------------------------
 testall.log |  351 -----------
 5 files changed, 5104 deletions(-)

commit ea14c5733d60ff6008756c965e7da951da4fef45
Merge: f88d4f1 5f4e3ab
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 15 15:24:55 2018 -0500

    Merge branch 'llvm' into optimization

commit 5f4e3abfbea36c47d263f1df8a5b63c718db6d0e
Merge: 1dca610 2ce4d2f
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 15 15:24:35 2018 -0500

    merged with SAST, added lots of documentation
```

```
commit 2ce4d2f63ee927df0581edffcfed7c2e50c463fd
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 15 15:20:51 2018 -0500

    added a ton of documentation

 _tags            |   2 +
 src/ast.ml       |   8 +++
 src/coral.ml     | 202 +++++++------------------------------------------------
 src/interpret.ml |  90 +++++++++++++++++++++++++
 src/parser.mly   | 115 ++++++++++++++++++++++++++-----
 src/scanner.mll  |  13 ++--
 src/semant.ml    | 157 ++++++++++++++++++++++++------------------
 src/utilities.ml | 112 ++++++++++++++++++++++++++----
 8 files changed, 418 insertions(+), 281 deletions(-)

commit f88d4f14b1a74dfe35591f615a6890b5b71ac77e
Merge: e41f093 1dca610
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 15 13:34:57 2018 -0500

    updated and merged

commit a710b2ab9697963eb451fa4a51b9fa134f8d76db
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 15 13:20:19 2018 -0500

    strings are now arrays

 src/utilities.ml | 1 +
 1 file changed, 1 insertion(+)

commit 1dca6107fb4ebfa0961d0c4f7994a39c865b3ba4
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 15 13:19:55 2018 -0500

    removed test files

 .ll         | 1408 -------------------------------------------
 main        |  Bin 14776 -> 0 bytes
 source.ll   | 1438 --------------------------------------------
 source.s    | 1918 -----------------------------------------------------------
 testall.log |  351 --------
 5 files changed, 5115 deletions(-)

commit a756a7a85a594109c601d354eb4ade92f1425e45
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 15 13:19:46 2018 -0500

    strings now work

 .ll              | 1408 +++++++++++++++++++++++++++++++++++++++
 main             |  Bin 0 -> 14776 bytes
```

```
 source.ll        | 1438 ++++++++++++++++++++++++++++++++++++++++
 source.s         | 1918 ++++++++++++++++++++++++++++++++++++++++++++++++++++++
 src/utilities.ml |    1 +
 testall.log      |  351 ++++++++++
 6 files changed, 5116 insertions(+)

commit c926336e6bb21f581d1c06176cff583212ebbcdb
Merge: de7f244 36ed024
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 15 13:15:37 2018 -0500

    merged new SAST, fixed lots of bugs

commit e41f093be5f87b103bf1875ec127931679813e2e
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Sat Dec 15 13:10:48 2018 -0500

    basic optimization of immutables works

 llvm-test.cl   |  19 ++++++--
 src/codegen.ml | 147 +++++++++++++++++++++++++++++++++++++++++----------------
 2 files changed, 121 insertions(+), 45 deletions(-)

commit 36ed0247ce1385c266aff9e0d5effe0c0461162b
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 15 12:58:08 2018 -0500

    huge overall of the syntax, etc.

 src/ast.ml               |  10 +-
 src/coral.ml             |  15 +--
 src/parser.mly           |   6 +-
 src/sast.ml              |  45 +++++----
 src/scanner.mll          |   1 +
 src/semant.ml            | 261 +++++++++++++++++++++++++++----------------------
 src/utilities.ml         |  14 +--
 testall.log              | 120 -----------------------
 tests/sfail-list3.cl     |   7 ++
 tests/sfail-list3.out    |   1 +
 tests/sfail-type2.cl     |   2 +-
 tests/sfail-type3.cl     |   4 +
 tests/sfail-type3.out    |   1 +
 tests/stest-exper1.cl    |   4 +
 tests/stest-exper1.out   |   1 +
 tests/stest-exper2.cl    |   4 +
 tests/stest-exper2.out   |   1 +
 tests/stest-func6.cl     |   8 ++
 tests/stest-func6.out    |   1 +
 tests/stest-func7.cl     |   8 ++
 tests/stest-func7.out    |   1 +
 tests/stest-func8.cl     |   8 ++
 tests/stest-func8.out    |   1 +
 tests/stest-list2.cl     |   7 ++
 tests/stest-list2.out    |   1 +
```

```
 25 files changed, 259 insertions(+), 273 deletions(-)

commit 02cd4b010e5dbd0a7f4ffcfbc7ae4a9d049ac451
Merge: 78c8cee 18ac802
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Dec 14 18:06:29 2018 -0500

    merged

commit 18ac8027e0bcb6d2800c70ef7461a7477724a3dc
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Dec 14 18:03:31 2018 -0500

    removed testall

 testall.log | 120 ------------------------------------------------------------
 1 file changed, 120 deletions(-)

commit de7f244ba9144e507807518c40977eac8e7f5ed3
Merge: 76bca8a d12d373
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Fri Dec 14 11:49:22 2018 -0500

    merged for-loops into llvm

commit d12d373118e4bef2adccde258cbf5e3415e95ac4
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Fri Dec 14 03:24:15 2018 -0500

    added for loops

 comp.sh                 |  2 +-
 src/codegen.ml          | 97 ++++++++++++++++++++++++++++++++++++++-------------
 src/semant.ml           | 15 ++++++---
 tests/test-forloops.cl  | 14 +++++++
 tests/test-forloops.out | 12 ++++++
 5 files changed, 106 insertions(+), 34 deletions(-)

commit 78c8cee9ef516a6448c7ee84679d3ee2214a77de
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Thu Dec 13 21:56:20 2018 -0500

    fixed for loops

 src/semant.ml |   6 +--
 testall.log   | 120 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 2 files changed, 123 insertions(+), 3 deletions(-)

commit 76bca8a70533c390ca74df0d57c03bcf477ab42a
Merge: df27498 c0f6b09
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Thu Dec 13 17:36:53 2018 -0500

    merged SAST into llvm branch. string access as a list doesnt work bc of semant, but codegen comp
```

```
commit c0f6b096c8cc24a118b7a2282d1dc6572b3f9f19
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Thu Dec 13 14:02:35 2018 -0500

    removed testall.log

 testall.log | 120 ------------------------------------------------------------
 1 file changed, 120 deletions(-)

commit b3f6920a5f66bf83684e29a372fa3053450099ab
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Thu Dec 13 14:02:22 2018 -0500

    SAST now includes types

 src/sast.ml   |   8 +++-
 src/semant.ml |  44 ++++++++++++----------
 testall.log   | 120 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 3 files changed, 151 insertions(+), 21 deletions(-)

commit df27498f72cd72fbc60560986cad3ede9c0fa470
Merge: 07ce337 8590949
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Thu Dec 13 13:27:05 2018 -0500

    Merge branch 'llvm' of https://github.com/ja3067/Coral into llvm

commit 07ce3378e3474b6ab5efa0588aba0bc6ba6f7409
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Thu Dec 13 13:26:39 2018 -0500

    added tests for lists and strings

 tests/test-idx.cl  | 4 ++++
 tests/test-idx.out | 1 +
 tests/test-str.cl  | 4 ++++
 tests/test-str.out | 1 +
 4 files changed, 10 insertions(+)

commit 8590949bb57ee2b9cf27411783d4d3ce5afc7304
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Thu Dec 13 12:47:45 2018 -0500

    no more segfaults

 llvm-test.cl | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 7170af6a96fed96810ddabc0d72ad81057ee3d2c
Merge: 302f2bf 6a628d8
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Thu Dec 13 12:45:34 2018 -0500
```

```
    Merge branch 'SAST' of https://github.com/ja3067/Coral into SAST

commit 302f2bfe96c67e8c6321abdd842600ef1bc3f6f7
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Thu Dec 13 12:45:31 2018 -0500

    updated Makefile

 Makefile | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit d48f8dd63eb06b3ac26f3f48a5d638e0d2342a87
Merge: 3201e91 7b08370
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Thu Dec 13 12:28:33 2018 -0500

    merged functions and lists

commit 3201e912eb01ad309461cd9be1f929ac3116aa88
Merge: fc7af74 858e3f8
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Thu Dec 13 11:54:40 2018 -0500

    Merge branch 'llvm' of https://github.com/ja3067/Coral into llvm

commit 858e3f8ec332c282f70b3490b72abc3618bb9d8b
Merge: 4cbc5c1 cc92a1d
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Thu Dec 13 11:53:02 2018 -0500

    Merge branch 'functions' into llvm

commit 7b08370abb4e2e686c188f719b2f68a5fb47042e
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Sun Dec 9 21:51:18 2018 -0500

    indexing

 src/codegen.ml | 37 ++++++++++++++++++++-----------------------
 1 file changed, 20 insertions(+), 17 deletions(-)

commit 3f35b9f3eb732ed1646b9527d888ce792766488e
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Tue Dec 4 17:36:00 2018 -0500

    saving progress

 llvm-test.cl     |   8 +--
 src/codegen.ml   | 196 +++++++++++++++++++++++++++++++++++++++++++++++++------------------------
 src/sast.ml      |   1 -
 src/semant.ml    |   1 +
 src/utilities.ml |   1 +
 5 files changed, 138 insertions(+), 69 deletions(-)
```

```
commit 6a628d8fe97500da09a7f149f2e3fd754f8530eb
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Dec 2 16:18:08 2018 -0500

    fixed shift reduce errors

 src/coral.ml    |   2 +
 src/parser.mly | 129 +++++++++++++++++++++++++++++-----------------------
 2 files changed, 68 insertions(+), 63 deletions(-)

commit e895e5fda55326bcf9a400c092c665c0c8fe9e79
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 1 17:10:47 2018 -0500

    removed temp files

 testall.log | 120 ------------------------------------------------------------
 1 file changed, 120 deletions(-)

commit ccfaec54a2f3801064e06b687294d999da3f3698
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Dec 1 17:10:30 2018 -0500

    first steps towards lists

 src/ast.ml            |   7 +-
 src/coral.ml          |   2 +-
 src/parser.mly        | 134 +++++++++++++++++++---------------------
 src/sast.ml           |  29 ++++----
 src/semant.ml         | 179 ++++++++++++++++++++++++++++++++++----------------
 src/utilities.ml      |   6 +-
 testall.log           | 120 +++++++++++++++++++++++++++++++++
 tests/stest-list1.cl  |   7 ++
 tests/stest-list1.out |   4 ++
 9 files changed, 342 insertions(+), 146 deletions(-)

commit cc92a1df3f86d15354c6bcac0ce34d12d411abf0
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Tue Nov 27 15:47:39 2018 -0500

    removed old print statements from interpreter for convenience

 src/coral.ml | 8 +++++++-
 1 file changed, 7 insertions(+), 1 deletion(-)

commit bcc09e4013ddcd712f185b068d8a097ea3a76ee9
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Tue Nov 27 13:03:23 2018 -0500

    added an interpreter to the functions branch. it is very crude and for instance will continue to

 Makefile       |  2 +-
 inter.sh       |  2 ++
 llvm-test.cl   |  3 ---
```

```
 src/codegen.ml |  2 +-
 src/coral.ml   | 50 ++++++++++++++++++++++++++++++++++++++++----------
 5 files changed, 44 insertions(+), 15 deletions(-)

commit 8bad7e30640244324de645899e3d5be351fb376b
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Tue Nov 27 00:56:38 2018 -0500

    added lots more tests

 src/semant.ml         |  2 +-
 tests/sfail-func4.cl  |  7 +++++++
 tests/sfail-func4.out |  1 +
 tests/sfail-func5.cl  | 10 ++++++++++
 tests/sfail-func5.out |  1 +
 5 files changed, 20 insertions(+), 1 deletion(-)

commit 4bad376156544a14cfd08946cc4d0a6c67b36bcb
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Nov 26 19:41:08 2018 -0500

    added new tests

 tests/sfail-func3.cl  | 5 +++++
 tests/sfail-func3.out | 1 +
 2 files changed, 6 insertions(+)

commit d055bc416403d6d4395cbd16d32c7a7515a6971b
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Nov 26 19:01:12 2018 -0500

    added many more tests

 tests/stest-type4.cl  | 29 +++++++++++++++++++++++++++++
 tests/stest-type4.out | 10 ++++++++++
 2 files changed, 39 insertions(+)

commit c3db017fc0a33b5a46b9a63990efe6c023a335ea
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Nov 26 18:38:15 2018 -0500

    added more recurse tests

 src/coral.ml             | 1 +
 tests/stest-globals2.cl  | 4 ++++
 tests/stest-globals2.out | 1 +
 tests/stest-recurse2.cl  | 9 +++++++++
 tests/stest-recurse2.out | 1 +
 5 files changed, 16 insertions(+)

commit 078f9e049ed9050d91e42b2f70e09ac413c08413
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Nov 26 18:33:34 2018 -0500
```

```
    fixed globals

 src/semant.ml            | 13 +++++---
 testall.log              | 85 ----------------------------------------------
 tests/stest-globals2.cl  |  8 +++++
 tests/stest-globals2.out |  1 +
 4 files changed, 17 insertions(+), 90 deletions(-)

commit b0136324d0bd1bfc8c2653bec8ea62ca6bb1eb32
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Nov 26 17:02:20 2018 -0500

    globals work now

 src/semant.ml            |  8 +++++
 testall.log              | 85 ++++++++++++++++++++++++++++++++++++++++++++++
 tests/stest-globals1.cl  |  9 +++++
 tests/stest-globals1.out |  2 ++
 4 files changed, 104 insertions(+)

commit 27cedcf489cdd57ce0c6aac6682b72343dc3cb59
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Nov 26 16:52:45 2018 -0500

    cleaned up semant

 src/semant.ml | 1 +
 1 file changed, 1 insertion(+)

commit f15d4588b4e5db5082d684920e4c1bdef0df70bb
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Nov 26 16:37:38 2018 -0500

    fixed bug in function assignment

 src/semant.ml         | 32 +++++++++++++++++-------------
 tests/sfail-func2.cl  |  4 ++++
 tests/sfail-func2.out |  1 +
 3 files changed, 22 insertions(+), 15 deletions(-)

commit 1ca110fc1158437fbbdaf49c077e845b34322fd3
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Nov 26 10:42:09 2018 -0500

    added boolean operations to binop

 src/semant.ml | 9 ++++++---
 1 file changed, 6 insertions(+), 3 deletions(-)

commit bf07133235d9320d9cf1664ac323ba5562bc2b68
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Nov 26 10:35:57 2018 -0500

    removed temp files
```

```
 testall.log | 75 ----------------------------------------------------------
 1 file changed, 75 deletions(-)

commit 40e60e8a0a79923e9d01c00412160f5ed021be92
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Nov 26 10:35:36 2018 -0500

    updated formatting and consolidated frequently used functions to easy maintenance

 src/semant.ml | 366 ++++++++++++++++++++++++++++++++++++-------------------------
 testall.log   |  75 ++++++++++++
 2 files changed, 274 insertions(+), 167 deletions(-)

commit fc7af74a5951b0dfd99f2c86534414882bfa8a7e
Merge: af51e0f 4cbc5c1
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Mon Nov 26 01:42:55 2018 -0500

    Merge branch 'llvm' of https://github.com/ja3067/Coral into llvm

commit abeaf89311381bf142f85c776a52527d40962f44
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Mon Nov 26 01:42:07 2018 -0500

    list instantiation

 llvm-test.cl   |  12 +-
 src/ast.ml     |   4 +-
 src/codegen.ml | 346 ++++++++++++++++++++++++++++++++++++-------------------------
 src/parser.mly |   2 +-
 4 files changed, 201 insertions(+), 163 deletions(-)

commit b23f6267569b80f272d9a0c694aa6212004acc3b
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Mon Nov 26 01:12:59 2018 -0500

    recursion works too

 fn-test.cl     | 25 +++++++++++++++++++++++++
 src/codegen.ml |  2 +-
 2 files changed, 26 insertions(+), 1 deletion(-)

commit 99472106e0e3592c1271e928115e00111522aecf
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Nov 26 00:50:28 2018 -0500

    modified makefile to compile tests

 Makefile | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)

commit 1cb15a6a1542a69ca4a39f2c39fcf79adfaffe9a
Author: Jacob Austin <ja3067@columbia.edu>
```

```
Date:   Mon Nov 26 00:49:57 2018 -0500

    added an extra test for naming

 tests/stest-name1.cl  | 7 +++++++
 tests/stest-name1.out | 1 +
 2 files changed, 8 insertions(+)

commit df99f65b654b161178df09c4ba5a76c1bfaa2fb7
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Nov 25 23:22:57 2018 -0500

    fixed bug with assigning to function names

 src/semant.ml | 8 ++++----
 1 file changed, 4 insertions(+), 4 deletions(-)

commit 3da9ecb01f5400840a85ed6e9750adfa4406e962
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Nov 25 23:20:21 2018 -0500

    added some recursion tests, fixed a bug with SCall

 src/semant.ml            | 16 ++++++++--------
 tests/stest-recurse1.cl  |  7 +++++++
 tests/stest-recurse1.out |  1 +
 3 files changed, 17 insertions(+), 7 deletions(-)

commit 95b06ddb4b65150a03b4dba4c54bfbe5a7bca553
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Nov 25 22:52:38 2018 -0500

    updated the SCall model to allow for annonymous functions

 src/ast.ml            |   4 +-
 src/coral.ml          |   4 +-
 src/parser.mly        |   2 +-
 src/sast.ml           |   4 +-
 src/semant.ml         | 142 ++++++++++++++++++++++++++++++++-------------------
 src/utilities.ml      |   2 +-
 tests/stest-func5.cl  |   6 +++
 tests/stest-func5.out |   1 +
 8 files changed, 90 insertions(+), 75 deletions(-)

commit 4cbc5c1f2c6ce29a07887734d223de0bb9bfe9b0
Merge: 92cdaf5 21c617f
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Nov 25 21:32:17 2018 -0500

    Merge branch 'SAST' into llvm

commit 21c617ffae4954255048af8a01c7ea5a58134a7a
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Nov 25 21:31:56 2018 -0500
```

```
    added an additional test to check name conflicts

 tests/stest-type3.cl  | 10 ++++++++++
 tests/stest-type3.out |  1 +
 2 files changed, 11 insertions(+)

commit 92cdaf54a2084ade43e9386b5f98f6f35d2218a5
Merge: 369bb97 56f5a10
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Nov 25 21:24:36 2018 -0500

    Merge branch 'llvm' of https://github.com/ja3067/Coral into llvm

commit 493c4c698ba4860c1c68dba1f0f227d2149c5aab
Merge: 238f7e6 56f5a10
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Sun Nov 25 20:29:52 2018 -0500

    added tests for functions, and merged in the updated semant from llvm

commit 56f5a108564b7e8b1cfcf591608575e43c009d6c
Merge: 466915b e5c2acb
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Sun Nov 25 19:56:06 2018 -0500

    Merge branch 'sast' into llvm
    merging in semant fixes

commit 238f7e6fb4a4ce7d85275c3e8a5fa63dcbd2f0b8
Merge: 704f94f 466915b
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Sun Nov 25 19:49:15 2018 -0500

    merged llvm and functions branches! No real conflicts

commit 704f94f71b4c2243ff541b4c559bb5eb035d6c22
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Sun Nov 25 19:29:55 2018 -0500

    added test file

 fn-test.cl | 83 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 83 insertions(+)

commit 369bb97a44b2ada89b08abce1e3868e603014362
Merge: 466915b e5c2acb
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Nov 25 18:24:21 2018 -0500

    Merge branch 'SAST' into llvm

commit e5c2acb2de8d91fd2bd155b247cf11e2a9867a7e
Author: Jacob Austin <ja3067@columbia.edu>
```

```
Date:   Sun Nov 25 18:24:10 2018 -0500

    updated Call to handle functions as arguments/fix bug

 src/semant.ml                   | 12 +++++++---
 testall.log                     | 50 ----------------------------------------
 test.cl => tests/stest-func4.cl |  1 +
 tests/stest-func4.out           |  1 +
 4 files changed, 11 insertions(+), 53 deletions(-)

commit 3e5f41577a2fea178fbe98264f3a231a9dd927bc
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Nov 25 18:13:40 2018 -0500

    fixed bug in function arguments

 src/semant.ml |  6 +++---
 test.cl       | 16 +++++++++-------
 testall.log   | 50 ++++++++++++++++++++++++++++++++++++++++++++++++++
 3 files changed, 62 insertions(+), 10 deletions(-)

commit e687bd15dddd1a7385f8945ca07ccbfa34e96698
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Sun Nov 25 17:43:11 2018 -0500

    refactored to use the new type 'state' and got functions working a bit better. still not totally

 Makefile       |  1 +
 src/codegen.ml | 97 ++++++++++++++++++++++++++++++++++++++++----------------
 2 files changed, 62 insertions(+), 36 deletions(-)

commit 27f7e34d5ea2831df29a20da06ddf116da9126ed
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Sun Nov 25 16:38:23 2018 -0500

    lots of function stuff works, tho conditional branches in functions and returning functions from

 Makefile       |   8 ++-
 comp.sh        |   2 +-
 src/codegen.ml | 211 ++++++++++++++++++++++++++++++++++++++++++++++++++++++---
 src/semant.ml  |   6 +-
 4 files changed, 207 insertions(+), 20 deletions(-)

commit 466915bfb07692fcfc525533dd9a04c69052ce0e
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 23:50:23 2018 -0500

    removed temp file

 test.cl | 8 --------
 1 file changed, 8 deletions(-)

commit 46826f4d25587d8d6168ee99f8bd38452fff6455
Merge: b83f5ff 828baa8
```

```
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 23:43:22 2018 -0500

    merged

commit 828baa87cc279bc0f2524cbe0301ec8fa7775f49
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 23:42:41 2018 -0500

    fixed name bug in function calls

 src/sast.ml   | 2 +-
 src/semant.ml | 4 ++--
 test.cl       | 8 ++++++++
 3 files changed, 11 insertions(+), 3 deletions(-)

commit b83f5ffaa7c0d80f123b53ac220c2aa5e7472465
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 23:04:54 2018 -0500

    removed temp file

 test.cl | 11 -----------
 1 file changed, 11 deletions(-)

commit 1f5afb6ad842e61e46089793df15216b1e7108d6
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 23:04:44 2018 -0500

    updated pretty-printing and removed error handling

 src/coral.ml |  2 +-
 src/sast.ml  |  4 ++--
 test.cl      | 11 ++++++++++
 3 files changed, 14 insertions(+), 3 deletions(-)

commit 2e7c4768caf801084454ede6d87152e76cc32240
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 15:31:40 2018 -0500

    updated makefile to compile before running tests

 Makefile | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit d8ab250b8e5e916d6359bfa1cebbf57333008262
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 13:50:12 2018 -0500

    fixed a small bug in the testing suite

 Makefile   |  2 +-
 testall.sh | 15 +++++++--------
 2 files changed, 8 insertions(+), 9 deletions(-)
```

```
commit a3a0bbe0c53e8a1d02862b330da0fb2d27b062c8
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 13:41:32 2018 -0500

    removed additional files, reverted interpreter

 main      |  Bin 14704 -> 0 bytes
 source.ll | 1405 ---------------------------------------------
 source.s  | 1855 -------------------------------------------------------------
 3 files changed, 3260 deletions(-)

commit 25b738999bfe26dd3865c34ca091accdaddb94f9
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 13:41:09 2018 -0500

    reverted interpreter to allow debugging

 main         |  Bin 0 -> 14704 bytes
 source.ll    | 1405 +++++++++++++++++++++++++++++++++++++++++++++
 source.s     | 1855 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 src/coral.ml |   10 +-
 4 files changed, 3265 insertions(+), 5 deletions(-)

commit 3c11679c7ef70cace9ab42e75a5cf90d9d09464d
Merge: c91ebeb ed52ac1
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 13:39:05 2018 -0500

    merged with latest SAST, added semant tests

commit ed52ac1c198c73c2582194093b8b053fbfa81cc4
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 13:13:35 2018 -0500

    updated tests and testall for semant

 testall.sh                                 | 6 ++++---
 tests/{fail-func1.cl => sfail-func1.cl}    | 0
 tests/{fail-func1.out => sfail-func1.out}  | 0
 tests/{fail-type1.cl => sfail-type1.cl}    | 0
 tests/{fail-type1.out => sfail-type1.out}  | 0
 tests/{fail-type2.cl => sfail-type2.cl}    | 0
 tests/{fail-type2.out => sfail-type2.out}  | 0
 tests/{test-add1.cl => stest-add1.cl}      | 0
 tests/{test-add1.out => stest-add1.out}    | 0
 tests/{test-func.cl => stest-func.cl}      | 0
 tests/{test-func.out => stest-func.out}    | 0
 tests/{test-func2.cl => stest-func2.cl}    | 0
 tests/{test-func2.out => stest-func2.out}  | 0
 tests/{test-func3.cl => stest-func3.cl}    | 0
 tests/{test-func3.out => stest-func3.out}  | 0
 tests/{test-gcd.cl => stest-gcd.cl}        | 0
 tests/{test-gcd.out => stest-gcd.out}      | 0
```

```
 tests/{test-type1.cl => stest-type1.cl}   | 0
 tests/{test-type1.out => stest-type1.out} | 0
 tests/{test-type2.cl => stest-type2.cl}   | 0
 tests/{test-type2.out => stest-type2.out} | 0
 21 files changed, 3 insertions(+), 3 deletions(-)

commit 9775ad41b4c480ebf02511bf0c6cbf7bf1bc7566
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Sat Nov 24 12:58:11 2018 -0500

    updated tags file

 _tags | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)

commit 72a69c6f4c50347519774b431bb0e25f083e93bd
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 12:45:33 2018 -0500

    can now have unindented characters at the end of files

 src/coral.ml | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 8677ba60401e32722dafbbedd69114f0756492f6
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 12:38:27 2018 -0500

    can now view SAST of file

 src/coral.ml | 1 +
 1 file changed, 1 insertion(+)

commit 58c611f023417958fab173f47ebd9efde4a9c149
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 12:30:42 2018 -0500

    added lots of tests

 src/semant.ml                                |  4 ++-
 testall.sh                                   | 38 ++++++++++++-----------------
 tests/fail-func1.cl                          |  7 +++++
 tests/fail-func1.out                         |  1 +
 tests/{test-typefail1.cl => fail-type1.cl}   |  0
 tests/{test-typefail1.out => fail-type1.out} |  0
 tests/{test-typefail2.cl => fail-type2.cl}   |  0
 tests/{test-typefail2.out => fail-type2.out} |  0
 tests/test-func2.cl                          |  3 ++-
 tests/test-func3.cl                          |  7 +++++
 tests/test-func3.out                         |  0
 tests/test-type2.cl                          |  8 ++++++
 tests/test-type2.out                         |  1 +
 13 files changed, 44 insertions(+), 25 deletions(-)
```

```
commit b0c99df82b5b766b7492c1c0f86de5d12418a6ce
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 12:15:58 2018 -0500

    added tests to SAST

 Makefile                 |   4 ++
 src/semant.ml            |  36 +++++-----
 testall.sh               | 167 +++++++++++++++++++++++++++++++++++++++++++++++
 tests/test-add1.cl       |   4 ++
 tests/test-add1.out      |   0
 tests/test-func.cl       |   7 ++
 tests/test-func.out      |   0
 tests/test-func2.cl      |   7 ++
 tests/test-func2.out     |   0
 tests/test-gcd.cl        |   9 +++
 tests/test-gcd.out       |   0
 tests/test-type1.cl      |  10 +++
 tests/test-type1.out     |   1 +
 tests/test-typefail1.cl  |   8 +++
 tests/test-typefail1.out |   1 +
 tests/test-typefail2.cl  |   8 +++
 tests/test-typefail2.out |   1 +
 17 files changed, 247 insertions(+), 16 deletions(-)

commit c91ebeb72abcc1cbe6edba438662419f350cd1b7
Merge: 02c0bab aa40ac6
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 11:53:31 2018 -0500

    Merge branch 'SAST' into llvm

commit aa40ac6ef1733e63dba5f519448dbd1ccb997592
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 11:53:14 2018 -0500

    fixed the semant bug for declaring functions with undefined variables

 src/semant.ml | 72 ++++++++++++++++++++++++++++++++---------------------------
 1 file changed, 39 insertions(+), 33 deletions(-)

commit af51e0ffc3e5d1fe4eb71f90fa458c6c9ceb6fe4
Merge: e2b8679 02c0bab
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Sat Nov 24 01:28:40 2018 -0600

    Merge branch 'llvm' of https://github.com/ja3067/Coral into llvm

commit 8d9800764f8c9fa280bdfaa247b3b14fcbfcd9e8
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 24 01:31:58 2018 -0500

    fixed bug with function arguments being added prematurely to the map
```

```
 src/semant.ml | 6 +++---
 1 file changed, 3 insertions(+), 3 deletions(-)

commit 02c0babeaeb48537e7bec40335de3b7713c27f69
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 23 23:40:19 2018 -0500

    updated Makefile to remove all temp files

 Makefile      | 6 +-----
 src/coral.ml | 2 +-
 2 files changed, 2 insertions(+), 6 deletions(-)

commit b257a59ac25e09e1266a4c549c0f2c69be14aaaa
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 23 23:33:24 2018 -0500

    removed temp files

 main      |  Bin 14704 -> 0 bytes
 source.ll | 1405 ----------------------------------------------
 source.s  | 1855 -------------------------------------------------------------
 3 files changed, 3260 deletions(-)

commit 34d4caa1e3af466753192fd265393f82c8bcca01
Merge: f712cb1 ddfddcf
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 23 23:31:52 2018 -0500

    merged with SAST

commit f712cb1584102abd7d12065ddf65b10cfec4ccae
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Fri Nov 23 22:07:20 2018 -0600

    operators complete

 llvm-test.cl   |   9 ++-
 src/codegen.ml | 239 ++++++++++++++++++++++++++++++---------------------
 2 files changed, 132 insertions(+), 116 deletions(-)

commit ddfddcf431889f1e7c65db7719c8441ddb5753ad
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 23 22:57:27 2018 -0500

    fixed a few match cases

 src/coral.ml  | 32 ++++++++++++++++----------------
 src/semant.ml |  3 ++-
 2 files changed, 18 insertions(+), 17 deletions(-)

commit a136424b117e01f486a07cfe2dc98d8aeed82136
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 23 22:50:59 2018 -0500
```

```
    added some comments and improved error messages

 src/coral.ml  | 24 ++++++++++++----------
 src/semant.ml | 10 ++++------
 2 files changed, 18 insertions(+), 16 deletions(-)

commit 4e9dc454d231e93c97f42216e12212b5a2764e91
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Fri Nov 23 20:40:46 2018 -0600

    saving progress

 llvm-test.cl   | 2 ++
 src/codegen.ml | 8 ++++----
 src/coral.ml   | 1 -
 3 files changed, 6 insertions(+), 5 deletions(-)

commit 98b523b498c417e2ca5378cbf9a901bc8768839e
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 23 20:26:07 2018 -0500

    reformatted SFunc as a record

 src/ast.ml    |  2 +-
 src/sast.ml   | 12 ++++++++++--
 src/semant.ml | 48 +++++++++++++++++++++++++------------------------
 3 files changed, 35 insertions(+), 27 deletions(-)

commit ef136aa8d27cf8d919c24b29d40be0eef7bbd27f
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 23 19:10:44 2018 -0500

    slightly modified local/global API

 src/semant.ml | 14 +++++++-------
 1 file changed, 7 insertions(+), 7 deletions(-)

commit d33992008201b9c1b69b8cc40d2a85e3275aaf40
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 23 19:09:32 2018 -0500

    removed SFuncDecl

 src/ast.ml     |   2 +-
 src/parser.mly |   2 +-
 src/sast.ml    |   5 +--
 src/semant.ml  | 140 ++++++++++++++++++++++++++++++++++++++++++------------
 4 files changed, 109 insertions(+), 40 deletions(-)

commit 8f3ec579ab72e69c86edc69c2a0bc578e7aec502
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 23 17:44:18 2018 -0500
```

```
    fixed edge case with function calls

 src/semant.ml | 6 ++++--
 1 file changed, 4 insertions(+), 2 deletions(-)

commit eff6c0a1672a891a7d86c3fec6a130c64c03153f
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 23 17:27:13 2018 -0500

    fixed a bug with return types in functions

 src/sast.ml   |  4 ++--
 src/semant.ml | 24 ++++++++++++++----------
 2 files changed, 16 insertions(+), 12 deletions(-)

commit bd456ed5167f5359e9aaaf021352cfa491611deb
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 23 16:42:52 2018 -0500

    fixed a few bugs with assign

 src/coral.ml  | 2 +-
 src/semant.ml | 6 +++---
 2 files changed, 4 insertions(+), 4 deletions(-)

commit ae661f5becf68b7965aa0796ae5e792db45c2d42
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 23 16:32:48 2018 -0500

    added documentation

 src/sast.ml      |  44 ++++++------
 src/semant.ml    | 205 +++++++++++++++++++++++++++++++++++++++++++++++++++++++
 src/utilities.ml | 134 ++++++++++++++++++++++++++++++++++++++
 3 files changed, 187 insertions(+), 196 deletions(-)

commit 28bd0b343e124485d1a2eaf125adac4da810855f
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 23 16:05:28 2018 -0500

    fixed a little bug

 src/semant.ml | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)

commit 581a7a34235160553b07ba3ab140861b1c94d121
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 23 16:01:00 2018 -0500

    recursion sort of works

 src/ast.ml    |  2 +-
 src/semant.ml | 95 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 2 files changed, 55 insertions(+), 42 deletions(-)
```

```
commit 739d2148d4d1f5bdeb51ea0ed76c36f0033f4102
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Fri Nov 23 14:35:58 2018 -0600

    still segfaults smh

 llvm-test.cl | 9 ---------
 src/coral.ml | 5 +++--
 2 files changed, 3 insertions(+), 11 deletions(-)

commit 8340940988066c445b862b3f8ceeab21507805f9
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 23 15:06:17 2018 -0500

    first class functions work

 src/ast.ml    |   5 +-
 src/coral.ml  |   9 +-
 src/sast.ml   |   5 +-
 src/semant.ml | 310 ++++++++++++++++++++++++++++++++++------------------------------
 4 files changed, 194 insertions(+), 135 deletions(-)

commit c1de5fcf6fec98c32bb42db50adc394e5b3ed53d
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Fri Nov 23 02:47:09 2018 -0600

    operators (binary and unary) compile but segfault at runtime

 Makefile       |  2 +-
 comp.sh        |  3 +-
 src/codegen.ml | 90 +++++++++++++++++++++++++++++-----------------------------
 3 files changed, 48 insertions(+), 47 deletions(-)

commit 5f47cd0e5a74fe697a0fb3e893eed582295742ed
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 23 02:11:32 2018 -0500

    added some list stuff

 src/ast.ml     |  1 +
 src/parser.mly |  1 +
 src/sast.ml    |  1 +
 src/semant.ml  | 16 +++++++++-------
 4 files changed, 12 insertions(+), 7 deletions(-)

commit 42f2aa5b23a68c809795a52b7833a032a139e56c
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Thu Nov 22 22:00:35 2018 -0600

    added unary operators

 src/codegen.ml | 235 ++++++++++++++++++++++++++++++++++++++++-----------------
 1 file changed, 167 insertions(+), 68 deletions(-)
```

```
commit e2b86790c0b8838b2749d1cf2b72095c783e3a62
Merge: 2157ce8 b00a2c1
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Thu Nov 22 02:29:55 2018 -0500

    finishing merge

commit 247d4328eccc7766a899bd7d44560af45004e681
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Wed Nov 21 23:35:36 2018 -0500

    resolving merge conflicts

 src/ast.mli  | 2 +-
 src/coral.ml | 2 +-
 2 files changed, 2 insertions(+), 2 deletions(-)

commit 027cb8cf9a42846ac1ff638cd2a2166249269f19
Merge: 873e57c b00a2c1
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Wed Nov 21 23:32:48 2018 -0500

    Merge branch 'llvm' of https://github.com/ja3067/Coral into HEAD

commit 873e57cb086a92618ef1bc11e4392358a06a0f11
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Wed Nov 21 23:32:25 2018 -0500

    operators all done

 src/codegen.ml | 100 ++++++++++++++++++++++++++++++++-------------------------
 1 file changed, 57 insertions(+), 43 deletions(-)

commit 4566735570ba8b21b8913675d0a88d9c927fb093
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Tue Nov 20 01:10:55 2018 -0500

    more operator stuff

 src/codegen.ml | 149 ++++++++++++++++++++++-----------------------------------
 1 file changed, 56 insertions(+), 93 deletions(-)

commit 2157ce8988b60b8a50a19a1f24ffd723da7a5b0f
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Mon Nov 19 17:21:33 2018 -0500

    better operators

 src/codegen.ml | 237 +++++++++++++++++++++++++++++++++------------------------
 1 file changed, 138 insertions(+), 99 deletions(-)

commit 6f711b98fbc143fdd15729800f79172e66b5e4e7
Author: Sanford Miller <skm2159@columbia.edu>
```

```
Date:   Mon Nov 19 00:50:37 2018 -0500

    added all of the remaining operators

 src/codegen.ml | 195 ++++++++++++++++++++++++++++++++++++++++++++----------------
 1 file changed, 119 insertions(+), 76 deletions(-)

commit 1ac386683b591e76f0c2b6d7e957354e9f83375d
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Nov 18 21:24:28 2018 -0500

    updated SAST and to_string

 src/sast.ml   |  4 ++--
 src/semant.ml | 19 +++++++++++++-----
 2 files changed, 16 insertions(+), 7 deletions(-)

commit cc7cf369ad18c83bbc4ca67e43726337c4fd7db1
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Thu Nov 15 13:16:14 2018 -0500

    fixed bug with type inference in multiple assignment

 src/sast.ml   | 2 +-
 src/semant.ml | 2 +-
 2 files changed, 2 insertions(+), 2 deletions(-)

commit 026d0e9599dbb5d9960c96cf2a1aa54b2e8b5ae5
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Thu Nov 15 12:02:48 2018 -0500

    pretty printing works

 src/coral.ml  | 2 +-
 src/semant.ml | 2 +-
 2 files changed, 2 insertions(+), 2 deletions(-)

commit fdbaacaf29fd7ba25e2f354c75eb4277a2ca5a12
Merge: 2049f13 bfc4786
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Thu Nov 15 12:02:23 2018 -0500

    Merge branch 'SAST' into pretty-printing

commit 2049f13a562f415301af54096b81bdbcd0294f86
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Nov 14 19:46:09 2018 -0500

    removed extra files

 main      | Bin 8776 -> 0 bytes
 source.ll | 225 ------------------------------------------------------
 source.s  | 257 ------------------------------------------------------------
 3 files changed, 482 deletions(-)
```

```
commit b56f89cbf23950537cad7a2c53c13cd2c7717468
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Nov 14 19:45:51 2018 -0500

    working sast printing with -d flag

 main          | Bin 0 -> 8776 bytes
 source.ll     | 225 ++++++++++++++++++++++++++++++++++++++++++++++++++
 source.s      | 257 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 src/coral.ml  |   5 +-
 src/sast.ml   |   2 +-
 src/semant.ml |   8 +-
 6 files changed, 490 insertions(+), 7 deletions(-)

commit bfc47862fee85fbe2b58e8a8ace76feba76c14e3
Merge: fa8ffc2 1d0b620
Author: Sanford Miller <sanford.miller@columbia.edu>
Date:   Tue Nov 13 22:08:18 2018 -0500

    Merge pull request #1 from ja3067/pretty-printing

    pretty printing works now

commit 1d0b62066c1237636239d3157c048cdda5a3cfbd
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Tue Nov 13 22:06:26 2018 -0500

    pretty printing works now

 src/ast.ml    | 12 ++++++------
 src/sast.ml   | 10 +++++-----
 src/semant.ml |  2 +-
 3 files changed, 12 insertions(+), 12 deletions(-)

commit b00a2c11a96ec48e89fe421baefda62b7d55bb51
Author: Rebecca <rcawkwelL@gmail.com>
Date:   Tue Nov 13 18:38:52 2018 -0500

    updated test suite without fails.

 testall.sh          | 8 ++++----
 tests/fail-add1.cl  | 3 ---
 tests/fail-add1.err | 1 -
 3 files changed, 4 insertions(+), 8 deletions(-)

commit e01b866befe830205d344d06f7d77a52770fb1ce
Author: Rebecca <rcawkwelL@gmail.com>
Date:   Tue Nov 13 18:27:56 2018 -0500

    Added the test suite script and removed the log i accidentally added last time

 testall.log | 437 ------------------------------------------------------------
 testall.sh  | 183 ++++++++++++++++++++++++++
```

```
 2 files changed, 183 insertions(+), 437 deletions(-)

commit d0366a569c6b869cadc0003e7fe2fcb585cc66c7
Author: Rebecca <rcawkwelL@gmail.com>
Date:   Tue Nov 13 18:25:39 2018 -0500

    added testing suite

 Makefile            |  25 ++-
 testall.log         | 437 ++++++++++++++++++++++++++++++++++++++++++++++++++++
 tests/fail-add1.cl  |   3 +
 tests/fail-add1.err |   1 +
 tests/test-add1.cl  |   3 +
 tests/test-add1.out |   1 +
 tests/test-gcd.cl   |  10 ++
 tests/test-gcd.out  |   1 +
 8 files changed, 478 insertions(+), 3 deletions(-)

commit fa8ffc205e71aca46a95ff91262854ab16cb8e8b
Merge: a8e3b6d 61f1505
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Tue Nov 13 03:09:54 2018 -0500

    resolving merge conflicts

commit a8e3b6ddb5031062b363956bfd6303ac4f2fa4e4
Author: Sanford Miller <skm2159@columbia.edu>
Date:   Tue Nov 13 03:08:05 2018 -0500

    pretty printing works until new errors in semant.ml

 .gitignore    |   6 ++++
 src/ast.ml    | 100 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 src/ast.mli   |  36 ----------------------
 src/sast.ml   |  72 ++++++++++++++++++++++++++++++++++++++++++
 src/sast.mli  |  43 ------------------------
 src/semant.ml |   4 ++-
 6 files changed, 181 insertions(+), 80 deletions(-)

commit 61f15058b2d7570fae63c082b9e93ad6ea359768
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Nov 12 00:33:54 2018 -0500

    added comments

 src/sast.mli  | 13 +++++++-----
 src/semant.ml | 11 +++++++---
 test.cl       |  8 --------
 3 files changed, 15 insertions(+), 17 deletions(-)

commit 927850aaf38b3167d05c1da433df245c0560517b
Author: Matt Bowers <mlb2251@columbia.edu>
Date:   Sat Nov 10 18:10:06 2018 -0500
```

```
    first working compiler to llvm just barely capable of compiling gcd. Had too update Makefile and

 Makefile        |   3 +-
 _tags           |   2 +-
 comp.sh         |   1 +
 llvm-test.cl    |  13 ++
 src/ast.mli     |   3 +-
 src/codegen.ml  | 385 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 src/coral.ml    |  14 ++-
 src/parser.mly  |   6 +-
 src/sast.mli    |   1 +
 src/scanner.mll |   3 +-
 src/semant.ml   |   3 +
 11 files changed, 425 insertions(+), 9 deletions(-)

commit e8755f1149413c90cd21a889a0cca1580ee91d7b
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Nov 10 13:33:28 2018 -0500

    added locals to SFunc and returned globals from Semant.check

 src/ast.mli   |  4 +--
 src/coral.ml  |  6 ++--
 src/sast.mli  | 18 +++++------
 src/semant.ml | 96 ++++++++++++++++++++++++++++++++---------------------------
 test.cl       |  8 +++++
 5 files changed, 73 insertions(+), 59 deletions(-)

commit 5772f301ee542f99871db7fdc2c083df54469ecf
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 9 19:36:20 2018 -0500

    added commments

 src/coral.ml  |  7 +++++--
 src/semant.ml | 24 +++++++++++++++++++++---
 2 files changed, 26 insertions(+), 5 deletions(-)

commit fc88f8211c7778e5376b5fca2e07aa6e8611d8e8
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Nov 9 12:31:54 2018 -0500

    added -c option to check

 src/coral.ml  | 13 ++++++-------
 src/semant.ml |  2 +-
 2 files changed, 7 insertions(+), 8 deletions(-)

commit ef582d9d496bb955adaf5af71335472c2c19eceb
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Thu Nov 8 20:19:27 2018 -0500

    removed test file
```

```
 test.cl | 4 ----
 1 file changed, 4 deletions(-)

commit 9f199e6281fd1086fc80eaf80033fbf3d8f17ff1
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Thu Nov 8 20:13:00 2018 -0500

    fixed bug with return statements inside loops

 src/semant.ml | 26 ++++++++++++++++++++++++--
 1 file changed, 24 insertions(+), 2 deletions(-)

commit 20d79afd3a0730f49f37301f0428c249b036a76c
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Thu Nov 8 19:57:20 2018 -0500

    can now run files with -f [path]

 src/coral.ml  | 52 +++++++++++++++++++++++++++++++++++++++++++++-------
 src/sast.mli  |  4 ++--
 src/semant.ml |  7 +++++--
 test.cl       |  4 ++++
 4 files changed, 56 insertions(+), 11 deletions(-)

commit c0cf3660ea09e4ccad7a6c7084b5f8fc213f49b6
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Thu Nov 8 00:42:49 2018 -0500

    type inference for functions and calls. dont try recursion

 src/ast.mli    |  2 +-
 src/parser.mly |  2 +-
 src/sast.mli   | 30 ++++++++++---------
 src/semant.ml  | 95 +++++++++++++++++++++++++++++++++++++++++++++-------------
 4 files changed, 91 insertions(+), 38 deletions(-)

commit e4feaaeae78341e2fb436fc9591ab4326608ce19
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Tue Nov 6 17:50:34 2018 -0500

    rudimentary type inferrence finished for everything except functions

 src/ast.mli   |   2 +-
 src/coral.ml  |   1 -
 src/sast.mli  |  14 ++++--
 src/semant.ml | 151 +++++++++++++++++++++++++++++++++++++++---------------------
 4 files changed, 108 insertions(+), 60 deletions(-)

commit 9062c04432f4b17633108223cbc3dbf718104b10
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Tue Nov 6 01:53:22 2018 -0500

    added type command for debugging convenience, parsed as NOP
```

```
 src/ast.mli     |  1 +
 src/coral.ml    |  1 +
 src/parser.mly  |  4 +++-
 src/scanner.mll |  1 +
 src/semant.ml   | 13 +++++++++++++
 5 files changed, 19 insertions(+), 1 deletion(-)

commit e37964a1c08ed1a4bbd2ef16ed9f5cd42dac74df
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Tue Nov 6 01:25:28 2018 -0500

    fixed typed arrays, added more type inference. still need to fix typing in for and while loops

 src/semant.ml | 30 ++++++++++++++++--------------
 1 file changed, 16 insertions(+), 14 deletions(-)

commit 9541375cd32b0b728876cf1667d8aa3c599158eb
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Tue Nov 6 00:49:12 2018 -0500

    basic type inference works

 src/sast.mli  | 2 ++
 src/semant.ml | 6 ++++--
 2 files changed, 6 insertions(+), 2 deletions(-)

commit dfc82ded7539e0b35a54e786d01c05236fbca4e1
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Tue Nov 6 00:17:00 2018 -0500

    binops mostly work

 src/coral.ml  |  2 +-
 src/semant.ml | 26 ++++++++++++++++++++------
 2 files changed, 21 insertions(+), 7 deletions(-)

commit a7fb31c3b9254748ac39f62a92b667e7da9e337e
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Nov 5 23:19:42 2018 -0500

    semant partly done

 _tags          |   2 +-
 src/ast.mli    |  12 +++---
 src/coral.ml   |  55 ++++++++++++------------
 src/parser.mly |  21 ++++-----
 src/sast.mli   |  51 ++++++++--------------
 src/semant.ml  | 132 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 6 files changed, 194 insertions(+), 79 deletions(-)

commit b69b7ecd4c603f02017abeb1408a38201e961f1a
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Nov 4 19:54:09 2018 -0500
```

```
    added blocks, cleaned up parser started sast and semant

 src/ast.mli    | 14 +++++++-------
 src/coral.ml   |  6 +++---
 src/parser.mly |  4 ++--
 src/sast.mli   |  8 ++++----
 4 files changed, 16 insertions(+), 16 deletions(-)

commit 631589e258c5feb735ae78dbb0ac3d181b22e4b7
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Nov 4 14:48:14 2018 -0500

    working

 src/sast.mli  | 52 ++++++++++++++++++++++++++++++++++++++++++++++++++++
 src/semant.ml |  0
 2 files changed, 52 insertions(+)

commit dfa7a44c0550857b0d9b038ebc927dbeba7729ef
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Nov 4 14:21:59 2018 -0500

    working parser

 src/ast.mli     | 12 ++++++------
 src/coral.ml    | 12 ++++++------
 src/parser.mly  | 37 ++++++++++++++++++++++---------------
 src/scanner.mll |  1 +
 4 files changed, 35 insertions(+), 27 deletions(-)

commit 42259ad26abdee3e6a8a6ceff4af18c50ec10817
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Nov 4 13:40:35 2018 -0500

    added first SAST steps

 .gitignore                             |   0
 Makefile                               |   2 +-
 README.md                              |   0
 _tags                                  |   0
 docs/Coral Language Reference Manual.pdf | Bin
 src/ast.mli                            |  11 ++++--
 src/coral.ml                           |  17 ++++++--
 src/getopt/getopt.ml                   |   0
 src/getopt/getopt.mli                  |   0
 src/parser.mly                         |  65 ++++++++++++++---------------
 src/scanner.mll                        |  12 ++++--
 11 files changed, 62 insertions(+), 45 deletions(-)

commit db382aaa9b89836bb17dfbc74793d7a7314d2af7
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Fri Oct 26 12:18:23 2018 -0400

    Update README.md
```

```
 README.md | 18 +++++++++++++++---
 1 file changed, 15 insertions(+), 3 deletions(-)

commit c0ed31c5b7ab60e510afaa12480fbd710442f027
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Oct 24 18:24:43 2018 -0400

    Update README.md

 README.md | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)

commit bcdc05d78750c499b5fef71e10ae262b92d9511f
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Oct 24 18:22:27 2018 -0400

    reorganized code

 Makefile                                                  |   2 +-
 .../Coral Language Reference Manual.pdf                   | Bin
 {getopt => src/getopt}/getopt.ml                          |   0
 {getopt => src/getopt}/getopt.mli                         |   0
 4 files changed, 1 insertion(+), 1 deletion(-)

commit d29000330bcbb90cf2fff26eb40a61f402ab77e2
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Oct 24 18:21:01 2018 -0400

    removed LLVM dependency temporarily

 _tags | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 2aacd49f7a1fbbbf75f9db58af66f7b67d3a5ed3
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Oct 24 18:17:56 2018 -0400

    added makefile, file structure

 Makefile                       | 15 +++++++++++++++
 _tags                          |  5 +++++
 getopt.ml => getopt/getopt.ml  |  0
 getopt.mli => getopt/getopt.mli |  0
 ast.mli => src/ast.mli         |  0
 coral.ml => src/coral.ml       |  0
 parser.mly => src/parser.mly   |  0
 scanner.mll => src/scanner.mll  |  0
 8 files changed, 20 insertions(+)

commit b512edb13ac876c27955d2006a039eeed18633bf
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Oct 24 17:08:59 2018 -0400
```

```
    commented the code better

 coral.ml | 44 +++++++++++++++++++----------------------------
 1 file changed, 15 insertions(+), 29 deletions(-)

commit dc427334391a71d950163066713dd80ff536b512
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Oct 24 16:48:33 2018 -0400

    fixed a small bug with printing and ebugging

 coral.ml | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 49a5665272375e2073a8dc7153b3d78d27ac4286
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Wed Oct 24 16:47:36 2018 -0400

    added debug tool, run with ./coral.native -d

 coral.ml   |  15 ++++++--
 getopt.ml  | 106 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 getopt.mli | 120 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 3 files changed, 239 insertions(+), 2 deletions(-)

commit 50f3ecfbf8d43eeea7ee6afd01ebcf390afd9df8
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Oct 15 23:14:10 2018 -0400

    modified scanner to removed ! negation token, added the reference manual

 Coral Language Reference Manual.pdf | Bin 0 -> 189613 bytes
 scanner.mll                         |   2 +-
 2 files changed, 1 insertion(+), 1 deletion(-)

commit a754cf674efae75a252ced0eb915d7bd4a64e937
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Oct 15 17:12:35 2018 -0400

    added class fields and methods, added periods.

 ast.mli     | 2 ++
 coral.ml    | 3 +++
 parser.mly  | 8 ++++++--
 scanner.mll | 9 +++++----
 4 files changed, 16 insertions(+), 6 deletions(-)

commit 304a6ae8fc52a70a3709d50d7ac1c75c81e69823
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Oct 15 16:28:13 2018 -0400

    removed debug message, updated float regex token

 coral.ml    | 2 +-
```

```
 scanner.mll | 4 ++--
 2 files changed, 3 insertions(+), 3 deletions(-)

commit 17208ff9a60544db6c17eecf4bc6dd52c3ca25b6
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Oct 15 11:31:41 2018 -0400

    fixed bugs with semicolons, updated parser, should be ready to submit

 coral.ml    |  5 +++--
 parser.mly  | 34 ++++++++++++++++++++++++++++++--------
 scanner.mll |  8 +++++++--
 3 files changed, 35 insertions(+), 12 deletions(-)

commit 4b83b86dff534b13699c0e52a329193e076fd245
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 14 22:57:14 2018 -0400

    removed debugging notice

 coral.ml | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit dd1225094d12309c82203387ccd3d703662d76ce
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 14 22:56:22 2018 -0400

    added support for strings, floats, ints, and bools to the parser

 ast.mli     |  8 +++++++-
 coral.ml    | 13 +++++++++----
 parser.mly  |  9 +++++----
 scanner.mll |  2 +-
 4 files changed, 22 insertions(+), 10 deletions(-)

commit 72063cad448198c7904605083b0aab6c7d46ae77
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Oct 8 22:51:04 2018 -0400

    final changes before meeting, made errors more consistent, etc

 coral.ml    | 16 ++++++++++++++++
 parser.mly  |  2 +-
 scanner.mll | 12 ++++++++++++
 3 files changed, 15 insertions(+), 15 deletions(-)

commit 8e6254e0f9d0c1bab9e0576624f5ee7a7c655b92
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Oct 8 22:43:42 2018 -0400

    everything added except for type hints and final typed variable support

 coral.ml    |  5 ++++-
 parser.mly  | 20 ++++++++++++++++++++
```

```
 scanner.mll | 22 ++++++++++-----------
 3 files changed, 29 insertions(+), 18 deletions(-)

commit a082456b1d6cf6c08ad7b95368f1ebc853b75252
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Oct 8 22:13:18 2018 -0400

    added lists, commented out types

 ast.mli     |  2 ++
 coral.ml    |  5 ++++-
 parser.mly  | 16 ++++++++++++++--
 scanner.mll | 26 ++++++++++++++++++++++++-----
 4 files changed, 41 insertions(+), 8 deletions(-)

commit 807c6b089726a21b49195ef985b410272667c03f
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Mon Oct 8 19:06:05 2018 -0400

    added all python keywords

 coral.ml    |  4 ++--
 parser.mly  |  2 +-
 scanner.mll | 10 +++++++++-
 3 files changed, 12 insertions(+), 4 deletions(-)

commit 118d2c83976854d26ec004975dde8a40cf1b6162
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 7 14:09:20 2018 -0400

    Update README.md

 README.md | 4 ++++
 1 file changed, 4 insertions(+)

commit 1103513635a3182e1308f7f670de8a50015851c3
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 7 09:24:18 2018 -0400

    Update README.md

 README.md | 14 +-------------
 1 file changed, 1 insertion(+), 13 deletions(-)

commit b92a9061694241c475c8d574c083565aadc87398
Merge: 36d1410 d1c5671
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 7 09:15:28 2018 -0400

    Merge branch 'master' of https://github.com/ja3067/Coral

commit 36d1410f7232041c9aff94398c8d7e23392e3d95
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 7 09:15:24 2018 -0400
```

```
    resolved all shift/reduce conflicts by adding precedence tokens for every token. may cause unfor

 parser.mly | 14 ++++++++++----
 1 file changed, 10 insertions(+), 4 deletions(-)

commit d1c5671bf874a1ecb43a9d5245c2025c08e176c8
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 7 01:11:28 2018 -0400

    Update README.md

 README.md | 14 ++++++++++++++
 1 file changed, 14 insertions(+)

commit e6fca0aaa65d633fe91f53e3cf363fc5734f13a4
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 7 00:45:26 2018 -0400

    Update README.md

 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 4b4e787d9f182ab8de18744c6e4db1b4cd04ba1a
Merge: c48ee24 8d426d1
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 7 00:43:14 2018 -0400

    Merge branch 'master' of https://github.com/ja3067/Coral

commit c48ee24caedd6e4c53cd2643c0ec6e971c3c9da5
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 7 00:43:10 2018 -0400

    added return statements

 coral.ml | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 8d426d1192dccd3bb40fb0bfcb28590f34b039e7
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 7 00:38:58 2018 -0400

    Update README.md

 README.md | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)

commit 161b1bbfc81c0a59c032c410577d6ddffeb428ad
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 7 00:28:24 2018 -0400

    Update README.md
```

```
 README.md | 16 +++++++++++++++-
 1 file changed, 15 insertions(+), 1 deletion(-)

commit 907364517916dec122c81af607a5cdc77cec63a3
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 7 00:26:19 2018 -0400

    added comments to coral.ml

 coral.ml | 53 ++++++++++++++++++++++++++++++++++++++++++++++++++-----------
 1 file changed, 42 insertions(+), 11 deletions(-)

commit 2a1fe564f0665c91e8bd0155e963d969abb28174
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 7 00:15:27 2018 -0400

    Update README.md

 README.md | 6 +++++-
 1 file changed, 5 insertions(+), 1 deletion(-)

commit 6d5bedeb56f207ce7cec00d5ec275b221bc9cec0
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 7 00:09:16 2018 -0400

    Update README.md

 README.md | 40 ++++++++++++++++++++----------------------
 1 file changed, 20 insertions(+), 20 deletions(-)

commit d9c9d509035247c53fce85071d98f56ca732fe88
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 7 00:08:50 2018 -0400

    Update README.md

 README.md | 20 ++++++++++++++++++++
 1 file changed, 20 insertions(+)

commit b85c0981ca2eab750498f374f71d1c4fc2c26bb7
Merge: 2bfca8a ee4d793
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 7 00:02:37 2018 -0400

    Merge branch 'master' of https://github.com/ja3067/Coral

commit 2bfca8a587ddc52477c5d16b5be59030d96a24fe
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sun Oct 7 00:02:31 2018 -0400

    working version with functions, while loops, if statements, and nearly full parsing. there are s

 ast.mli     |  8 ++++----
```

```
 coral.ml    | 43 ++++++++++++++++++++++++++-----------------
 parser.mly  | 13 +++++++++----
 scanner.mll |  4 ++++
 4 files changed, 43 insertions(+), 25 deletions(-)

commit ee4d793da84d87f2f67a0824d64d9fcefd7a1d6a
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Oct 6 19:52:48 2018 -0400

    Update README.md

 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit b4c37209c4294df4097dfefc3ab829e147f219c4
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Oct 6 19:47:40 2018 -0400

    Update README.md

 README.md | 16 ++++++++++++++++
 1 file changed, 8 insertions(+), 8 deletions(-)

commit 9e9a464dadde0c07e2342474243e322fdc667b57
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Oct 6 19:46:27 2018 -0400

    Update README.md

 README.md | 104 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 104 insertions(+)

commit a22ada87f0bf0e0692c77fce27c3f25a8c453c83
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Oct 6 19:39:52 2018 -0400

    Update README.md

 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit f7212e15d84317b9eb60cd7dfa16b5cedcb324fc
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Oct 6 19:39:16 2018 -0400

    Update README.md

 README.md | 42 ++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 42 insertions(+)

commit 192aaaf68cf5274de629c79961d9102c9c8a0dbf
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Oct 6 19:28:07 2018 -0400
```

```
    initial commit with mostly working parser

 ast.mli     |  20 +++++++
 coral.ml    | 171 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 parser.mly  | 135 +++++++++++++++++++++++++++++++++++++++++++++++
 scanner.mll |  54 +++++++++++++++++++
 4 files changed, 380 insertions(+)

commit 937199e7a8f90254c2372f286a53cab08b9876c3
Author: Jacob Austin <ja3067@columbia.edu>
Date:   Sat Oct 6 19:26:53 2018 -0400

    Initial commit

 .gitignore | 23 +++++++++++++++++++++++
 README.md  |  2 ++
 2 files changed, 25 insertions(+)

\end{verbatim}
```