Lingo

Final Report

Sophia Danielle Kolak - sdk2147
Jay Karp - jlk2225
Benjamin Flin - brf2117

April 2021

Contents

1 Introduction

2 Language Tutorial

2.1 Syntax

lingo uses an extremely verbose syntax that allows for specificity in function and type declarations. It is similar to ocaml and c in its use of type declarations and let statements as well as semi-colon deliminated expressions. Every lingo program must include a `main` function as well, similar to c and this main function must return an Int. Our first lingo program will be extremely simple: just adding two numbers and printing them.

```
print_int : Int -> Int;
main : Int = print_int (10 + 15);
```

While this function may look simple, there is actually a lot going on here! First, we have an external function declaration, print_int. In lingo you can link c functions into your programs by declaring them in a library file. After this all you need to do is give them a type declaration, and you are free to use them throughout your program. As you can see from this example, print_int takes in an integer and returns an integer (as is required by the main function of all lingo programs). Type declarations like the one made for print_int or main are written using a semi-colon `:` followed by the type. For now, application is denoted by the `->` arrow, however, as you will see later this gets a little more complicated. Finally, all function declarations must end in semi-colons.

2.2 Typing

Typing in lingo is extremely verbose and allows for multiple cool features. It is also very similar to Haskell or Ocaml in terms of syntax. One of the best features of lingo is its ability to have both type and multiplicity polymorphism. In the following example, we will see an instance of both of these different types of polymorphism and what they do.

```
id a c x : @a #c a -c> a = x;
main : Int = id @Int #Unr 10;
```

Again, there is a lot of stuff going on in this simple example as well. Here we have defined a global function id which takes in three arguments, a, c, and x. Here, a

is a polymorphic type variable. This means that a can take in multiple different type values for a and still be well-typed. In a similar way c here is a polymorphic multiplicity variable. This can have either a value of `#Unr` or `#One`. If id is passed an `Int` as the polymorphic type variable, id must take in and return an `Int`. If id is passed `#Unr` this means that x can be used as many times as necessary in id, however, if id is passed `#One`, x can only be used linearly in id. You must specify the type variables when calling a function like id. Types use the `@` and multiplicities use the `#`.

2.3 Data Types and Lets

Finally, lingo also has rich support for abstract data types and let in statements. They can be used in the following way:

```
print_int : Int -> Int;
data Tuple a b #p #q where
    Tuple : a -p> b -q> Tuple a b #p #q;

main : Int =
    let x = 10 in
    print_int x;
```

Here we have defined a simple ADT for the tuple, which takes in two types and two multiplicities for each of the variables usages. In the main function, we can see the use of a let in statement which allows for the value of x to be used when printing.

3 Language Manual

3.1 Introduction

The Lingo programming language is a functional programming language with rank-n polymorphism and linear types. Our language is heavily inspired by Haskell's implementation linear-core[1], however, we extend some aspects of Haskell's and Ocaml's syntax with a few features such as a specialized syntax for linear arrows. We choose OCaml as inspiration for our syntax design because it is compact and

[1] https://arxiv.org/pdf/1710.09756.pdf

easy to understand. Unlike Haskell's linear core, however, we wanted the design of our language to highlight linearity as primary feature.

3.2 Lexical Conventions

There are six different kinds of tokens: identifiers, keywords, constants, binary operators, arrows and other special tokens. These categories of tokens are not disjoint. Similar to C, blanks, tabs newlines and comments are ignored except for their use in separating tokens where at least one is required.

3.2.1 Comments

The characters `(*` introduce a comment. The characters `*)` terminate the comment. Lingo does not allow single-line comments or nested multi-line comments

```
(*
    This is a comment in Lingo, how cool.
*)
```

3.2.2 Identifiers

Lowercase identifiers must begin with a lowercase ASCII character. The rest of the identifier string can be any combination of letters (uppercase or lower-case), digits and underscores. Identifiers can also end with any number of single quotes `'`. Uppercase identifiers follow the same rules as lowercase identifiers except that they must start with an uppercase ASCII character.

```
foo' : Int = 10;
(* foo' is a lowercase identifier *)
(* Int is an uppercase identifier *)
```

3.2.3 Keywords

The following keywords are reserved, and cannot be used as identifiers.

`if, then, else, let, in, data, case, of, where, Unr, One`

3.2.4 Integer Constants

Integer constants consist of a sequence of characters from 0 through 9. Note that integers are sized to 64 bits, so the value of any integer literal x is $x \mod 2^{64}$. The matching regex is `['0'-'9']+`

```
integer : Int = 42;
```

3.2.5 Character Constants

Character constants are a single upper or lowercase character surrounded by single quote as well as any special characters excluding single quote `'`. We also include escape sequences `\n, \t, \r, \000` for newline, tab, carriage return and null respectively.

```
a : Char = 'a';
b : Char = '\n';
```

3.2.6 String Constants

A string constant can be any sequence of characters surrounded with ". The rules for which characters a string may contain are the same as that for characters (except " is excluded instead of '). Note that the following data structure must be defined, otherwise strings will not work. This is because strings are syntactic sugar for this data structure, as in the following:

```
data String where
    C : Char -> String -> String;
    E : String;
a : String = "hi";
b : String = "\t\n";
a' : String = C 'h' (C 'i' E); (* same as a *)
```

3.2.7 Boolean Constants

Boolean constants are characters with either a value of `true` or `false`.

```
coolbool : Bool = true;
```

3.2.8 Special Identifiers

3.2.9 Unit

If the characters `()` appear in sequence then they form a token called unit. Otherwise, they form two separate tokens left parentheses `(` and right parentheses `)`. Note that the unit data structure must be defined, as in the following:

```
data Unit where
    Unit : Unit;
unit : () = (); (* unit *)
foo : Int = (3 + 4); (* parentheses (not unit) *)
```

3.2.10 Forall and ForallM

The tokens `@` and `#` are called forall and forallm respectively form two tokens. Note that each of these tokens can and often appear next to an identifier without whitespace.

```
id a p x : @a #p (a -p> a) = x;
```

3.2.11 Operators

Lingo supports the following tokens as Operators.

```
!= Not Equal
<= Less Than or Equal to
<  Less Than
>= Greater Than or Equal to
>  Greater Than
== Equal
|| Or
&& And
*  Multiplication (a.k.a. Star)
/  Division (a.k.a. Slash)
+  Addition
-  Minus or Negate (a.k.a. Dash)
!  Not
```

The greater than token `>`, the dash token `-`, the slash token `/` and the star token `*` are all tokens which are also used in syntax outside the context of binary and unary operators, as will be discussed later in this paper.

```
x : Int = 10 * 10 / 10 + 10 - 10
```

We will discuss each of these operators behaviors and precedence later in the paper.

3.2.12 Separating and other tokens

The backslash token \, the wildcard token _ the colon `:`, the `.` token and the semicolon `;` are the remaining tokens not present elsewhere in this document.

```
data PartyAnimal a #p where
    Cow     : a -p> PartyAnimal;
    Giraffe : a -p> PartyAnimal;
    Hyenas  : a ->  PartyAnimal;
    Donkeys : a ->  PartyAnimal; (* both : and ; tokens *)
foo : Int = case (Just 4) of
    Just a -> a + 1;
    _ -> 0; (* wildcard *)
;
bar : Int -> Int = \x . x;
```

3.3 Types

Types are written after certain expressions and top-level statements, and are usually separated from the expression/top-level statement using a colon.

```
foo : Int = 0;
```

Since Lingo has no type inference, types are needed in most expressions.

There are three primitive types in Lingo: Int, Char, Bool. There is an infix type operator `a -> b` which represents functions from types a to b. This binary operator on types is right associative.

```
f x : Int -> Int = x;
```

Lingo does support polymorphic types. This allows functions to be variable in the types that they accept. In order to deal with polymorphism, lingo uses System F. In System F, there is a abstraction for binding type variables, and an application for applying them. In lingo, type abstraction uses the same syntax as value abstraction. This means that a type variable is treated exactly the same as any other

variable in terms of how they are bound. Type application is denoted using `@` before the type. The type of a type abstraction is denoted using `@` before a variable name and followed by another type with may or may not use that type variable. Note that in the following example the bound type variables `a` and `b` are the same as the quantified variables `@a` and `@b` in the type, but in general they do not have to be the same.

```
foo a b f x : @a @b (a -> b) -> a -> b = f x;
foo1 : Int = foo @Int @Int print_int 10;
foo2 : Int = foo @Char @Int toInt '1';
```

Type application can also use other type variables, as in the following example:

```
data Tuple a b where
    Tuple : a -> b -> Tuple;
id a x : @a a -> a = x;
tup b c x y : @b @c (@a a -> a) -> Tuple b c
    = Tuple (id @b x) (id @c y);
```

3.4 Multiplicities

Multiplicities describe how an argument is evaluated inside an expression. A multiplicity is either `One` or `Unr`. If a multiplicity of a variable is `One`, then the argument is evaluated in the expression exactly once. If it is `Unr` than can be evaluated any number of times, including zero.

The following expression is well-typed:

```
let #One x : Int = 10 in x + 1
```

Arrows can be annotated with a multiplicity. The arrow `(->)` by default has multiplicity `Unr` and `(-*)` has multiplicity `One`.

Multiplicities can also be polymorphic over multiplicites. Introduction of multiplicity variables is done exactly the same as type variables, and multiplicity quantifiers appear in types prepended with a `#` token. In arrows, the multiplicity variables appear infix. For example:

```
id a p x : @a #p (a -p> a) = x;
```

Multiplicity application is like type application, except it is prepended with a `#` rather than a `@`.

Multiplicities can also be multiplied. For concrete multiplicities `One` and `Unr`, `One * One` is equivalent to `One` and all other cases are `Unr`. Multiplication of multiplicities is associative and commutative by construction.

For example:

```
compose p q a b c f g x :
    #p #q @a @b @c
    (b -q> c) ->
    (a -p> b) ->
    a -p*q> c = f (g x);
```

The above is well typed because `g` uses its argument `p` times and `f` uses its argument `q` times. Therefore, `x` is used `p*q`.

Below is a more comprehensive grammar for type, multiplicites and arrows:

```
<type> :=  @ <lowercase-id> <type>
        |  # <lowercase-id> <type>
        |  <type> <arrow> <type>
        |  <type> <type>
        |  <type> # <mult>
        |  <lowercase-id>
<arrow> :=  ->
        |   -*
        |   - <mult> >
<mult> := <lowercase-id>
        | Unr
        | One
        | <mult> * <mult>
```

3.5 Expressions

We use the following syntax for specifying grammars. A terminal/token is either written by the characters from which it is constituted or by using a self-descriptive name like `integer-literal`. Non-terminals are given with angle brackets, such as `<expr>`. A production rule is given using `<non-terminal> := token <other-non-terminal> ...` Anything contained in square brackets `[]` is considered optional.

3.5.1 Lambda expressions

There are three types of lambda expressions, one for values, types and multiplicities. We use \ to denote all of them. Lambda expression:

```
<expr> := \ lowercase-id . <expr>
```

For example:

```
\x. x + 1
```

Note that, for lambda expressions to be parsed, they must either include an annotated with a type in a `let in` expression, in a top level declaration or by themselves appended with a type with a colon separating it and surrounded by parentheses. For example:

```
main : Int = (\x. print_int x : Int -> Int) 0;
```

3.5.2 Let-in expressions

A let-in expression allows for binding a new variable equal to an expression inside another expression.

```
let x : Int = 5 in x + 3
```

The above let-in expression is equivalent in terms of operational semantics to the following expression:

```
(\x -> x + 3 : Int -> Int) 5
```

A let-in expression can optionally take in a list of parameters. (Note that a param-list is any number of params separated by whitespace, and in general, a x-list is any number of xs surrounded by whitespace).

```
<expr> := let [# <mult>] <lowercase-id> <lowercase-id-list> : <type>
    = <expr> in <expr>
```

The type parameter of the let-in expression is the type of the bound name given as the lowercase identifier after the `let`.

Each parameter enclosed inside vertical bars, curly braces, or no enclosure represent multiplicity abstraction, type abstraction and value abstraction respectively. To illustrate, the following represents pairs of let-in expressions which are equivalent:

```
let id x : Int -> Int = x in id 0
let id : Int -> Int = \x -> x in id 0

let id a x : @a a -> a = x in id 0
let id : @a a -> a = \a. \x. x in id @Int 0

let id a p x : @a #p a -p> a = x in id @Int #Unr 0
let id : @a #p a -p> a = \a. \p. \x. x in id @Int #Unr 0
```

3.5.3 If expressions

An if expression takes the form

```
if <expr> then <expr> else <expr>
```

Note that the else is mandatory. The first expression after the if is expected to have a type `Bool`.

3.5.4 Case expressions

Case expressions are syntactically similar to those in Haskell, however, in Lingo each case expression is concluded with a `;`, they take the following form:

```
<expr> := case <expr> of <casealts>
```

Casealts is a representation of all of the different case alternatives that are involved in matching. They take the form:

```
<case-alt> := uppercase-id <lowercase-ids> -> <expr>;
            | _ -> <expr>;
```

Each Casealt statement is made up of an uppercase identifier which allows for argument deconstruction, a list of lowercase identifiers in sequence to pattern match on, as well as the resulting expression to be evaluated when matching to that case. We do not support deep pattern matching for individual cases. e.g.

```
x : Maybe (Maybe Int) #One = case (Just @Int #One 4) of
    Just a -> Just a;
    _ -> Just 0;
;
```

3.5.5 Binary expressions

Binary expressions take the form

```
<expr> <binop> <expr>
```

Where the binary operators are:

```
||, &&, ==, !=, <, >, <=, >=, +, -, /, *, !
```

The operators are presented in order of least to most precedence, where the following groups of operators each have equal precedence and are left-to right associative.

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

In this context, * means multiply, / means divide, and - means subtract. Note that we label these tokens as Star, Slash, and Dash in section 3.2.11 because each of these tokens are used in other parts of Lingo's syntax.

3.5.6 Application expressions

Lingo supports application of multiple different kinds. This includes type application, multiplicity application and function application. The following two expression are logically equivalent a display how application works in Lingo. Application is also left associative by default.

```
let foo a b c p q f g x :
    #p #q @a @b @c  (a -p> b) -> ((a -p> b) ->
        (b -q> c)) -> a -p*q> c
    = f g x in
foo #Unr #Unr @Int @Char @Int

let foo' a b c p q f g x :
    #p #q @a @b @c  (a -p> b) -> ((a -p> b) ->
        (b -q> c)) -> a -p*q> c
    = (f g) x in
foo' #Unr #Unr @Int @Char @Int
```

3.5.7 Literal expressions

Expressions can consist of a single lowercase id which can refer to a variable. Expressions can also consist of a integer literal token as well as a character token described in sections 3.2.4 and 3.2.5 respectively.

3.5.8 Precedence of Expressions

Expressions described in this section are generally listed in order of lowest to highest precedence, where the first four (sections 3.5.1-3.5.4) are of equal precedence. Application has a higher precedence than any binary operator, for example. The following expression

```
f a + g a
```

is equivalent to

```
(f a) + (g a)
```

Any expression surrounded with parentheses has highest precedence.

3.6 Algebraic Data Types

Algebraic Data Types allow Lingo to support sum types as well as product types. These data types can take in both type and multiplicity parameters.

```
data Tuple a b #p where
    Tuple : a -p> b -p> Tuple;

data Maybe a #p  where
    Just    : a -p> Maybe;
    Nothing : Maybe;

x : Maybe @Int #Unr = Nothing;
y : Maybe @Int #Unr = Just 10;
```

Here each case, which are separated by semicolons, represents a sum type. In this example, a Tuple represents a product type. `a` and `b` represent type variables and `p` represents a multiplicity variable. We also do not allow quantifiers in each case definition. We can instantiate the type and multiplicity variables in Algebraic Data Types using the above syntax.

3.7 Types Revisited and Kinds

Kinds describe the 'type' of types. A kind can be either `Mult` or `Type` or an `(->)` from any two kinds. Concrete types such as `Bool` or `Maybe Int` have a kind `Type`. All arrows `(->)` are infix operators with kind

```
Type -> Mult -> Type
```

which means `(->)` can produce a type if given two other types. The following definition

```
data Maybe @a #p where
    Just    : a -p> Maybe;
    Nothing : Maybe;
```

has kind

```
Type -> Mult -> Type
```

We can instantiate any kind of the form

```
Type -> k
```

where `k` is any kind by application with curly braces, and we can instantiate a kind of the form

```
Mult -> k
```

with vertical bars. Using the definition for Maybe given above, we can build the following:

```
x : Maybe @Int #Unr = Just 1;
```

In Lingo, all kinds are inferred, so there is no syntax for specifying the kind of a type. However, kinds are still useful concept in reasoning about types.

3.8 Top-level Statements

A top-level statement can consist of an identifier followed optionally by parameters and a type annotation assigned to some expression and then terminated with a semicolon.

```
id a p x : @a #p a -p> a = x;
foo : Maybe Int #Unr = Just @Int #One 1;
```

In general, a let statement follows the same syntax as a let-in expression, with the `let`, `in` and the second <expr> omitted.

3.9 Top-level Declarations

A top-level declaration is like a top-level statement without its right hand side. This is how external functions are declared in lingo. For example,

```
print_int : Int -> Int;
```

3.10 Programs

A program consists a sequence of top-level statements, declarations and algebraic data type definition. In a program, all top-level statements are automatically mutually recursive, so the order in which they are defined is irrelevant. The entry point for a program is a let statement with the identifier `main` of type `Int`. For example,

```
print_int : Int -> Int;
main : Int = print_int 0;
```

3.11 Example Program

Below is an example of a valid program:

```
data Maybe a #p where
  Just    : a -p> Maybe a #p;
  Nothing : Maybe a #p;

foo : Int -> Maybe Int #One
    = \x. Just @Int #One x;

foo' m
  : Maybe Int #One -> Int
  = case m of
    Just i -> i;
    _ -> 0;
  ;

print_int : Int -> Int;
main : Int = print_int (foo' (Just @Int #One 1));
```

4 Project Plan

The development of lingo had many components which we were initially unsure of how to solve. This forced our development process to be interactive and intentional. We met every week with professor Edwards and also had a weekly briefing and overview of what we wanted to accomplish for that week. We also had a weekly development session which allowed us to talk through and explore and understand some of the more difficult aspects of our compiler such as the typechecker. Overall, we had multiple main milestones which corresponded to the different segments of the compiler architecture. While there was some overlap in segment development in order to reach these milestones, we generally stuck to completing earlier steps in the pipeline before later ones.

4.1 Planning, Spec, Dev, and Testing

The main method we used to plan out our weekly development were our meetings with Professor Edwards. These meetings solidified what we needed to get done each week through our discussion and help from Professor Edwards. From here, during our weekly meetings we split up the tasks based on what we thought would be beneficial to pair program together, and which segments we though we

could do individually. Often times we used pair programming to complete more difficult initial implementations and split the work from here after we had a solid understanding of the fundamentals of that component of the compiler.
Testing was another key component of our compiler as it allowed us to move forward and be confident in the work that we were completing. As we only finished the entire compiler pipeline in the last month of development, our original tests were simple unit tests that only tested on aspect of the compiler. As our compiler matured and out entire pipeline was complete (initially completion for the hello world assignment). We began to write regression tests to test more than one component of the compiler. These tests helped us as we continued development because it allowed us to know which segments of the compiler were causing tests to break and on top of this forcing bugs from previous pieces of the pipeline to be recognized even in the later stages of development. Finally, we implemented a githook which forced all the tests to be passing before a commit. This forced all of us to make sure that our changes were not breaking any other tests.

4.2 Style Guide

1. Code should not be more than 80 lines
2. We only ever use snake case for all variables
3. Variable names should be as descriptive as necessary in order to conserve readability
4. Use two spaces for indentation.

4.3 Project Timeline

Team Member	Responsibility
Feb 3	Language Proposal
Feb 24	Language Reference Manual, Scanner, Parser
March 3	Testing Framework
March 24	Type Checker and Hello World
April 7	Monomorphisation and Closure Conversion
April 21	Code Generation
April 26	Extra Debugging, Presentation and Final Report

4.4 Roles and Responsibilities

Team Member	Responsibility
Sophia Kolak	Unit, Regression Testing, Compiler Front-end, Code-Gen
Ben Flin	Typechecker, Closure Conversion, Monomorphization, Code-Gen
Jay Karp	Testing Environment, Unit, Regression Tests, Code-Gen, C Interop

4.5 Development Environment

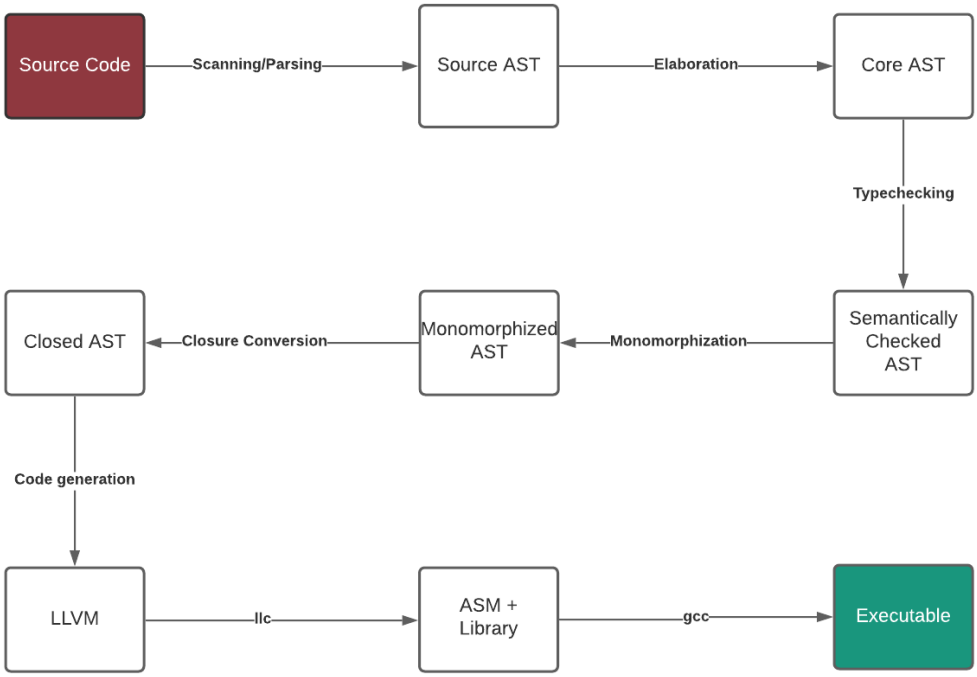
We had the following programming and development environments:

1. Custom Docker Container version 20.10.5, build 55c4c8
2. Ocaml version 4.11
3. LLVM version 10.0.0
4. Dune version 2.8.5 (Ocaml Build Environment)
5. Python3 (testing)
6. VS Code with Ocaml Platform Extension

4.6 Git Log

*See Appendix

5 Architectural Design



1. Elaboration: During elaboration, all syntax is expanded into a form which the typechecker can understand. This includes converting all variable names to Debruijn indices, each lambda into their corresponding abstractions (value, type, or multiplicity) and variable lists into nested lambdas. This was mostly authored by Ben.

2. Typechecking: The typechecking recieves a core ast and produces a semantically checked ast with annotated types. Here is where all the linearity checking happens. Multiplicities are stripped out and the resulting semantically checked ast has no knowledge of multiplicites whatsoever as they are not needed for the rest of the pipeline. If there is a type or multiplicity mismatch of any sort, or any other type related error, the typechecker will throw an error. This was mostly authored by Ben, with some help from Jay.

3. Monomorphization: Monomorphization is the process of turning polymorphic/ rank-n code and turning it into rank-1 code. Our strategy is to replace

every polymorphic variable with a special type called `BoxT`, which is a single type which represents all polymorphic variables. We convert to and from `BoxT` through the use of a new expressions Box and `Unbox`. Box turns any value into a `BoxT`, and `Unbox` turns any `Box` value back into a given type. Thus, application to a lambda of type `BoxT -> BoxT` amounts to boxing the argument and unboxing the result. Later down the pipeline during code generation, this will result in a simple cast in LLVM. This was mostly authored by Ben with some help from Jay.

4. Closure Conversion: During closure conversion, every lambda abstraction is named uniquely. The free variables of every lambda is collected, and then they are appended to the body of the lambda to produce a closure. These free variables are explicitly passed between lambdas in an environment. Once this process happens, we are free to "lift" each lambda into a top level function safely, as all the free variables will have been provided in the closure's environment. Adding closures for external declarations, and data type construction are also done during closure conversion. Let-in expressions require additional processing as their 0th argument is always a Debruijn index which refers to itself for recursion and the ith argument refers to the $(i-1)$th argument outside the let expression. This system is useful for typechecking but here we must decrement every variable inside the left hand side of the let-in expression and replace the 0th argument with a unique name and lift, as before. The work to implement closure conversion was shared roughly equally between Ben and Jay

5. Code generation: During code generation, all abstract data types become tagged unions in LLVM, i.e. `{ i64, i8* }`. The first parameter, the tag, tells us what constructor is pointed to as the second parameter. Algebraic data types are allocated on the heap. Every closed top-level function becomes a function which takes in `{ i8*, i8** }`. The first parameter contains the formal argument, which gets cast to a certain data type in body of the function itself. The second is a reference to an environment, which is a `malloc`'d array of pointers referring to free variables in the expression. Last, all external `c` calls are processed and declared in LLVM. The work in code generation was shared roughly equally between Ben, Jay and Sophia.

6 Test Plan

6.1 Test Suites

In order to full test lingo, we wrote regression tests in an attempt to cover all of the different features of our language while also allowing us to test our pipeline. We wrote multiple different types of tests looking for both program failures and sucessfull programs. We wrote a couple of regular use case tests such as the ones seen below for Fibonacci Sequence, Function Composition and Safe Linear File IO. On top of this we also wrote a multitude of test suites to sets aspects of our compiler such as: Arithmetic, functions, let statements, malloc and free, abstract data types, correct and incorrect syntax, operartions, control flow, assignment and typechecking.

6.2 Automation

As we were running our compiler a lot and testing it often, we wanted to have two different things. An easy method of running all test and individual tests inside of the docker container as well as testing integration with github. For the first aspect, we built a custom docker container which mounts compiler repository and runs tests. By running the `test.sh` program in the root of the directory all the tests will be run inside docker. By specifiying the `-s` flag, you can run just a single file. These tests automatically build the llvm, assembly, executable, and diff every time you run a test. The test directory can also be cleaned by using the `-c` or `--clean` flags with `test.sh`. Finally, we also included custom githooks which forced each member of the team to pass all tests before commiting to the main github branch.

6.3 Test Listing

See appendix.

6.4 Sample Test Programs and LLVM

6.4.1 Test One: Fibonacci

```
print_int : Int -> Int;

f : Int -> Int = \x. if x == 0 then 1 else x * f (x - 1);
```

```
main : Int = print_int (f 5);
```

Output LLVM:

```
; ModuleID = 'lingo'
source_filename = "lingo"

%clos = type { i8*, i8** }

declare void @__die__()

define i1* @__prim__not(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i1
  %arg_ptr = bitcast i8* %0 to i1*
  %arg = load i1, i1* %arg_ptr
  store i1 %arg, i1* %call_arg_ptr
  %call_arg = load i1, i1* %call_arg_ptr
  %call_ret = call i1 @__prim__unop__not(i1 %call_arg)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define i64* @__prim__neg(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %call_arg = load i64, i64* %call_arg_ptr
  %call_ret = call i64 @__prim__unop__neg(i64 %call_arg)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}
```

```
define %clos* @__prim__or(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__or1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1*
  %closarg_ptr = bitcast i8* %malloccall2 to i1*
  %arg_ptr = bitcast i8* %0 to i1*
  %arg = load i1, i1* %arg_ptr
  store i1 %arg, i1* %closarg_ptr
  %raw_arg_ptr = bitcast i1* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__or1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i1
  %call_arg_ptr1 = alloca i1
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i1*
  %arg = load i1, i1* %arg_ptr
  store i1 %arg, i1* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i1*
  %arg3 = load i1, i1* %arg_ptr2
  store i1 %arg3, i1* %call_arg_ptr1
```

```
  %call_arg = load i1, i1* %call_arg_ptr
  %call_arg4 = load i1, i1* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__or(i1 %call_arg, i1 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__and(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__and1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1*
  %closarg_ptr = bitcast i8* %malloccall2 to i1*
  %arg_ptr = bitcast i8* %0 to i1*
  %arg = load i1, i1* %arg_ptr
  store i1 %arg, i1* %closarg_ptr
  %raw_arg_ptr = bitcast i1* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__and1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i1
  %call_arg_ptr1 = alloca i1
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
```

```
  %arg_ptr = bitcast i8* %raw_arg_ptr to i1*
  %arg = load i1, i1* %arg_ptr
  store i1 %arg, i1* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i1*
  %arg3 = load i1, i1* %arg_ptr2
  store i1 %arg3, i1* %call_arg_ptr1
  %call_arg = load i1, i1* %call_arg_ptr
  %call_arg4 = load i1, i1* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__and(i1 %call_arg, i1 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__neq(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__neq1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__neq1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
```

```
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__neq(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__lt(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__lt1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
```

```
  ret %clos* %rval_ptr
}

define i1* @__prim__lt1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__lt(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__leq(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__leq1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
```

```
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__leq1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__leq(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__eq(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__eq1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
```

```
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__eq1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__eq(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__gt(i8* %0, i8** %1) {
```

```
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__gt1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__gt1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
```

```
  %call_ret = call i1 @__prim__binop__gt(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__geq(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__geq1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__geq1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
```

```
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__geq(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__divide(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i64* (i8*, i8**)* @__prim__divide1 to i8*), i8** %clos_fn_p
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i64* @__prim__divide1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
```

```
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i64 @__prim__binop__divide(i64 %call_arg, i64 %call_arg4)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}

define %clos* @__prim__times(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i64* (i8*, i8**)* @__prim__times1 to i8*), i8** %clos_fn_pt
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}
```

```
define i64* @__prim__times1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i64 @__prim__binop__times(i64 %call_arg, i64 %call_arg4)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}

define %clos* @__prim__minus(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i64* (i8*, i8**)* @__prim__minus1 to i8*), i8** %clos_fn_pt
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
```

```
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i64* @__prim__minus1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i64 @__prim__binop__minus(i64 %call_arg, i64 %call_arg4)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}

define %clos* @__prim__plus(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i64* (i8*, i8**)* @__prim__plus1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
```

```
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i64* @__prim__plus1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i64 @__prim__binop__plus(i64 %call_arg, i64 %call_arg4)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}

define i64* @__print_int__(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
```

```
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %call_arg = load i64, i64* %call_arg_ptr
  %call_ret = call i64 @print_int(i64 %call_arg)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}

define %clos* @f(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i64* (i8*, i8**)* @fn_f to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall1 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i64* @fn_f(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %ifcond = alloca i1
  %app_lhs64 = alloca %clos
  %app_lhs65 = alloca %clos
  %clos_fn_ptr66 = getelementptr inbounds %clos, %clos* %app_lhs65, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @__prim__eq to i8*), i8** %clos_fn_ptr6
  %malloccall67 = tail call i8* @malloc(i32 0)
  %env_aloc68 = bitcast i8* %malloccall67 to [0 x i8*]*
  %env69 = getelementptr inbounds %clos, %clos* %app_lhs65, i32 0, i32 1
  %env_ptr_raw70 = bitcast [0 x i8*]* %env_aloc68 to i8**
```

```
  store i8** %env_ptr_raw70, i8*** %env69
  %app_rhs71 = alloca i64
  %arg_ptr72 = bitcast i8* %0 to i64*
  %arg73 = load i64, i64* %arg_ptr72
  store i64 %arg73, i64* %app_rhs71
  %raw_app_rhs74 = bitcast i64* %app_rhs71 to i8*
  %raw_fn_ptr_ptr75 = getelementptr inbounds %clos, %clos* %app_lhs65, i32 0, i3
  %raw_fn_ptr76 = load i8*, i8** %raw_fn_ptr_ptr75
  %fn_ptr77 = bitcast i8* %raw_fn_ptr76 to %clos* (i8*, i8**)*
  %args_ptr78 = getelementptr inbounds %clos, %clos* %app_lhs65, i32 0, i32 1
  %args79 = load i8**, i8*** %args_ptr78
  %app_res_ptr80 = call %clos* %fn_ptr77(i8* %raw_app_rhs74, i8** %args79)
  %app_res81 = load %clos, %clos* %app_res_ptr80
  store %clos %app_res81, %clos* %app_lhs64
  %app_rhs82 = alloca i64
  store i64 0, i64* %app_rhs82
  %raw_app_rhs83 = bitcast i64* %app_rhs82 to i8*
  %raw_fn_ptr_ptr84 = getelementptr inbounds %clos, %clos* %app_lhs64, i32 0, i3
  %raw_fn_ptr85 = load i8*, i8** %raw_fn_ptr_ptr84
  %fn_ptr86 = bitcast i8* %raw_fn_ptr85 to i1* (i8*, i8**)*
  %args_ptr87 = getelementptr inbounds %clos, %clos* %app_lhs64, i32 0, i32 1
  %args88 = load i8**, i8*** %args_ptr87
  %app_res_ptr89 = call i1* %fn_ptr86(i8* %raw_app_rhs83, i8** %args88)
  %app_res90 = load i1, i1* %app_res_ptr89
  store i1 %app_res90, i1* %ifcond
  %cond_val = load i1, i1* %ifcond
  br i1 %cond_val, label %brtrue, label %brfalse

brtrue:                                           ; preds = %entry
  store i64 1, i64* %rval_ptr
  br label %end

brfalse:                                          ; preds = %entry
  %app_lhs = alloca %clos
  %app_lhs1 = alloca %clos
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @__prim__times to i8*), i8** %clos_fn_p
  %malloccall2 = tail call i8* @malloc(i32 0)
```

```
%env_aloc = bitcast i8* %malloccall2 to [0 x i8*]*
%env = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 1
%env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
store i8** %env_ptr_raw, i8*** %env
%app_rhs = alloca i64
%arg_ptr = bitcast i8* %0 to i64*
%arg = load i64, i64* %arg_ptr
store i64 %arg, i64* %app_rhs
%raw_app_rhs = bitcast i64* %app_rhs to i8*
%raw_fn_ptr_ptr = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 0
%raw_fn_ptr = load i8*, i8** %raw_fn_ptr_ptr
%fn_ptr = bitcast i8* %raw_fn_ptr to %clos* (i8*, i8**)*
%args_ptr = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 1
%args = load i8**, i8*** %args_ptr
%app_res_ptr = call %clos* %fn_ptr(i8* %raw_app_rhs, i8** %args)
%app_res = load %clos, %clos* %app_res_ptr
store %clos %app_res, %clos* %app_lhs
%app_rhs3 = alloca i64
%app_lhs4 = alloca %clos
%app_lhs5 = alloca %clos
%clos_fn_ptr6 = getelementptr inbounds %clos, %clos* %app_lhs5, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @f to i8*), i8** %clos_fn_ptr6
%malloccall7 = tail call i8* @malloc(i32 0)
%env_aloc8 = bitcast i8* %malloccall7 to [0 x i8*]*
%env9 = getelementptr inbounds %clos, %clos* %app_lhs5, i32 0, i32 1
%env_ptr_raw10 = bitcast [0 x i8*]* %env_aloc8 to i8**
store i8** %env_ptr_raw10, i8*** %env9
%app_rhs11 = alloca i64
store i64 0, i64* %app_rhs11
%raw_app_rhs12 = bitcast i64* %app_rhs11 to i8*
%raw_fn_ptr_ptr13 = getelementptr inbounds %clos, %clos* %app_lhs5, i32 0, i32
%raw_fn_ptr14 = load i8*, i8** %raw_fn_ptr_ptr13
%fn_ptr15 = bitcast i8* %raw_fn_ptr14 to %clos* (i8*, i8**)*
%args_ptr16 = getelementptr inbounds %clos, %clos* %app_lhs5, i32 0, i32 1
%args17 = load i8**, i8*** %args_ptr16
%app_res_ptr18 = call %clos* %fn_ptr15(i8* %raw_app_rhs12, i8** %args17)
%app_res19 = load %clos, %clos* %app_res_ptr18
store %clos %app_res19, %clos* %app_lhs4
```

```
%app_rhs20 = alloca i64
%app_lhs21 = alloca %clos
%app_lhs22 = alloca %clos
%clos_fn_ptr23 = getelementptr inbounds %clos, %clos* %app_lhs22, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @__prim__minus to i8*), i8** %clos_fn_p
%malloccall24 = tail call i8* @malloc(i32 0)
%env_aloc25 = bitcast i8* %malloccall24 to [0 x i8*]*
%env26 = getelementptr inbounds %clos, %clos* %app_lhs22, i32 0, i32 1
%env_ptr_raw27 = bitcast [0 x i8*]* %env_aloc25 to i8**
store i8** %env_ptr_raw27, i8*** %env26
%app_rhs28 = alloca i64
%arg_ptr29 = bitcast i8* %0 to i64*
%arg30 = load i64, i64* %arg_ptr29
store i64 %arg30, i64* %app_rhs28
%raw_app_rhs31 = bitcast i64* %app_rhs28 to i8*
%raw_fn_ptr_ptr32 = getelementptr inbounds %clos, %clos* %app_lhs22, i32 0, i3
%raw_fn_ptr33 = load i8*, i8** %raw_fn_ptr_ptr32
%fn_ptr34 = bitcast i8* %raw_fn_ptr33 to %clos* (i8*, i8**)*
%args_ptr35 = getelementptr inbounds %clos, %clos* %app_lhs22, i32 0, i32 1
%args36 = load i8**, i8*** %args_ptr35
%app_res_ptr37 = call %clos* %fn_ptr34(i8* %raw_app_rhs31, i8** %args36)
%app_res38 = load %clos, %clos* %app_res_ptr37
store %clos %app_res38, %clos* %app_lhs21
%app_rhs39 = alloca i64
store i64 1, i64* %app_rhs39
%raw_app_rhs40 = bitcast i64* %app_rhs39 to i8*
%raw_fn_ptr_ptr41 = getelementptr inbounds %clos, %clos* %app_lhs21, i32 0, i3
%raw_fn_ptr42 = load i8*, i8** %raw_fn_ptr_ptr41
%fn_ptr43 = bitcast i8* %raw_fn_ptr42 to i64* (i8*, i8**)*
%args_ptr44 = getelementptr inbounds %clos, %clos* %app_lhs21, i32 0, i32 1
%args45 = load i8**, i8*** %args_ptr44
%app_res_ptr46 = call i64* %fn_ptr43(i8* %raw_app_rhs40, i8** %args45)
%app_res47 = load i64, i64* %app_res_ptr46
store i64 %app_res47, i64* %app_rhs20
%raw_app_rhs48 = bitcast i64* %app_rhs20 to i8*
%raw_fn_ptr_ptr49 = getelementptr inbounds %clos, %clos* %app_lhs4, i32 0, i32
%raw_fn_ptr50 = load i8*, i8** %raw_fn_ptr_ptr49
%fn_ptr51 = bitcast i8* %raw_fn_ptr50 to i64* (i8*, i8**)*
```

```
  %args_ptr52 = getelementptr inbounds %clos, %clos* %app_lhs4, i32 0, i32 1
  %args53 = load i8**, i8*** %args_ptr52
  %app_res_ptr54 = call i64* %fn_ptr51(i8* %raw_app_rhs48, i8** %args53)
  %app_res55 = load i64, i64* %app_res_ptr54
  store i64 %app_res55, i64* %app_rhs3
  %raw_app_rhs56 = bitcast i64* %app_rhs3 to i8*
  %raw_fn_ptr_ptr57 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32
  %raw_fn_ptr58 = load i8*, i8** %raw_fn_ptr_ptr57
  %fn_ptr59 = bitcast i8* %raw_fn_ptr58 to i64* (i8*, i8**)*
  %args_ptr60 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
  %args61 = load i8**, i8*** %args_ptr60
  %app_res_ptr62 = call i64* %fn_ptr59(i8* %raw_app_rhs56, i8** %args61)
  %app_res63 = load i64, i64* %app_res_ptr62
  store i64 %app_res63, i64* %rval_ptr
  br label %end

end:                                              ; preds = %brfalse, %brtrue
  ret i64* %rval_ptr
}

declare i1 @__prim__unop__not(i1)

declare i64 @__prim__unop__neg(i64)

declare i1 @__prim__binop__or(i1, i1)

declare i1 @__prim__binop__and(i1, i1)

declare i1 @__prim__binop__neq(i64, i64)

declare i1 @__prim__binop__lt(i64, i64)

declare i1 @__prim__binop__leq(i64, i64)

declare i1 @__prim__binop__eq(i64, i64)

declare i1 @__prim__binop__gt(i64, i64)
```

```
declare i1 @__prim__binop__geq(i64, i64)

declare i64 @__prim__binop__divide(i64, i64)

declare i64 @__prim__binop__times(i64, i64)

declare i64 @__prim__binop__minus(i64, i64)

declare i64 @__prim__binop__plus(i64, i64)

declare i64 @print_int(i64)

declare noalias i8* @malloc(i32)

define i64 @main() {
entry:
  %ret = alloca i64
  %app_lhs = alloca %clos
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 0
  store i8* bitcast (i64* (i8*, i8**)* @__print_int__ to i8*), i8** %clos_fn_ptr
  %malloccall = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  %app_rhs = alloca i64
  %app_lhs1 = alloca %clos
  %app_lhs2 = alloca %clos
  %clos_fn_ptr3 = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @f to i8*), i8** %clos_fn_ptr3
  %malloccall4 = tail call i8* @malloc(i32 0)
  %env_aloc5 = bitcast i8* %malloccall4 to [0 x i8*]*
  %env6 = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 1
  %env_ptr_raw7 = bitcast [0 x i8*]* %env_aloc5 to i8**
  store i8** %env_ptr_raw7, i8*** %env6
  %app_rhs8 = alloca i64
  store i64 0, i64* %app_rhs8
  %raw_app_rhs = bitcast i64* %app_rhs8 to i8*
```

```
  %raw_fn_ptr_ptr = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 0
  %raw_fn_ptr = load i8*, i8** %raw_fn_ptr_ptr
  %fn_ptr = bitcast i8* %raw_fn_ptr to %clos* (i8*, i8**)*
  %args_ptr = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 1
  %args = load i8**, i8*** %args_ptr
  %app_res_ptr = call %clos* %fn_ptr(i8* %raw_app_rhs, i8** %args)
  %app_res = load %clos, %clos* %app_res_ptr
  store %clos %app_res, %clos* %app_lhs1
  %app_rhs9 = alloca i64
  store i64 5, i64* %app_rhs9
  %raw_app_rhs10 = bitcast i64* %app_rhs9 to i8*
  %raw_fn_ptr_ptr11 = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32
  %raw_fn_ptr12 = load i8*, i8** %raw_fn_ptr_ptr11
  %fn_ptr13 = bitcast i8* %raw_fn_ptr12 to i64* (i8*, i8**)*
  %args_ptr14 = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 1
  %args15 = load i8**, i8*** %args_ptr14
  %app_res_ptr16 = call i64* %fn_ptr13(i8* %raw_app_rhs10, i8** %args15)
  %app_res17 = load i64, i64* %app_res_ptr16
  store i64 %app_res17, i64* %app_rhs
  %raw_app_rhs18 = bitcast i64* %app_rhs to i8*
  %raw_fn_ptr_ptr19 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32
  %raw_fn_ptr20 = load i8*, i8** %raw_fn_ptr_ptr19
  %fn_ptr21 = bitcast i8* %raw_fn_ptr20 to i64* (i8*, i8**)*
  %args_ptr22 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
  %args23 = load i8**, i8*** %args_ptr22
  %app_res_ptr24 = call i64* %fn_ptr21(i8* %raw_app_rhs18, i8** %args23)
  %app_res25 = load i64, i64* %app_res_ptr24
  store i64 %app_res25, i64* %ret
  %retval = load i64, i64* %ret
  ret i64 %retval
}
```

6.4.2 Test Two: Compose

```
print_int : Int -> Int;
succ x : Int -> Int = x + 1;

compose p q a b c f g x :
```

```
    #p #q @a @b @c (b -p> c) -> (a -q> b) -> a -p*q> c = f (g x);
main : Int = (compose #Unr #Unr @Int @Int @Int print_int succ) 100;
```

Output LLVM:

```
; ModuleID = 'lingo'
source_filename = "lingo"

%clos = type { i8*, i8** }
%boxt = type { i8*, i8* }

declare void @__die__()

define i1* @__prim__not(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i1
  %arg_ptr = bitcast i8* %0 to i1*
  %arg = load i1, i1* %arg_ptr
  store i1 %arg, i1* %call_arg_ptr
  %call_arg = load i1, i1* %call_arg_ptr
  %call_ret = call i1 @__prim__unop__not(i1 %call_arg)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define i64* @__prim__neg(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %call_arg = load i64, i64* %call_arg_ptr
  %call_ret = call i64 @__prim__unop__neg(i64 %call_arg)
  store i64 %call_ret, i64* %rval_ptr
```

```
  ret i64* %rval_ptr
}

define %clos* @__prim__or(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__or1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1*
  %closarg_ptr = bitcast i8* %malloccall2 to i1*
  %arg_ptr = bitcast i8* %0 to i1*
  %arg = load i1, i1* %arg_ptr
  store i1 %arg, i1* %closarg_ptr
  %raw_arg_ptr = bitcast i1* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__or1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i1
  %call_arg_ptr1 = alloca i1
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i1*
  %arg = load i1, i1* %arg_ptr
  store i1 %arg, i1* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i1*
```

```
  %arg3 = load i1, i1* %arg_ptr2
  store i1 %arg3, i1* %call_arg_ptr1
  %call_arg = load i1, i1* %call_arg_ptr
  %call_arg4 = load i1, i1* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__or(i1 %call_arg, i1 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__and(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__and1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1*
  %closarg_ptr = bitcast i8* %malloccall2 to i1*
  %arg_ptr = bitcast i8* %0 to i1*
  %arg = load i1, i1* %arg_ptr
  store i1 %arg, i1* %closarg_ptr
  %raw_arg_ptr = bitcast i1* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__and1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i1
  %call_arg_ptr1 = alloca i1
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
```

```
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i1*
  %arg = load i1, i1* %arg_ptr
  store i1 %arg, i1* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i1*
  %arg3 = load i1, i1* %arg_ptr2
  store i1 %arg3, i1* %call_arg_ptr1
  %call_arg = load i1, i1* %call_arg_ptr
  %call_arg4 = load i1, i1* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__and(i1 %call_arg, i1 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__neq(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__neq1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__neq1(i8* %0, i8** %1) {
```

```
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__neq(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__lt(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__lt1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
```

```
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__lt1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__lt(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__leq(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__leq1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
```

```
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__leq1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__leq(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__eq(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
```

```
  store i8* bitcast (i1* (i8*, i8**)* @__prim__eq1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__eq1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__eq(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}
```

```
define %clos* @__prim__gt(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__gt1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__gt1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
```

```
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__gt(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__geq(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__geq1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__geq1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
```

```
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__geq(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__divide(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i64* (i8*, i8**)* @__prim__divide1 to i8*), i8** %clos_fn_p
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i64* @__prim__divide1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
```

```
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i64 @__prim__binop__divide(i64 %call_arg, i64 %call_arg4)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}

define %clos* @__prim__times(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i64* (i8*, i8**)* @__prim__times1 to i8*), i8** %clos_fn_pt
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
```

```
  ret %clos* %rval_ptr
}

define i64* @__prim__times1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i64 @__prim__binop__times(i64 %call_arg, i64 %call_arg4)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}

define %clos* @__prim__minus(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i64* (i8*, i8**)* @__prim__minus1 to i8*), i8** %clos_fn_pt
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
```

```
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i64* @__prim__minus1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i64 @__prim__binop__minus(i64 %call_arg, i64 %call_arg4)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}

define %clos* @__prim__plus(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i64* (i8*, i8**)* @__prim__plus1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
```

```
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i64* @__prim__plus1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i64 @__prim__binop__plus(i64 %call_arg, i64 %call_arg4)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}

define i64* @__print_int__(i8* %0, i8** %1) {
```

```
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %call_arg = load i64, i64* %call_arg_ptr
  %call_ret = call i64 @print_int(i64 %call_arg)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}

define %clos* @succ(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i64* (i8*, i8**)* @fn_succ to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall1 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %clos* @compose(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @fn_compose to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall1 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
```

```
  ret %clos* %rval_ptr
}

define i64* @fn_succ(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %app_lhs = alloca %clos
  %app_lhs1 = alloca %clos
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @__prim__plus to i8*), i8** %clos_fn_pt
  %malloccall2 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall2 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  %app_rhs = alloca i64
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %app_rhs
  %raw_app_rhs = bitcast i64* %app_rhs to i8*
  %raw_fn_ptr_ptr = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 0
  %raw_fn_ptr = load i8*, i8** %raw_fn_ptr_ptr
  %fn_ptr = bitcast i8* %raw_fn_ptr to %clos* (i8*, i8**)*
  %args_ptr = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 1
  %args = load i8**, i8*** %args_ptr
  %app_res_ptr = call %clos* %fn_ptr(i8* %raw_app_rhs, i8** %args)
  %app_res = load %clos, %clos* %app_res_ptr
  store %clos %app_res, %clos* %app_lhs
  %app_rhs3 = alloca i64
  store i64 1, i64* %app_rhs3
  %raw_app_rhs4 = bitcast i64* %app_rhs3 to i8*
  %raw_fn_ptr_ptr5 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 0
  %raw_fn_ptr6 = load i8*, i8** %raw_fn_ptr_ptr5
  %fn_ptr7 = bitcast i8* %raw_fn_ptr6 to i64* (i8*, i8**)*
  %args_ptr8 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
  %args9 = load i8**, i8*** %args_ptr8
  %app_res_ptr10 = call i64* %fn_ptr7(i8* %raw_app_rhs4, i8** %args9)
```

```
  %app_res11 = load i64, i64* %app_res_ptr10
  store i64 %app_res11, i64* %rval_ptr
  ret i64* %rval_ptr
}

define %clos* @fn_compose(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @fn_compose79851791761346646400 to i8*),
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1*
  %closarg_ptr = bitcast i8* %malloccall2 to %clos*
  %arg_ptr = bitcast i8* %0 to %clos*
  %arg = load %clos, %clos* %arg_ptr
  store %clos %arg, %clos* %closarg_ptr
  %raw_arg_ptr = bitcast %clos* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %clos* @fn_compose79851791761346646400(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%boxt* (i8*, i8**)* @fn_compose4968362049263200281 to i8*),
  %malloccall1 = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1*
  %env_aloc = bitcast i8* %malloccall1 to [2 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [2 x i8*], [2 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1*
  %closarg_ptr = bitcast i8* %malloccall2 to %clos*
```

```
  %arg_ptr = bitcast i8* %0 to %clos*
  %arg = load %clos, %clos* %arg_ptr
  store %clos %arg, %clos* %closarg_ptr
  %raw_arg_ptr = bitcast %clos* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %raw_closarg_ptr_ptr3 = getelementptr inbounds [2 x i8*], [2 x i8*]* %env_aloc
  %malloccall4 = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1*
  %closarg_ptr5 = bitcast i8* %malloccall4 to %clos*
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr6 = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr7 = bitcast i8* %raw_arg_ptr6 to %clos*
  %arg8 = load %clos, %clos* %arg_ptr7
  store %clos %arg8, %clos* %closarg_ptr5
  %raw_arg_ptr9 = bitcast %clos* %closarg_ptr5 to i8*
  store i8* %raw_arg_ptr9, i8** %raw_closarg_ptr_ptr3
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [2 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %boxt* @fn_compose49683620492632002 81(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %boxt*
  %app_lhs = alloca %clos
  %arg_gep_ptr = bitcast i8** %1 to [2 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [2 x i8*], [2 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to %clos*
  %arg = load %clos, %clos* %arg_ptr
  store %clos %arg, %clos* %app_lhs
  %app_rhs = alloca %boxt
  %app_lhs1 = alloca %clos
  %arg_gep_ptr2 = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr3 = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr2, i32 0,
  %raw_arg_ptr4 = load i8*, i8** %raw_arg_ptr_ptr3
```

```
  %arg_ptr5 = bitcast i8* %raw_arg_ptr4 to %clos*
  %arg6 = load %clos, %clos* %arg_ptr5
  store %clos %arg6, %clos* %app_lhs1
  %app_rhs7 = alloca %boxt
  %arg_ptr8 = bitcast i8* %0 to %boxt*
  %arg9 = load %boxt, %boxt* %arg_ptr8
  store %boxt %arg9, %boxt* %app_rhs7
  %raw_app_rhs = bitcast %boxt* %app_rhs7 to i8*
  %raw_fn_ptr_ptr = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 0
  %raw_fn_ptr = load i8*, i8** %raw_fn_ptr_ptr
  %fn_ptr = bitcast i8* %raw_fn_ptr to %boxt* (i8*, i8**)*
  %args_ptr = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 1
  %args = load i8**, i8*** %args_ptr
  %app_res_ptr = call %boxt* %fn_ptr(i8* %raw_app_rhs, i8** %args)
  %app_res = load %boxt, %boxt* %app_res_ptr
  store %boxt %app_res, %boxt* %app_rhs
  %raw_app_rhs10 = bitcast %boxt* %app_rhs to i8*
  %raw_fn_ptr_ptr11 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32
  %raw_fn_ptr12 = load i8*, i8** %raw_fn_ptr_ptr11
  %fn_ptr13 = bitcast i8* %raw_fn_ptr12 to %boxt* (i8*, i8**)*
  %args_ptr14 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
  %args15 = load i8**, i8*** %args_ptr14
  %app_res_ptr16 = call %boxt* %fn_ptr13(i8* %raw_app_rhs10, i8** %args15)
  %app_res17 = load %boxt, %boxt* %app_res_ptr16
  store %boxt %app_res17, %boxt* %rval_ptr
  ret %boxt* %rval_ptr
}

declare i1 @__prim__unop__not(i1)

declare i64 @__prim__unop__neg(i64)

declare i1 @__prim__binop__or(i1, i1)

declare i1 @__prim__binop__and(i1, i1)

declare i1 @__prim__binop__neq(i64, i64)
```

```
declare i1 @__prim__binop__lt(i64, i64)

declare i1 @__prim__binop__leq(i64, i64)

declare i1 @__prim__binop__eq(i64, i64)

declare i1 @__prim__binop__gt(i64, i64)

declare i1 @__prim__binop__geq(i64, i64)

declare i64 @__prim__binop__divide(i64, i64)

declare i64 @__prim__binop__times(i64, i64)

declare i64 @__prim__binop__minus(i64, i64)

declare i64 @__prim__binop__plus(i64, i64)

declare i64 @print_int(i64)

declare noalias i8* @malloc(i32)

define i64 @main() {
entry:
  %ret = alloca i64
  %boxed = alloca %boxt
  %app_lhs = alloca %clos
  %app_lhs1 = alloca %clos
  %app_lhs2 = alloca %clos
  %app_lhs3 = alloca %clos
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %app_lhs3, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @compose to i8*), i8** %clos_fn_ptr
  %malloccall = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %app_lhs3, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  %app_rhs = alloca i64
```

```
store i64 0, i64* %app_rhs
%raw_app_rhs = bitcast i64* %app_rhs to i8*
%raw_fn_ptr_ptr = getelementptr inbounds %clos, %clos* %app_lhs3, i32 0, i32 0
%raw_fn_ptr = load i8*, i8** %raw_fn_ptr_ptr
%fn_ptr = bitcast i8* %raw_fn_ptr to %clos* (i8*, i8**)*
%args_ptr = getelementptr inbounds %clos, %clos* %app_lhs3, i32 0, i32 1
%args = load i8**, i8*** %args_ptr
%app_res_ptr = call %clos* %fn_ptr(i8* %raw_app_rhs, i8** %args)
%app_res = load %clos, %clos* %app_res_ptr
store %clos %app_res, %clos* %app_lhs2
%app_rhs4 = alloca %clos
%clos_fn_ptr5 = getelementptr inbounds %clos, %clos* %app_rhs4, i32 0, i32 0
store i8* bitcast (i64* (i8*, i8**)* @__print_int__ to i8*), i8** %clos_fn_ptr
%malloccall6 = tail call i8* @malloc(i32 0)
%env_aloc7 = bitcast i8* %malloccall6 to [0 x i8*]*
%env8 = getelementptr inbounds %clos, %clos* %app_rhs4, i32 0, i32 1
%env_ptr_raw9 = bitcast [0 x i8*]* %env_aloc7 to i8**
store i8** %env_ptr_raw9, i8*** %env8
%raw_app_rhs10 = bitcast %clos* %app_rhs4 to i8*
%raw_fn_ptr_ptr11 = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32
%raw_fn_ptr12 = load i8*, i8** %raw_fn_ptr_ptr11
%fn_ptr13 = bitcast i8* %raw_fn_ptr12 to %clos* (i8*, i8**)*
%args_ptr14 = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 1
%args15 = load i8**, i8*** %args_ptr14
%app_res_ptr16 = call %clos* %fn_ptr13(i8* %raw_app_rhs10, i8** %args15)
%app_res17 = load %clos, %clos* %app_res_ptr16
store %clos %app_res17, %clos* %app_lhs1
%app_rhs18 = alloca %clos
%app_lhs19 = alloca %clos
%clos_fn_ptr20 = getelementptr inbounds %clos, %clos* %app_lhs19, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @succ to i8*), i8** %clos_fn_ptr20
%malloccall21 = tail call i8* @malloc(i32 0)
%env_aloc22 = bitcast i8* %malloccall21 to [0 x i8*]*
%env23 = getelementptr inbounds %clos, %clos* %app_lhs19, i32 0, i32 1
%env_ptr_raw24 = bitcast [0 x i8*]* %env_aloc22 to i8**
store i8** %env_ptr_raw24, i8*** %env23
%app_rhs25 = alloca i64
store i64 0, i64* %app_rhs25
```

```
%raw_app_rhs26 = bitcast i64* %app_rhs25 to i8*
%raw_fn_ptr_ptr27 = getelementptr inbounds %clos, %clos* %app_lhs19, i32 0, i3
%raw_fn_ptr28 = load i8*, i8** %raw_fn_ptr_ptr27
%fn_ptr29 = bitcast i8* %raw_fn_ptr28 to %clos* (i8*, i8**)*
%args_ptr30 = getelementptr inbounds %clos, %clos* %app_lhs19, i32 0, i32 1
%args31 = load i8**, i8*** %args_ptr30
%app_res_ptr32 = call %clos* %fn_ptr29(i8* %raw_app_rhs26, i8** %args31)
%app_res33 = load %clos, %clos* %app_res_ptr32
store %clos %app_res33, %clos* %app_rhs18
%raw_app_rhs34 = bitcast %clos* %app_rhs18 to i8*
%raw_fn_ptr_ptr35 = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32
%raw_fn_ptr36 = load i8*, i8** %raw_fn_ptr_ptr35
%fn_ptr37 = bitcast i8* %raw_fn_ptr36 to %clos* (i8*, i8**)*
%args_ptr38 = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 1
%args39 = load i8**, i8*** %args_ptr38
%app_res_ptr40 = call %clos* %fn_ptr37(i8* %raw_app_rhs34, i8** %args39)
%app_res41 = load %clos, %clos* %app_res_ptr40
store %clos %app_res41, %clos* %app_lhs
%app_rhs42 = alloca %boxt
%unbox = alloca i64
store i64 100, i64* %unbox
%box_ptr = bitcast i64* %unbox to %boxt*
%box_val = load %boxt, %boxt* %box_ptr
store %boxt %box_val, %boxt* %app_rhs42
%raw_app_rhs43 = bitcast %boxt* %app_rhs42 to i8*
%raw_fn_ptr_ptr44 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32
%raw_fn_ptr45 = load i8*, i8** %raw_fn_ptr_ptr44
%fn_ptr46 = bitcast i8* %raw_fn_ptr45 to %boxt* (i8*, i8**)*
%args_ptr47 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
%args48 = load i8**, i8*** %args_ptr47
%app_res_ptr49 = call %boxt* %fn_ptr46(i8* %raw_app_rhs43, i8** %args48)
%app_res50 = load %boxt, %boxt* %app_res_ptr49
store %boxt %app_res50, %boxt* %boxed
%unbox_ptr = bitcast %boxt* %boxed to i64*
%unbox_val = load i64, i64* %unbox_ptr
store i64 %unbox_val, i64* %ret
%retval = load i64, i64* %ret
ret i64 %retval
```

```
}
```

6.4.3 Test Three : Safe File IO

```
data File where
data Mem where

data Tuple a b #p #q where
    Tuple : a -p> b -q> Tuple a b;

print_int : Int -* Int;
print_string : Mem -* Mem;
prim_alloc : Int -* Mem;
prim_drop : Mem -* ();
set_bit : Mem -* Int -> Char -> Mem;
open_file : Mem -> Mem -> File;
read_file : File -* Tuple Mem File #Unr #One;
close_file : File -* ();

data String where
    E : String;
    C : Char -> String -> String;

len : String -> Int
    = \s. case s of
        C c s -> 1 + len s;
        _ -> 0;
    ;
printString : Mem -* Mem = print_string;

string_to_mem : String -> Mem
            = \s.
            let m : Mem = prim_alloc (1 + len s) in
            let string_to_mem' : String -> Mem -* Int -> Mem
               = \s. \m. \i. case s of
                     C c s -> string_to_mem' s (set_bit m i c) (i + 1);
                     _ -> set_bit m i '\000';
            in
```

```
            string_to_mem' s m 0;

openFile : String -> String -> (File -* ()) -> ()
         = \filename. \mode. \k. k (open_file (string_to_mem filename)
            (string_to_mem mode));

readFile : File -* Tuple Mem File #Unr #One = read_file;

closeFile : File -* () = close_file;

const : @a @b a -> b -> a
      = \a. \b. \x. \y. x;

printFile : String -> String -> ()
          =
          \filename. \mode.
            let f : File -* () = \file.
              case (readFile file) of
                Tuple str file ->
                  let x : Mem = printString str in
                  closeFile file;
            in
            openFile filename mode f;

main : Int = const @Int @() 0 (printFile "./reg-tests/file.txt" "r");


; ModuleID = 'lingo'
source_filename = "lingo"

%clos = type { i8*, i8** }
%adt = type { i64, i8* }
%C = type { i8, %adt }
%boxt = type { i8*, i8* }
%Tuple = type { %boxt, %boxt }
%E = type {}

declare void @__die__()
```

```
define i1* @__prim__not(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i1
  %arg_ptr = bitcast i8* %0 to i1*
  %arg = load i1, i1* %arg_ptr
  store i1 %arg, i1* %call_arg_ptr
  %call_arg = load i1, i1* %call_arg_ptr
  %call_ret = call i1 @__prim__unop__not(i1 %call_arg)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define i64* @__prim__neg(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %call_arg = load i64, i64* %call_arg_ptr
  %call_ret = call i64 @__prim__unop__neg(i64 %call_arg)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}

define %clos* @__prim__or(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__or1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
```

```
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1*
  %closarg_ptr = bitcast i8* %malloccall2 to i1*
  %arg_ptr = bitcast i8* %0 to i1*
  %arg = load i1, i1* %arg_ptr
  store i1 %arg, i1* %closarg_ptr
  %raw_arg_ptr = bitcast i1* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__or1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i1
  %call_arg_ptr1 = alloca i1
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i1*
  %arg = load i1, i1* %arg_ptr
  store i1 %arg, i1* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i1*
  %arg3 = load i1, i1* %arg_ptr2
  store i1 %arg3, i1* %call_arg_ptr1
  %call_arg = load i1, i1* %call_arg_ptr
  %call_arg4 = load i1, i1* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__or(i1 %call_arg, i1 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__and(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
```

```
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__and1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1*
  %closarg_ptr = bitcast i8* %malloccall2 to i1*
  %arg_ptr = bitcast i8* %0 to i1*
  %arg = load i1, i1* %arg_ptr
  store i1 %arg, i1* %closarg_ptr
  %raw_arg_ptr = bitcast i1* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__and1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i1
  %call_arg_ptr1 = alloca i1
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i1*
  %arg = load i1, i1* %arg_ptr
  store i1 %arg, i1* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i1*
  %arg3 = load i1, i1* %arg_ptr2
  store i1 %arg3, i1* %call_arg_ptr1
  %call_arg = load i1, i1* %call_arg_ptr
  %call_arg4 = load i1, i1* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__and(i1 %call_arg, i1 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
```

```
  ret i1* %rval_ptr
}

define %clos* @__prim__neq(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__neq1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__neq1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
```

```
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__neq(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__lt(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__lt1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__lt1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
```

```
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__lt(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__leq(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__leq1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__leq1(i8* %0, i8** %1) {
```

```
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__leq(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__eq(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__eq1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
```

```
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__eq1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__eq(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__gt(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i1* (i8*, i8**)* @__prim__gt1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
```

```
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__gt1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__gt(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}

define %clos* @__prim__geq(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
```

```
  store i8* bitcast (i1* (i8*, i8**)* @__prim__geq1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i1* @__prim__geq1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* n
  %rval_ptr = bitcast i8* %malloccall to i1*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i1 @__prim__binop__geq(i64 %call_arg, i64 %call_arg4)
  store i1 %call_ret, i1* %rval_ptr
  ret i1* %rval_ptr
}
```

```
define %clos* @__prim__divide(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i64* (i8*, i8**)* @__prim__divide1 to i8*), i8** %clos_fn_p
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i64* @__prim__divide1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
```

```
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i64 @__prim__binop__divide(i64 %call_arg, i64 %call_arg4)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}

define %clos* @__prim__times(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i64* (i8*, i8**)* @__prim__times1 to i8*), i8** %clos_fn_pt
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i64* @__prim__times1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
```

```
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i64 @__prim__binop__times(i64 %call_arg, i64 %call_arg4)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}

define %clos* @__prim__minus(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i64* (i8*, i8**)* @__prim__minus1 to i8*), i8** %clos_fn_pt
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i64* @__prim__minus1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
```

```
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i64 @__prim__binop__minus(i64 %call_arg, i64 %call_arg4)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}

define %clos* @__prim__plus(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i64* (i8*, i8**)* @__prim__plus1 to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
```

```
  ret %clos* %rval_ptr
}

define i64* @__prim__plus1(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %call_arg_ptr1 = alloca i64
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to i64*
  %arg3 = load i64, i64* %arg_ptr2
  store i64 %arg3, i64* %call_arg_ptr1
  %call_arg = load i64, i64* %call_arg_ptr
  %call_arg4 = load i64, i64* %call_arg_ptr1
  %call_ret = call i64 @__prim__binop__plus(i64 %call_arg, i64 %call_arg4)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}

define i64* @__print_int__(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %call_arg_ptr = alloca i64
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %call_arg = load i64, i64* %call_arg_ptr
  %call_ret = call i64 @print_int(i64 %call_arg)
  store i64 %call_ret, i64* %rval_ptr
  ret i64* %rval_ptr
}
```

```
define %adt* @__print_string__(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %call_arg_ptr = alloca %adt
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %call_arg_ptr
  %call_arg = load %adt, %adt* %call_arg_ptr
  %call_ret = call %adt @print_string(%adt %call_arg)
  store %adt %call_ret, %adt* %rval_ptr
  ret %adt* %rval_ptr
}

define %adt* @__prim_alloc__(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %call_arg_ptr = alloca i64
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %call_arg_ptr
  %call_arg = load i64, i64* %call_arg_ptr
  %call_ret = call %adt @prim_alloc(i64 %call_arg)
  store %adt %call_ret, %adt* %rval_ptr
  ret %adt* %rval_ptr
}

define %adt* @__prim_drop__(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %call_arg_ptr = alloca %adt
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %call_arg_ptr
  %call_arg = load %adt, %adt* %call_arg_ptr
```

```
  %call_ret = call %adt @prim_drop(%adt %call_arg)
  store %adt %call_ret, %adt* %rval_ptr
  ret %adt* %rval_ptr
}

define %clos* @__set_bit__(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @__set_bit__2242192272592927321 to i8*)
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt,
  %closarg_ptr = bitcast i8* %malloccall2 to %adt*
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %closarg_ptr
  %raw_arg_ptr = bitcast %adt* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %clos* @__set_bit__2242192272592927321(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%adt* (i8*, i8**)* @__set_bit__171252171853525041 to i8*),
  %malloccall1 = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1*
  %env_aloc = bitcast i8* %malloccall1 to [2 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [2 x i8*], [2 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i6
  %closarg_ptr = bitcast i8* %malloccall2 to i64*
```

```
  %arg_ptr = bitcast i8* %0 to i64*
  %arg = load i64, i64* %arg_ptr
  store i64 %arg, i64* %closarg_ptr
  %raw_arg_ptr = bitcast i64* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %raw_closarg_ptr_ptr3 = getelementptr inbounds [2 x i8*], [2 x i8*]* %env_aloc
  %malloccall4 = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt,
  %closarg_ptr5 = bitcast i8* %malloccall4 to %adt*
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr6 = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr7 = bitcast i8* %raw_arg_ptr6 to %adt*
  %arg8 = load %adt, %adt* %arg_ptr7
  store %adt %arg8, %adt* %closarg_ptr5
  %raw_arg_ptr9 = bitcast %adt* %closarg_ptr5 to i8*
  store i8* %raw_arg_ptr9, i8** %raw_closarg_ptr_ptr3
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [2 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %adt* @__set_bit__17125217185352504l(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %call_arg_ptr = alloca %adt
  %call_arg_ptr1 = alloca i64
  %call_arg_ptr2 = alloca i8
  %arg_gep_ptr = bitcast i8** %1 to [2 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [2 x i8*], [2 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %call_arg_ptr
  %arg_gep_ptr3 = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr4 = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr3, i32 0,
  %raw_arg_ptr5 = load i8*, i8** %raw_arg_ptr_ptr4
```

```
  %arg_ptr6 = bitcast i8* %raw_arg_ptr5 to i64*
  %arg7 = load i64, i64* %arg_ptr6
  store i64 %arg7, i64* %call_arg_ptr1
  %arg8 = load i8, i8* %0
  store i8 %arg8, i8* %call_arg_ptr2
  %call_arg = load %adt, %adt* %call_arg_ptr
  %call_arg9 = load i64, i64* %call_arg_ptr1
  %call_arg10 = load i8, i8* %call_arg_ptr2
  %call_ret = call %adt @set_bit(%adt %call_arg, i64 %call_arg9, i8 %call_arg10)
  store %adt %call_ret, %adt* %rval_ptr
  ret %adt* %rval_ptr
}

define %clos* @__open_file__(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%adt* (i8*, i8**)* @__open_file__5981029516721887925 to i8*
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt,
  %closarg_ptr = bitcast i8* %malloccall2 to %adt*
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %closarg_ptr
  %raw_arg_ptr = bitcast %adt* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %adt* @__open_file__5981029516721887925(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
```

```
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %call_arg_ptr = alloca %adt
  %call_arg_ptr1 = alloca %adt
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %call_arg_ptr
  %arg_ptr2 = bitcast i8* %0 to %adt*
  %arg3 = load %adt, %adt* %arg_ptr2
  store %adt %arg3, %adt* %call_arg_ptr1
  %call_arg = load %adt, %adt* %call_arg_ptr
  %call_arg4 = load %adt, %adt* %call_arg_ptr1
  %call_ret = call %adt @open_file(%adt %call_arg, %adt %call_arg4)
  store %adt %call_ret, %adt* %rval_ptr
  ret %adt* %rval_ptr
}

define %adt* @__read_file__(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %call_arg_ptr = alloca %adt
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %call_arg_ptr
  %call_arg = load %adt, %adt* %call_arg_ptr
  %call_ret = call %adt @read_file(%adt %call_arg)
  store %adt %call_ret, %adt* %rval_ptr
  ret %adt* %rval_ptr
}

define %adt* @__close_file__(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %call_arg_ptr = alloca %adt
```

```
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %call_arg_ptr
  %call_arg = load %adt, %adt* %call_arg_ptr
  %call_ret = call %adt @close_file(%adt %call_arg)
  store %adt %call_ret, %adt* %rval_ptr
  ret %adt* %rval_ptr
}

define %clos* @len(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (i64* (i8*, i8**)* @fn_len to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall1 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %clos* @printString(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%adt* (i8*, i8**)* @__print_string__ to i8*), i8** %clos_fn
  %malloccall1 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall1 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %clos* @string_to_mem(i8* %0, i8** %1) {
```

```
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%adt* (i8*, i8**)* @fn_string_to_mem to i8*), i8** %clos_fn
  %malloccall1 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall1 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %clos* @openFile(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @fn_openFile to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall1 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %clos* @readFile(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%adt* (i8*, i8**)* @__read_file__ to i8*), i8** %clos_fn_pt
  %malloccall1 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall1 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
```

```
  ret %clos* %rval_ptr
}

define %clos* @closeFile(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%adt* (i8*, i8**)* @__close_file__ to i8*), i8** %clos_fn_p
  %malloccall1 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall1 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %clos* @const(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @fn_const to i8*), i8** %clos_fn_ptr
  %malloccall1 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall1 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %clos* @printFile(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @fn_printFile to i8*), i8** %clos_fn_pt
  %malloccall1 = tail call i8* @malloc(i32 0)
```

```
  %env_aloc = bitcast i8* %malloccall1 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define i64* @fn_len(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i64* getelementptr (i64, i64
  %rval_ptr = bitcast i8* %malloccall to i64*
  %scrut = alloca %adt
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %scrut
  %switch_tag_ptr = getelementptr inbounds %adt, %adt* %scrut, i32 0, i32 0
  %scrut_data = getelementptr inbounds %adt, %adt* %scrut, i32 0, i32 1
  %switch_tag = load i64, i64* %switch_tag_ptr
  switch i64 %switch_tag, label %default [
    i64 1, label %case
  ]

default:                                          ; preds = %entry
  store i64 0, i64* %rval_ptr
  br label %case_continue

case:                                             ; preds = %entry
  %scrut_data_deref = load i8*, i8** %scrut_data
  %cons_cast = bitcast i8* %scrut_data_deref to %C*
  %cons_destruct_ptr = getelementptr inbounds %C, %C* %cons_cast, i32 0, i32 0
  %cons_destruct = load i8, i8* %cons_destruct_ptr
  %cons_destruct_ptr1 = getelementptr inbounds %C, %C* %cons_cast, i32 0, i32 1
  %cons_destruct2 = load %adt, %adt* %cons_destruct_ptr1
  %app_lhs = alloca %clos
  %app_lhs3 = alloca %clos
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %app_lhs3, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @__prim__plus to i8*), i8** %clos_fn_pt
  %malloccall4 = tail call i8* @malloc(i32 0)
```

```
%env_aloc = bitcast i8* %malloccall4 to [0 x i8*]*
%env = getelementptr inbounds %clos, %clos* %app_lhs3, i32 0, i32 1
%env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
store i8** %env_ptr_raw, i8*** %env
%app_rhs = alloca i64
store i64 1, i64* %app_rhs
%raw_app_rhs = bitcast i64* %app_rhs to i8*
%raw_fn_ptr_ptr = getelementptr inbounds %clos, %clos* %app_lhs3, i32 0, i32 0
%raw_fn_ptr = load i8*, i8** %raw_fn_ptr_ptr
%fn_ptr = bitcast i8* %raw_fn_ptr to %clos* (i8*, i8**)*
%args_ptr = getelementptr inbounds %clos, %clos* %app_lhs3, i32 0, i32 1
%args = load i8**, i8*** %args_ptr
%app_res_ptr = call %clos* %fn_ptr(i8* %raw_app_rhs, i8** %args)
%app_res = load %clos, %clos* %app_res_ptr
store %clos %app_res, %clos* %app_lhs
%app_rhs5 = alloca i64
%app_lhs6 = alloca %clos
%app_lhs7 = alloca %clos
%clos_fn_ptr8 = getelementptr inbounds %clos, %clos* %app_lhs7, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @len to i8*), i8** %clos_fn_ptr8
%malloccall9 = tail call i8* @malloc(i32 0)
%env_aloc10 = bitcast i8* %malloccall9 to [0 x i8*]*
%env11 = getelementptr inbounds %clos, %clos* %app_lhs7, i32 0, i32 1
%env_ptr_raw12 = bitcast [0 x i8*]* %env_aloc10 to i8**
store i8** %env_ptr_raw12, i8*** %env11
%app_rhs13 = alloca i64
store i64 0, i64* %app_rhs13
%raw_app_rhs14 = bitcast i64* %app_rhs13 to i8*
%raw_fn_ptr_ptr15 = getelementptr inbounds %clos, %clos* %app_lhs7, i32 0, i32
%raw_fn_ptr16 = load i8*, i8** %raw_fn_ptr_ptr15
%fn_ptr17 = bitcast i8* %raw_fn_ptr16 to %clos* (i8*, i8**)*
%args_ptr18 = getelementptr inbounds %clos, %clos* %app_lhs7, i32 0, i32 1
%args19 = load i8**, i8*** %args_ptr18
%app_res_ptr20 = call %clos* %fn_ptr17(i8* %raw_app_rhs14, i8** %args19)
%app_res21 = load %clos, %clos* %app_res_ptr20
store %clos %app_res21, %clos* %app_lhs6
%app_rhs22 = alloca %adt
store %adt %cons_destruct2, %adt* %app_rhs22
```

```
  %raw_app_rhs23 = bitcast %adt* %app_rhs22 to i8*
  %raw_fn_ptr_ptr24 = getelementptr inbounds %clos, %clos* %app_lhs6, i32 0, i32
  %raw_fn_ptr25 = load i8*, i8** %raw_fn_ptr_ptr24
  %fn_ptr26 = bitcast i8* %raw_fn_ptr25 to i64* (i8*, i8**)*
  %args_ptr27 = getelementptr inbounds %clos, %clos* %app_lhs6, i32 0, i32 1
  %args28 = load i8**, i8*** %args_ptr27
  %app_res_ptr29 = call i64* %fn_ptr26(i8* %raw_app_rhs23, i8** %args28)
  %app_res30 = load i64, i64* %app_res_ptr29
  store i64 %app_res30, i64* %app_rhs5
  %raw_app_rhs31 = bitcast i64* %app_rhs5 to i8*
  %raw_fn_ptr_ptr32 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32
  %raw_fn_ptr33 = load i8*, i8** %raw_fn_ptr_ptr32
  %fn_ptr34 = bitcast i8* %raw_fn_ptr33 to i64* (i8*, i8**)*
  %args_ptr35 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
  %args36 = load i8**, i8*** %args_ptr35
  %app_res_ptr37 = call i64* %fn_ptr34(i8* %raw_app_rhs31, i8** %args36)
  %app_res38 = load i64, i64* %app_res_ptr37
  store i64 %app_res38, i64* %rval_ptr
  br label %case_continue

case_continue:                                    ; preds = %case, %default
  ret i64* %rval_ptr
}

define %adt* @fn_string_to_mem(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %app_lhs = alloca %clos
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 0
  store i8* bitcast (%adt* (i8*, i8**)* @string_to_mem628303324353860165 to i8*)
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt,
  %closarg_ptr = bitcast i8* %malloccall2 to %adt*
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg = load %adt, %adt* %arg_ptr
```

```
store %adt %arg, %adt* %closarg_ptr
%raw_arg_ptr = bitcast %adt* %closarg_ptr to i8*
store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
%env = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
%env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
store i8** %env_ptr_raw, i8*** %env
%app_rhs = alloca %adt
%app_lhs3 = alloca %clos
%clos_fn_ptr4 = getelementptr inbounds %clos, %clos* %app_lhs3, i32 0, i32 0
store i8* bitcast (%adt* (i8*, i8**)* @__prim_alloc__ to i8*), i8** %clos_fn_p
%malloccall5 = tail call i8* @malloc(i32 0)
%env_aloc6 = bitcast i8* %malloccall5 to [0 x i8*]*
%env7 = getelementptr inbounds %clos, %clos* %app_lhs3, i32 0, i32 1
%env_ptr_raw8 = bitcast [0 x i8*]* %env_aloc6 to i8**
store i8** %env_ptr_raw8, i8*** %env7
%app_rhs9 = alloca i64
%app_lhs10 = alloca %clos
%app_lhs11 = alloca %clos
%clos_fn_ptr12 = getelementptr inbounds %clos, %clos* %app_lhs11, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @__prim__plus to i8*), i8** %clos_fn_pt
%malloccall13 = tail call i8* @malloc(i32 0)
%env_aloc14 = bitcast i8* %malloccall13 to [0 x i8*]*
%env15 = getelementptr inbounds %clos, %clos* %app_lhs11, i32 0, i32 1
%env_ptr_raw16 = bitcast [0 x i8*]* %env_aloc14 to i8**
store i8** %env_ptr_raw16, i8*** %env15
%app_rhs17 = alloca i64
store i64 1, i64* %app_rhs17
%raw_app_rhs = bitcast i64* %app_rhs17 to i8*
%raw_fn_ptr_ptr = getelementptr inbounds %clos, %clos* %app_lhs11, i32 0, i32
%raw_fn_ptr = load i8*, i8** %raw_fn_ptr_ptr
%fn_ptr = bitcast i8* %raw_fn_ptr to %clos* (i8*, i8**)*
%args_ptr = getelementptr inbounds %clos, %clos* %app_lhs11, i32 0, i32 1
%args = load i8**, i8*** %args_ptr
%app_res_ptr = call %clos* %fn_ptr(i8* %raw_app_rhs, i8** %args)
%app_res = load %clos, %clos* %app_res_ptr
store %clos %app_res, %clos* %app_lhs10
%app_rhs18 = alloca i64
%app_lhs19 = alloca %clos
```

```
%app_lhs20 = alloca %clos
%clos_fn_ptr21 = getelementptr inbounds %clos, %clos* %app_lhs20, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @len to i8*), i8** %clos_fn_ptr21
%malloccall22 = tail call i8* @malloc(i32 0)
%env_aloc23 = bitcast i8* %malloccall22 to [0 x i8*]*
%env24 = getelementptr inbounds %clos, %clos* %app_lhs20, i32 0, i32 1
%env_ptr_raw25 = bitcast [0 x i8*]* %env_aloc23 to i8**
store i8** %env_ptr_raw25, i8*** %env24
%app_rhs26 = alloca i64
store i64 0, i64* %app_rhs26
%raw_app_rhs27 = bitcast i64* %app_rhs26 to i8*
%raw_fn_ptr_ptr28 = getelementptr inbounds %clos, %clos* %app_lhs20, i32 0, i3
%raw_fn_ptr29 = load i8*, i8** %raw_fn_ptr_ptr28
%fn_ptr30 = bitcast i8* %raw_fn_ptr29 to %clos* (i8*, i8**)*
%args_ptr31 = getelementptr inbounds %clos, %clos* %app_lhs20, i32 0, i32 1
%args32 = load i8**, i8*** %args_ptr31
%app_res_ptr33 = call %clos* %fn_ptr30(i8* %raw_app_rhs27, i8** %args32)
%app_res34 = load %clos, %clos* %app_res_ptr33
store %clos %app_res34, %clos* %app_lhs19
%app_rhs35 = alloca %adt
%arg_ptr36 = bitcast i8* %0 to %adt*
%arg37 = load %adt, %adt* %arg_ptr36
store %adt %arg37, %adt* %app_rhs35
%raw_app_rhs38 = bitcast %adt* %app_rhs35 to i8*
%raw_fn_ptr_ptr39 = getelementptr inbounds %clos, %clos* %app_lhs19, i32 0, i3
%raw_fn_ptr40 = load i8*, i8** %raw_fn_ptr_ptr39
%fn_ptr41 = bitcast i8* %raw_fn_ptr40 to i64* (i8*, i8**)*
%args_ptr42 = getelementptr inbounds %clos, %clos* %app_lhs19, i32 0, i32 1
%args43 = load i8**, i8*** %args_ptr42
%app_res_ptr44 = call i64* %fn_ptr41(i8* %raw_app_rhs38, i8** %args43)
%app_res45 = load i64, i64* %app_res_ptr44
store i64 %app_res45, i64* %app_rhs18
%raw_app_rhs46 = bitcast i64* %app_rhs18 to i8*
%raw_fn_ptr_ptr47 = getelementptr inbounds %clos, %clos* %app_lhs10, i32 0, i3
%raw_fn_ptr48 = load i8*, i8** %raw_fn_ptr_ptr47
%fn_ptr49 = bitcast i8* %raw_fn_ptr48 to i64* (i8*, i8**)*
%args_ptr50 = getelementptr inbounds %clos, %clos* %app_lhs10, i32 0, i32 1
%args51 = load i8**, i8*** %args_ptr50
```

```
  %app_res_ptr52 = call i64* %fn_ptr49(i8* %raw_app_rhs46, i8** %args51)
  %app_res53 = load i64, i64* %app_res_ptr52
  store i64 %app_res53, i64* %app_rhs9
  %raw_app_rhs54 = bitcast i64* %app_rhs9 to i8*
  %raw_fn_ptr_ptr55 = getelementptr inbounds %clos, %clos* %app_lhs3, i32 0, i32
  %raw_fn_ptr56 = load i8*, i8** %raw_fn_ptr_ptr55
  %fn_ptr57 = bitcast i8* %raw_fn_ptr56 to %adt* (i8*, i8**)*
  %args_ptr58 = getelementptr inbounds %clos, %clos* %app_lhs3, i32 0, i32 1
  %args59 = load i8**, i8*** %args_ptr58
  %app_res_ptr60 = call %adt* %fn_ptr57(i8* %raw_app_rhs54, i8** %args59)
  %app_res61 = load %adt, %adt* %app_res_ptr60
  store %adt %app_res61, %adt* %app_rhs
  %raw_app_rhs62 = bitcast %adt* %app_rhs to i8*
  %raw_fn_ptr_ptr63 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32
  %raw_fn_ptr64 = load i8*, i8** %raw_fn_ptr_ptr63
  %fn_ptr65 = bitcast i8* %raw_fn_ptr64 to %adt* (i8*, i8**)*
  %args_ptr66 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
  %args67 = load i8**, i8*** %args_ptr66
  %app_res_ptr68 = call %adt* %fn_ptr65(i8* %raw_app_rhs62, i8** %args67)
  %app_res69 = load %adt, %adt* %app_res_ptr68
  store %adt %app_res69, %adt* %rval_ptr
  ret %adt* %rval_ptr
}

define %adt* @string_to_mem628303324353860165(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %app_lhs = alloca %clos
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 0
  store i8* bitcast (%adt* (i8*, i8**)* @string_to_mem500843661005760695 5 to i8*
  %malloccall1 = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1*
  %env_aloc = bitcast i8* %malloccall1 to [2 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [2 x i8*], [2 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt,
  %closarg_ptr = bitcast i8* %malloccall2 to %adt*
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg = load %adt, %adt* %arg_ptr
```

```
store %adt %arg, %adt* %closarg_ptr
%raw_arg_ptr = bitcast %adt* %closarg_ptr to i8*
store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
%raw_closarg_ptr_ptr3 = getelementptr inbounds [2 x i8*], [2 x i8*]* %env_aloc
%malloccall4 = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt,
%closarg_ptr5 = bitcast i8* %malloccall4 to %adt*
%arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
%raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
%raw_arg_ptr6 = load i8*, i8** %raw_arg_ptr_ptr
%arg_ptr7 = bitcast i8* %raw_arg_ptr6 to %adt*
%arg8 = load %adt, %adt* %arg_ptr7
store %adt %arg8, %adt* %closarg_ptr5
%raw_arg_ptr9 = bitcast %adt* %closarg_ptr5 to i8*
store i8* %raw_arg_ptr9, i8** %raw_closarg_ptr_ptr3
%env = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
%env_ptr_raw = bitcast [2 x i8*]* %env_aloc to i8**
store i8** %env_ptr_raw, i8*** %env
%app_rhs = alloca %clos
%app_lhs10 = alloca %clos
%clos_fn_ptr11 = getelementptr inbounds %clos, %clos* %app_lhs10, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @string_to_mem3602583590299280641 to i8
%malloccall12 = tail call i8* @malloc(i32 0)
%env_aloc13 = bitcast i8* %malloccall12 to [0 x i8*]*
%env14 = getelementptr inbounds %clos, %clos* %app_lhs10, i32 0, i32 1
%env_ptr_raw15 = bitcast [0 x i8*]* %env_aloc13 to i8**
store i8** %env_ptr_raw15, i8*** %env14
%app_rhs16 = alloca i64
store i64 0, i64* %app_rhs16
%raw_app_rhs = bitcast i64* %app_rhs16 to i8*
%raw_fn_ptr_ptr = getelementptr inbounds %clos, %clos* %app_lhs10, i32 0, i32
%raw_fn_ptr = load i8*, i8** %raw_fn_ptr_ptr
%fn_ptr = bitcast i8* %raw_fn_ptr to %clos* (i8*, i8**)*
%args_ptr = getelementptr inbounds %clos, %clos* %app_lhs10, i32 0, i32 1
%args = load i8**, i8*** %args_ptr
%app_res_ptr = call %clos* %fn_ptr(i8* %raw_app_rhs, i8** %args)
%app_res = load %clos, %clos* %app_res_ptr
store %clos %app_res, %clos* %app_rhs
%raw_app_rhs17 = bitcast %clos* %app_rhs to i8*
```

```
  %raw_fn_ptr_ptr18 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32
  %raw_fn_ptr19 = load i8*, i8** %raw_fn_ptr_ptr18
  %fn_ptr20 = bitcast i8* %raw_fn_ptr19 to %adt* (i8*, i8**)*
  %args_ptr21 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
  %args22 = load i8**, i8*** %args_ptr21
  %app_res_ptr23 = call %adt* %fn_ptr20(i8* %raw_app_rhs17, i8** %args22)
  %app_res24 = load %adt, %adt* %app_res_ptr23
  store %adt %app_res24, %adt* %rval_ptr
  ret %adt* %rval_ptr
}

define %adt* @string_to_mem500843661005760695 5(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %app_lhs = alloca %clos
  %app_lhs1 = alloca %clos
  %app_lhs2 = alloca %clos
  %arg_ptr = bitcast i8* %0 to %clos*
  %arg = load %clos, %clos* %arg_ptr
  store %clos %arg, %clos* %app_lhs2
  %app_rhs = alloca %adt
  %arg_gep_ptr = bitcast i8** %1 to [2 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [2 x i8*], [2 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr3 = bitcast i8* %raw_arg_ptr to %adt*
  %arg4 = load %adt, %adt* %arg_ptr3
  store %adt %arg4, %adt* %app_rhs
  %raw_app_rhs = bitcast %adt* %app_rhs to i8*
  %raw_fn_ptr_ptr = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 0
  %raw_fn_ptr = load i8*, i8** %raw_fn_ptr_ptr
  %fn_ptr = bitcast i8* %raw_fn_ptr to %clos* (i8*, i8**)*
  %args_ptr = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 1
  %args = load i8**, i8*** %args_ptr
  %app_res_ptr = call %clos* %fn_ptr(i8* %raw_app_rhs, i8** %args)
  %app_res = load %clos, %clos* %app_res_ptr
  store %clos %app_res, %clos* %app_lhs1
  %app_rhs5 = alloca %adt
```

```
  %arg_gep_ptr6 = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr7 = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr6, i32 0,
  %raw_arg_ptr8 = load i8*, i8** %raw_arg_ptr_ptr7
  %arg_ptr9 = bitcast i8* %raw_arg_ptr8 to %adt*
  %arg10 = load %adt, %adt* %arg_ptr9
  store %adt %arg10, %adt* %app_rhs5
  %raw_app_rhs11 = bitcast %adt* %app_rhs5 to i8*
  %raw_fn_ptr_ptr12 = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32
  %raw_fn_ptr13 = load i8*, i8** %raw_fn_ptr_ptr12
  %fn_ptr14 = bitcast i8* %raw_fn_ptr13 to %clos* (i8*, i8**)*
  %args_ptr15 = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 1
  %args16 = load i8**, i8*** %args_ptr15
  %app_res_ptr17 = call %clos* %fn_ptr14(i8* %raw_app_rhs11, i8** %args16)
  %app_res18 = load %clos, %clos* %app_res_ptr17
  store %clos %app_res18, %clos* %app_lhs
  %app_rhs19 = alloca i64
  store i64 0, i64* %app_rhs19
  %raw_app_rhs20 = bitcast i64* %app_rhs19 to i8*
  %raw_fn_ptr_ptr21 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32
  %raw_fn_ptr22 = load i8*, i8** %raw_fn_ptr_ptr21
  %fn_ptr23 = bitcast i8* %raw_fn_ptr22 to %adt* (i8*, i8**)*
  %args_ptr24 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
  %args25 = load i8**, i8*** %args_ptr24
  %app_res_ptr26 = call %adt* %fn_ptr23(i8* %raw_app_rhs20, i8** %args25)
  %app_res27 = load %adt, %adt* %app_res_ptr26
  store %adt %app_res27, %adt* %rval_ptr
  ret %adt* %rval_ptr
}

define %clos* @string_to_mem36025835902992806 41(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @fn_string_to_mem5134325039177141690 to
  %malloccall1 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall1 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
```

```
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %clos* @fn_string_to_mem5134325039177141690(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @fn_string_to_mem5509272086977288127 to
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt,
  %closarg_ptr = bitcast i8* %malloccall2 to %adt*
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %closarg_ptr
  %raw_arg_ptr = bitcast %adt* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %clos* @fn_string_to_mem5509272086977288127(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%adt* (i8*, i8**)* @fn_string_to_mem4100419911398359694 to
  %malloccall1 = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1*
  %env_aloc = bitcast i8* %malloccall1 to [2 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [2 x i8*], [2 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt,
  %closarg_ptr = bitcast i8* %malloccall2 to %adt*
```

```
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %closarg_ptr
  %raw_arg_ptr = bitcast %adt* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %raw_closarg_ptr_ptr3 = getelementptr inbounds [2 x i8*], [2 x i8*]* %env_aloc
  %malloccall4 = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt,
  %closarg_ptr5 = bitcast i8* %malloccall4 to %adt*
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr6 = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr7 = bitcast i8* %raw_arg_ptr6 to %adt*
  %arg8 = load %adt, %adt* %arg_ptr7
  store %adt %arg8, %adt* %closarg_ptr5
  %raw_arg_ptr9 = bitcast %adt* %closarg_ptr5 to i8*
  store i8* %raw_arg_ptr9, i8** %raw_closarg_ptr_ptr3
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [2 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %adt* @fn_string_to_mem41004199113983596 94(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %scrut = alloca %adt
  %arg_gep_ptr = bitcast i8** %1 to [2 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [2 x i8*], [2 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %scrut
  %switch_tag_ptr = getelementptr inbounds %adt, %adt* %scrut, i32 0, i32 0
  %scrut_data = getelementptr inbounds %adt, %adt* %scrut, i32 0, i32 1
  %switch_tag = load i64, i64* %switch_tag_ptr
  switch i64 %switch_tag, label %default [
    i64 1, label %case
```

```
  ]

default:                                          ; preds = %entry
  %app_lhs = alloca %clos
  %app_lhs1 = alloca %clos
  %app_lhs2 = alloca %clos
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @__set_bit__ to i8*), i8** %clos_fn_ptr
  %malloccall3 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall3 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  %app_rhs = alloca %adt
  %arg_gep_ptr4 = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr5 = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr4, i32 0,
  %raw_arg_ptr6 = load i8*, i8** %raw_arg_ptr_ptr5
  %arg_ptr7 = bitcast i8* %raw_arg_ptr6 to %adt*
  %arg8 = load %adt, %adt* %arg_ptr7
  store %adt %arg8, %adt* %app_rhs
  %raw_app_rhs = bitcast %adt* %app_rhs to i8*
  %raw_fn_ptr_ptr = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 0
  %raw_fn_ptr = load i8*, i8** %raw_fn_ptr_ptr
  %fn_ptr = bitcast i8* %raw_fn_ptr to %clos* (i8*, i8**)*
  %args_ptr = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 1
  %args = load i8**, i8*** %args_ptr
  %app_res_ptr = call %clos* %fn_ptr(i8* %raw_app_rhs, i8** %args)
  %app_res = load %clos, %clos* %app_res_ptr
  store %clos %app_res, %clos* %app_lhs1
  %app_rhs9 = alloca i64
  %arg_ptr10 = bitcast i8* %0 to i64*
  %arg11 = load i64, i64* %arg_ptr10
  store i64 %arg11, i64* %app_rhs9
  %raw_app_rhs12 = bitcast i64* %app_rhs9 to i8*
  %raw_fn_ptr_ptr13 = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32
  %raw_fn_ptr14 = load i8*, i8** %raw_fn_ptr_ptr13
  %fn_ptr15 = bitcast i8* %raw_fn_ptr14 to %clos* (i8*, i8**)*
  %args_ptr16 = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 1
```

```
  %args17 = load i8**, i8*** %args_ptr16
  %app_res_ptr18 = call %clos* %fn_ptr15(i8* %raw_app_rhs12, i8** %args17)
  %app_res19 = load %clos, %clos* %app_res_ptr18
  store %clos %app_res19, %clos* %app_lhs
  %app_rhs20 = alloca i8
  store i8 0, i8* %app_rhs20
  %raw_fn_ptr_ptr21 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32
  %raw_fn_ptr22 = load i8*, i8** %raw_fn_ptr_ptr21
  %fn_ptr23 = bitcast i8* %raw_fn_ptr22 to %adt* (i8*, i8**)*
  %args_ptr24 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
  %args25 = load i8**, i8*** %args_ptr24
  %app_res_ptr26 = call %adt* %fn_ptr23(i8* %app_rhs20, i8** %args25)
  %app_res27 = load %adt, %adt* %app_res_ptr26
  store %adt %app_res27, %adt* %rval_ptr
  br label %case_continue

case:                                             ; preds = %entry
  %scrut_data_deref = load i8*, i8** %scrut_data
  %cons_cast = bitcast i8* %scrut_data_deref to %C*
  %cons_destruct_ptr = getelementptr inbounds %C, %C* %cons_cast, i32 0, i32 0
  %cons_destruct = load i8, i8* %cons_destruct_ptr
  %cons_destruct_ptr28 = getelementptr inbounds %C, %C* %cons_cast, i32 0, i32 1
  %cons_destruct29 = load %adt, %adt* %cons_destruct_ptr28
  %app_lhs30 = alloca %clos
  %app_lhs31 = alloca %clos
  %app_lhs32 = alloca %clos
  %app_lhs33 = alloca %clos
  %clos_fn_ptr34 = getelementptr inbounds %clos, %clos* %app_lhs33, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @string_to_mem3602583590299280641 to i8
  %malloccall35 = tail call i8* @malloc(i32 0)
  %env_aloc36 = bitcast i8* %malloccall35 to [0 x i8*]*
  %env37 = getelementptr inbounds %clos, %clos* %app_lhs33, i32 0, i32 1
  %env_ptr_raw38 = bitcast [0 x i8*]* %env_aloc36 to i8**
  store i8** %env_ptr_raw38, i8*** %env37
  %app_rhs39 = alloca i64
  store i64 0, i64* %app_rhs39
  %raw_app_rhs40 = bitcast i64* %app_rhs39 to i8*
  %raw_fn_ptr_ptr41 = getelementptr inbounds %clos, %clos* %app_lhs33, i32 0, i3
```

```
%raw_fn_ptr42 = load i8*, i8** %raw_fn_ptr_ptr41
%fn_ptr43 = bitcast i8* %raw_fn_ptr42 to %clos* (i8*, i8**)*
%args_ptr44 = getelementptr inbounds %clos, %clos* %app_lhs33, i32 0, i32 1
%args45 = load i8**, i8*** %args_ptr44
%app_res_ptr46 = call %clos* %fn_ptr43(i8* %raw_app_rhs40, i8** %args45)
%app_res47 = load %clos, %clos* %app_res_ptr46
store %clos %app_res47, %clos* %app_lhs32
%app_rhs48 = alloca %adt
store %adt %cons_destruct29, %adt* %app_rhs48
%raw_app_rhs49 = bitcast %adt* %app_rhs48 to i8*
%raw_fn_ptr_ptr50 = getelementptr inbounds %clos, %clos* %app_lhs32, i32 0, i3
%raw_fn_ptr51 = load i8*, i8** %raw_fn_ptr_ptr50
%fn_ptr52 = bitcast i8* %raw_fn_ptr51 to %clos* (i8*, i8**)*
%args_ptr53 = getelementptr inbounds %clos, %clos* %app_lhs32, i32 0, i32 1
%args54 = load i8**, i8*** %args_ptr53
%app_res_ptr55 = call %clos* %fn_ptr52(i8* %raw_app_rhs49, i8** %args54)
%app_res56 = load %clos, %clos* %app_res_ptr55
store %clos %app_res56, %clos* %app_lhs31
%app_rhs57 = alloca %adt
%app_lhs58 = alloca %clos
%app_lhs59 = alloca %clos
%app_lhs60 = alloca %clos
%clos_fn_ptr61 = getelementptr inbounds %clos, %clos* %app_lhs60, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @__set_bit__ to i8*), i8** %clos_fn_ptr
%malloccall62 = tail call i8* @malloc(i32 0)
%env_aloc63 = bitcast i8* %malloccall62 to [0 x i8*]*
%env64 = getelementptr inbounds %clos, %clos* %app_lhs60, i32 0, i32 1
%env_ptr_raw65 = bitcast [0 x i8*]* %env_aloc63 to i8**
store i8** %env_ptr_raw65, i8*** %env64
%app_rhs66 = alloca %adt
%arg_gep_ptr67 = bitcast i8** %1 to [1 x i8*]*
%raw_arg_ptr_ptr68 = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr67, i32 0
%raw_arg_ptr69 = load i8*, i8** %raw_arg_ptr_ptr68
%arg_ptr70 = bitcast i8* %raw_arg_ptr69 to %adt*
%arg71 = load %adt, %adt* %arg_ptr70
store %adt %arg71, %adt* %app_rhs66
%raw_app_rhs72 = bitcast %adt* %app_rhs66 to i8*
%raw_fn_ptr_ptr73 = getelementptr inbounds %clos, %clos* %app_lhs60, i32 0, i3
```

```
%raw_fn_ptr74 = load i8*, i8** %raw_fn_ptr_ptr73
%fn_ptr75 = bitcast i8* %raw_fn_ptr74 to %clos* (i8*, i8**)*
%args_ptr76 = getelementptr inbounds %clos, %clos* %app_lhs60, i32 0, i32 1
%args77 = load i8**, i8*** %args_ptr76
%app_res_ptr78 = call %clos* %fn_ptr75(i8* %raw_app_rhs72, i8** %args77)
%app_res79 = load %clos, %clos* %app_res_ptr78
store %clos %app_res79, %clos* %app_lhs59
%app_rhs80 = alloca i64
%arg_ptr81 = bitcast i8* %0 to i64*
%arg82 = load i64, i64* %arg_ptr81
store i64 %arg82, i64* %app_rhs80
%raw_app_rhs83 = bitcast i64* %app_rhs80 to i8*
%raw_fn_ptr_ptr84 = getelementptr inbounds %clos, %clos* %app_lhs59, i32 0, i3
%raw_fn_ptr85 = load i8*, i8** %raw_fn_ptr_ptr84
%fn_ptr86 = bitcast i8* %raw_fn_ptr85 to %clos* (i8*, i8**)*
%args_ptr87 = getelementptr inbounds %clos, %clos* %app_lhs59, i32 0, i32 1
%args88 = load i8**, i8*** %args_ptr87
%app_res_ptr89 = call %clos* %fn_ptr86(i8* %raw_app_rhs83, i8** %args88)
%app_res90 = load %clos, %clos* %app_res_ptr89
store %clos %app_res90, %clos* %app_lhs58
%app_rhs91 = alloca i8
store i8 %cons_destruct, i8* %app_rhs91
%raw_fn_ptr_ptr92 = getelementptr inbounds %clos, %clos* %app_lhs58, i32 0, i3
%raw_fn_ptr93 = load i8*, i8** %raw_fn_ptr_ptr92
%fn_ptr94 = bitcast i8* %raw_fn_ptr93 to %adt* (i8*, i8**)*
%args_ptr95 = getelementptr inbounds %clos, %clos* %app_lhs58, i32 0, i32 1
%args96 = load i8**, i8*** %args_ptr95
%app_res_ptr97 = call %adt* %fn_ptr94(i8* %app_rhs91, i8** %args96)
%app_res98 = load %adt, %adt* %app_res_ptr97
store %adt %app_res98, %adt* %app_rhs57
%raw_app_rhs99 = bitcast %adt* %app_rhs57 to i8*
%raw_fn_ptr_ptr100 = getelementptr inbounds %clos, %clos* %app_lhs31, i32 0, i
%raw_fn_ptr101 = load i8*, i8** %raw_fn_ptr_ptr100
%fn_ptr102 = bitcast i8* %raw_fn_ptr101 to %clos* (i8*, i8**)*
%args_ptr103 = getelementptr inbounds %clos, %clos* %app_lhs31, i32 0, i32 1
%args104 = load i8**, i8*** %args_ptr103
%app_res_ptr105 = call %clos* %fn_ptr102(i8* %raw_app_rhs99, i8** %args104)
%app_res106 = load %clos, %clos* %app_res_ptr105
```

```
store %clos %app_res106, %clos* %app_lhs30
%app_rhs107 = alloca i64
%app_lhs108 = alloca %clos
%app_lhs109 = alloca %clos
%clos_fn_ptr110 = getelementptr inbounds %clos, %clos* %app_lhs109, i32 0, i32
store i8* bitcast (%clos* (i8*, i8**)* @__prim__plus to i8*), i8** %clos_fn_pt
%malloccall111 = tail call i8* @malloc(i32 0)
%env_aloc112 = bitcast i8* %malloccall111 to [0 x i8*]*
%env113 = getelementptr inbounds %clos, %clos* %app_lhs109, i32 0, i32 1
%env_ptr_raw114 = bitcast [0 x i8*]* %env_aloc112 to i8**
store i8** %env_ptr_raw114, i8*** %env113
%app_rhs115 = alloca i64
%arg_ptr116 = bitcast i8* %0 to i64*
%arg117 = load i64, i64* %arg_ptr116
store i64 %arg117, i64* %app_rhs115
%raw_app_rhs118 = bitcast i64* %app_rhs115 to i8*
%raw_fn_ptr_ptr119 = getelementptr inbounds %clos, %clos* %app_lhs109, i32 0,
%raw_fn_ptr120 = load i8*, i8** %raw_fn_ptr_ptr119
%fn_ptr121 = bitcast i8* %raw_fn_ptr120 to %clos* (i8*, i8**)*
%args_ptr122 = getelementptr inbounds %clos, %clos* %app_lhs109, i32 0, i32 1
%args123 = load i8**, i8*** %args_ptr122
%app_res_ptr124 = call %clos* %fn_ptr121(i8* %raw_app_rhs118, i8** %args123)
%app_res125 = load %clos, %clos* %app_res_ptr124
store %clos %app_res125, %clos* %app_lhs108
%app_rhs126 = alloca i64
store i64 1, i64* %app_rhs126
%raw_app_rhs127 = bitcast i64* %app_rhs126 to i8*
%raw_fn_ptr_ptr128 = getelementptr inbounds %clos, %clos* %app_lhs108, i32 0,
%raw_fn_ptr129 = load i8*, i8** %raw_fn_ptr_ptr128
%fn_ptr130 = bitcast i8* %raw_fn_ptr129 to i64* (i8*, i8**)*
%args_ptr131 = getelementptr inbounds %clos, %clos* %app_lhs108, i32 0, i32 1
%args132 = load i8**, i8*** %args_ptr131
%app_res_ptr133 = call i64* %fn_ptr130(i8* %raw_app_rhs127, i8** %args132)
%app_res134 = load i64, i64* %app_res_ptr133
store i64 %app_res134, i64* %app_rhs107
%raw_app_rhs135 = bitcast i64* %app_rhs107 to i8*
%raw_fn_ptr_ptr136 = getelementptr inbounds %clos, %clos* %app_lhs30, i32 0, i
%raw_fn_ptr137 = load i8*, i8** %raw_fn_ptr_ptr136
```

```
  %fn_ptr138 = bitcast i8* %raw_fn_ptr137 to %adt* (i8*, i8**)*
  %args_ptr139 = getelementptr inbounds %clos, %clos* %app_lhs30, i32 0, i32 1
  %args140 = load i8**, i8*** %args_ptr139
  %app_res_ptr141 = call %adt* %fn_ptr138(i8* %raw_app_rhs135, i8** %args140)
  %app_res142 = load %adt, %adt* %app_res_ptr141
  store %adt %app_res142, %adt* %rval_ptr
  br label %case_continue

case_continue:                                    ; preds = %case, %default
  ret %adt* %rval_ptr
}

define %clos* @fn_openFile(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @fn_openFile8489902106938695875 to i8*)
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt,
  %closarg_ptr = bitcast i8* %malloccall2 to %adt*
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %closarg_ptr
  %raw_arg_ptr = bitcast %adt* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %clos* @fn_openFile8489902106938695875(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
```

```
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%adt* (i8*, i8**)* @fn_openFile5447851695057989743 to i8*),
  %malloccall1 = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1*
  %env_aloc = bitcast i8* %malloccall1 to [2 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [2 x i8*], [2 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt,
  %closarg_ptr = bitcast i8* %malloccall2 to %adt*
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %closarg_ptr
  %raw_arg_ptr = bitcast %adt* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %raw_closarg_ptr_ptr3 = getelementptr inbounds [2 x i8*], [2 x i8*]* %env_aloc
  %malloccall4 = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt,
  %closarg_ptr5 = bitcast i8* %malloccall4 to %adt*
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr6 = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr7 = bitcast i8* %raw_arg_ptr6 to %adt*
  %arg8 = load %adt, %adt* %arg_ptr7
  store %adt %arg8, %adt* %closarg_ptr5
  %raw_arg_ptr9 = bitcast %adt* %closarg_ptr5 to i8*
  store i8* %raw_arg_ptr9, i8** %raw_closarg_ptr_ptr3
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [2 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %adt* @fn_openFile5447851695057989743(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %app_lhs = alloca %clos
  %arg_ptr = bitcast i8* %0 to %clos*
  %arg = load %clos, %clos* %arg_ptr
  store %clos %arg, %clos* %app_lhs
  %app_rhs = alloca %adt
```

```
%app_lhs1 = alloca %clos
%app_lhs2 = alloca %clos
%clos_fn_ptr = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @__open_file__ to i8*), i8** %clos_fn_p
%malloccall3 = tail call i8* @malloc(i32 0)
%env_aloc = bitcast i8* %malloccall3 to [0 x i8*]*
%env = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 1
%env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
store i8** %env_ptr_raw, i8*** %env
%app_rhs4 = alloca %adt
%app_lhs5 = alloca %clos
%app_lhs6 = alloca %clos
%clos_fn_ptr7 = getelementptr inbounds %clos, %clos* %app_lhs6, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @string_to_mem to i8*), i8** %clos_fn_p
%malloccall8 = tail call i8* @malloc(i32 0)
%env_aloc9 = bitcast i8* %malloccall8 to [0 x i8*]*
%env10 = getelementptr inbounds %clos, %clos* %app_lhs6, i32 0, i32 1
%env_ptr_raw11 = bitcast [0 x i8*]* %env_aloc9 to i8**
store i8** %env_ptr_raw11, i8*** %env10
%app_rhs12 = alloca i64
store i64 0, i64* %app_rhs12
%raw_app_rhs = bitcast i64* %app_rhs12 to i8*
%raw_fn_ptr_ptr = getelementptr inbounds %clos, %clos* %app_lhs6, i32 0, i32 0
%raw_fn_ptr = load i8*, i8** %raw_fn_ptr_ptr
%fn_ptr = bitcast i8* %raw_fn_ptr to %clos* (i8*, i8**)*
%args_ptr = getelementptr inbounds %clos, %clos* %app_lhs6, i32 0, i32 1
%args = load i8**, i8*** %args_ptr
%app_res_ptr = call %clos* %fn_ptr(i8* %raw_app_rhs, i8** %args)
%app_res = load %clos, %clos* %app_res_ptr
store %clos %app_res, %clos* %app_lhs5
%app_rhs13 = alloca %adt
%arg_gep_ptr = bitcast i8** %1 to [2 x i8*]*
%raw_arg_ptr_ptr = getelementptr [2 x i8*], [2 x i8*]* %arg_gep_ptr, i32 0, i3
%raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
%arg_ptr14 = bitcast i8* %raw_arg_ptr to %adt*
%arg15 = load %adt, %adt* %arg_ptr14
store %adt %arg15, %adt* %app_rhs13
%raw_app_rhs16 = bitcast %adt* %app_rhs13 to i8*
```

```
%raw_fn_ptr_ptr17 = getelementptr inbounds %clos, %clos* %app_lhs5, i32 0, i32
%raw_fn_ptr18 = load i8*, i8** %raw_fn_ptr_ptr17
%fn_ptr19 = bitcast i8* %raw_fn_ptr18 to %adt* (i8*, i8**)*
%args_ptr20 = getelementptr inbounds %clos, %clos* %app_lhs5, i32 0, i32 1
%args21 = load i8**, i8*** %args_ptr20
%app_res_ptr22 = call %adt* %fn_ptr19(i8* %raw_app_rhs16, i8** %args21)
%app_res23 = load %adt, %adt* %app_res_ptr22
store %adt %app_res23, %adt* %app_rhs4
%raw_app_rhs24 = bitcast %adt* %app_rhs4 to i8*
%raw_fn_ptr_ptr25 = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32
%raw_fn_ptr26 = load i8*, i8** %raw_fn_ptr_ptr25
%fn_ptr27 = bitcast i8* %raw_fn_ptr26 to %clos* (i8*, i8**)*
%args_ptr28 = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 1
%args29 = load i8**, i8*** %args_ptr28
%app_res_ptr30 = call %clos* %fn_ptr27(i8* %raw_app_rhs24, i8** %args29)
%app_res31 = load %clos, %clos* %app_res_ptr30
store %clos %app_res31, %clos* %app_lhs1
%app_rhs32 = alloca %adt
%app_lhs33 = alloca %clos
%app_lhs34 = alloca %clos
%clos_fn_ptr35 = getelementptr inbounds %clos, %clos* %app_lhs34, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @string_to_mem to i8*), i8** %clos_fn_p
%malloccall36 = tail call i8* @malloc(i32 0)
%env_aloc37 = bitcast i8* %malloccall36 to [0 x i8*]*
%env38 = getelementptr inbounds %clos, %clos* %app_lhs34, i32 0, i32 1
%env_ptr_raw39 = bitcast [0 x i8*]* %env_aloc37 to i8**
store i8** %env_ptr_raw39, i8*** %env38
%app_rhs40 = alloca i64
store i64 0, i64* %app_rhs40
%raw_app_rhs41 = bitcast i64* %app_rhs40 to i8*
%raw_fn_ptr_ptr42 = getelementptr inbounds %clos, %clos* %app_lhs34, i32 0, i3
%raw_fn_ptr43 = load i8*, i8** %raw_fn_ptr_ptr42
%fn_ptr44 = bitcast i8* %raw_fn_ptr43 to %clos* (i8*, i8**)*
%args_ptr45 = getelementptr inbounds %clos, %clos* %app_lhs34, i32 0, i32 1
%args46 = load i8**, i8*** %args_ptr45
%app_res_ptr47 = call %clos* %fn_ptr44(i8* %raw_app_rhs41, i8** %args46)
%app_res48 = load %clos, %clos* %app_res_ptr47
store %clos %app_res48, %clos* %app_lhs33
```

```
  %app_rhs49 = alloca %adt
  %arg_gep_ptr50 = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr51 = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr50, i32 0
  %raw_arg_ptr52 = load i8*, i8** %raw_arg_ptr_ptr51
  %arg_ptr53 = bitcast i8* %raw_arg_ptr52 to %adt*
  %arg54 = load %adt, %adt* %arg_ptr53
  store %adt %arg54, %adt* %app_rhs49
  %raw_app_rhs55 = bitcast %adt* %app_rhs49 to i8*
  %raw_fn_ptr_ptr56 = getelementptr inbounds %clos, %clos* %app_lhs33, i32 0, i3
  %raw_fn_ptr57 = load i8*, i8** %raw_fn_ptr_ptr56
  %fn_ptr58 = bitcast i8* %raw_fn_ptr57 to %adt* (i8*, i8**)*
  %args_ptr59 = getelementptr inbounds %clos, %clos* %app_lhs33, i32 0, i32 1
  %args60 = load i8**, i8*** %args_ptr59
  %app_res_ptr61 = call %adt* %fn_ptr58(i8* %raw_app_rhs55, i8** %args60)
  %app_res62 = load %adt, %adt* %app_res_ptr61
  store %adt %app_res62, %adt* %app_rhs32
  %raw_app_rhs63 = bitcast %adt* %app_rhs32 to i8*
  %raw_fn_ptr_ptr64 = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32
  %raw_fn_ptr65 = load i8*, i8** %raw_fn_ptr_ptr64
  %fn_ptr66 = bitcast i8* %raw_fn_ptr65 to %adt* (i8*, i8**)*
  %args_ptr67 = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 1
  %args68 = load i8**, i8*** %args_ptr67
  %app_res_ptr69 = call %adt* %fn_ptr66(i8* %raw_app_rhs63, i8** %args68)
  %app_res70 = load %adt, %adt* %app_res_ptr69
  store %adt %app_res70, %adt* %app_rhs
  %raw_app_rhs71 = bitcast %adt* %app_rhs to i8*
  %raw_fn_ptr_ptr72 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32
  %raw_fn_ptr73 = load i8*, i8** %raw_fn_ptr_ptr72
  %fn_ptr74 = bitcast i8* %raw_fn_ptr73 to %adt* (i8*, i8**)*
  %args_ptr75 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
  %args76 = load i8**, i8*** %args_ptr75
  %app_res_ptr77 = call %adt* %fn_ptr74(i8* %raw_app_rhs71, i8** %args76)
  %app_res78 = load %adt, %adt* %app_res_ptr77
  store %adt %app_res78, %adt* %rval_ptr
  ret %adt* %rval_ptr
}

define %clos* @fn_const(i8* %0, i8** %1) {
```

```
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%boxt* (i8*, i8**)* @fn_const6419075023523112259 to i8*), i
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1*
  %closarg_ptr = bitcast i8* %malloccall2 to %boxt*
  %arg_ptr = bitcast i8* %0 to %boxt*
  %arg = load %boxt, %boxt* %arg_ptr
  store %boxt %arg, %boxt* %closarg_ptr
  %raw_arg_ptr = bitcast %boxt* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %boxt* @fn_const6419075023523112259(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %boxt*
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to %boxt*
  %arg = load %boxt, %boxt* %arg_ptr
  store %boxt %arg, %boxt* %rval_ptr
  ret %boxt* %rval_ptr
}

define %clos* @fn_printFile(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
```

```
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%adt* (i8*, i8**)* @fn_printFile2152367932835595560 to i8*)
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt,
  %closarg_ptr = bitcast i8* %malloccall2 to %adt*
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %closarg_ptr
  %raw_arg_ptr = bitcast %adt* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %adt* @fn_printFile2152367932835595560(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %app_lhs = alloca %clos
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 0
  store i8* bitcast (%adt* (i8*, i8**)* @printFile7976305469002585692 to i8*), i
  %malloccall1 = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1*
  %env_aloc = bitcast i8* %malloccall1 to [2 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [2 x i8*], [2 x i8*]* %env_aloc,
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt,
  %closarg_ptr = bitcast i8* %malloccall2 to %adt*
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %closarg_ptr
  %raw_arg_ptr = bitcast %adt* %closarg_ptr to i8*
  store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
  %raw_closarg_ptr_ptr3 = getelementptr inbounds [2 x i8*], [2 x i8*]* %env_aloc
  %malloccall4 = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt,
  %closarg_ptr5 = bitcast i8* %malloccall4 to %adt*
```

```
%arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
%raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
%raw_arg_ptr6 = load i8*, i8** %raw_arg_ptr_ptr
%arg_ptr7 = bitcast i8* %raw_arg_ptr6 to %adt*
%arg8 = load %adt, %adt* %arg_ptr7
store %adt %arg8, %adt* %closarg_ptr5
%raw_arg_ptr9 = bitcast %adt* %closarg_ptr5 to i8*
store i8* %raw_arg_ptr9, i8** %raw_closarg_ptr_ptr3
%env = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
%env_ptr_raw = bitcast [2 x i8*]* %env_aloc to i8**
store i8** %env_ptr_raw, i8*** %env
%app_rhs = alloca %clos
%app_lhs10 = alloca %clos
%clos_fn_ptr11 = getelementptr inbounds %clos, %clos* %app_lhs10, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @printFile8175353689487733453 to i8*),
%malloccall12 = tail call i8* @malloc(i32 0)
%env_aloc13 = bitcast i8* %malloccall12 to [0 x i8*]*
%env14 = getelementptr inbounds %clos, %clos* %app_lhs10, i32 0, i32 1
%env_ptr_raw15 = bitcast [0 x i8*]* %env_aloc13 to i8**
store i8** %env_ptr_raw15, i8*** %env14
%app_rhs16 = alloca i64
store i64 0, i64* %app_rhs16
%raw_app_rhs = bitcast i64* %app_rhs16 to i8*
%raw_fn_ptr_ptr = getelementptr inbounds %clos, %clos* %app_lhs10, i32 0, i32
%raw_fn_ptr = load i8*, i8** %raw_fn_ptr_ptr
%fn_ptr = bitcast i8* %raw_fn_ptr to %clos* (i8*, i8**)*
%args_ptr = getelementptr inbounds %clos, %clos* %app_lhs10, i32 0, i32 1
%args = load i8**, i8*** %args_ptr
%app_res_ptr = call %clos* %fn_ptr(i8* %raw_app_rhs, i8** %args)
%app_res = load %clos, %clos* %app_res_ptr
store %clos %app_res, %clos* %app_rhs
%raw_app_rhs17 = bitcast %clos* %app_rhs to i8*
%raw_fn_ptr_ptr18 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32
%raw_fn_ptr19 = load i8*, i8** %raw_fn_ptr_ptr18
%fn_ptr20 = bitcast i8* %raw_fn_ptr19 to %adt* (i8*, i8**)*
%args_ptr21 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
%args22 = load i8**, i8*** %args_ptr21
%app_res_ptr23 = call %adt* %fn_ptr20(i8* %raw_app_rhs17, i8** %args22)
```

```
  %app_res24 = load %adt, %adt* %app_res_ptr23
  store %adt %app_res24, %adt* %rval_ptr
  ret %adt* %rval_ptr
}

define %adt* @printFile7976305469002585692(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %app_lhs = alloca %clos
  %app_lhs1 = alloca %clos
  %app_lhs2 = alloca %clos
  %app_lhs3 = alloca %clos
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %app_lhs3, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @openFile to i8*), i8** %clos_fn_ptr
  %malloccall4 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall4 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %app_lhs3, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  %app_rhs = alloca i64
  store i64 0, i64* %app_rhs
  %raw_app_rhs = bitcast i64* %app_rhs to i8*
  %raw_fn_ptr_ptr = getelementptr inbounds %clos, %clos* %app_lhs3, i32 0, i32 0
  %raw_fn_ptr = load i8*, i8** %raw_fn_ptr_ptr
  %fn_ptr = bitcast i8* %raw_fn_ptr to %clos* (i8*, i8**)*
  %args_ptr = getelementptr inbounds %clos, %clos* %app_lhs3, i32 0, i32 1
  %args = load i8**, i8*** %args_ptr
  %app_res_ptr = call %clos* %fn_ptr(i8* %raw_app_rhs, i8** %args)
  %app_res = load %clos, %clos* %app_res_ptr
  store %clos %app_res, %clos* %app_lhs2
  %app_rhs5 = alloca %adt
  %arg_gep_ptr = bitcast i8** %1 to [2 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [2 x i8*], [2 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %app_rhs5
```

```
%raw_app_rhs6 = bitcast %adt* %app_rhs5 to i8*
%raw_fn_ptr_ptr7 = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32
%raw_fn_ptr8 = load i8*, i8** %raw_fn_ptr_ptr7
%fn_ptr9 = bitcast i8* %raw_fn_ptr8 to %clos* (i8*, i8**)*
%args_ptr10 = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 1
%args11 = load i8**, i8*** %args_ptr10
%app_res_ptr12 = call %clos* %fn_ptr9(i8* %raw_app_rhs6, i8** %args11)
%app_res13 = load %clos, %clos* %app_res_ptr12
store %clos %app_res13, %clos* %app_lhs1
%app_rhs14 = alloca %adt
%arg_gep_ptr15 = bitcast i8** %1 to [1 x i8*]*
%raw_arg_ptr_ptr16 = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr15, i32 0
%raw_arg_ptr17 = load i8*, i8** %raw_arg_ptr_ptr16
%arg_ptr18 = bitcast i8* %raw_arg_ptr17 to %adt*
%arg19 = load %adt, %adt* %arg_ptr18
store %adt %arg19, %adt* %app_rhs14
%raw_app_rhs20 = bitcast %adt* %app_rhs14 to i8*
%raw_fn_ptr_ptr21 = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32
%raw_fn_ptr22 = load i8*, i8** %raw_fn_ptr_ptr21
%fn_ptr23 = bitcast i8* %raw_fn_ptr22 to %clos* (i8*, i8**)*
%args_ptr24 = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 1
%args25 = load i8**, i8*** %args_ptr24
%app_res_ptr26 = call %clos* %fn_ptr23(i8* %raw_app_rhs20, i8** %args25)
%app_res27 = load %clos, %clos* %app_res_ptr26
store %clos %app_res27, %clos* %app_lhs
%app_rhs28 = alloca %clos
%arg_ptr29 = bitcast i8* %0 to %clos*
%arg30 = load %clos, %clos* %arg_ptr29
store %clos %arg30, %clos* %app_rhs28
%raw_app_rhs31 = bitcast %clos* %app_rhs28 to i8*
%raw_fn_ptr_ptr32 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32
%raw_fn_ptr33 = load i8*, i8** %raw_fn_ptr_ptr32
%fn_ptr34 = bitcast i8* %raw_fn_ptr33 to %adt* (i8*, i8**)*
%args_ptr35 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
%args36 = load i8**, i8*** %args_ptr35
%app_res_ptr37 = call %adt* %fn_ptr34(i8* %raw_app_rhs31, i8** %args36)
%app_res38 = load %adt, %adt* %app_res_ptr37
store %adt %app_res38, %adt* %rval_ptr
```

```
  ret %adt* %rval_ptr
}

define %clos* @printFile8175353689487733453(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
  store i8* bitcast (%adt* (i8*, i8**)* @fn_printFile65752345156188090 58 to i8*)
  %malloccall1 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall1 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %adt* @fn_printFile65752345156188090 58(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %scrut = alloca %adt
  %app_lhs = alloca %clos
  %app_lhs1 = alloca %clos
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @readFile to i8*), i8** %clos_fn_ptr
  %malloccall2 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall2 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  %app_rhs = alloca i64
  store i64 0, i64* %app_rhs
  %raw_app_rhs = bitcast i64* %app_rhs to i8*
  %raw_fn_ptr_ptr = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 0
  %raw_fn_ptr = load i8*, i8** %raw_fn_ptr_ptr
  %fn_ptr = bitcast i8* %raw_fn_ptr to %clos* (i8*, i8**)*
  %args_ptr = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 1
```

```
  %args = load i8**, i8*** %args_ptr
  %app_res_ptr = call %clos* %fn_ptr(i8* %raw_app_rhs, i8** %args)
  %app_res = load %clos, %clos* %app_res_ptr
  store %clos %app_res, %clos* %app_lhs
  %app_rhs3 = alloca %adt
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg = load %adt, %adt* %arg_ptr
  store %adt %arg, %adt* %app_rhs3
  %raw_app_rhs4 = bitcast %adt* %app_rhs3 to i8*
  %raw_fn_ptr_ptr5 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 0
  %raw_fn_ptr6 = load i8*, i8** %raw_fn_ptr_ptr5
  %fn_ptr7 = bitcast i8* %raw_fn_ptr6 to %adt* (i8*, i8**)*
  %args_ptr8 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
  %args9 = load i8**, i8*** %args_ptr8
  %app_res_ptr10 = call %adt* %fn_ptr7(i8* %raw_app_rhs4, i8** %args9)
  %app_res11 = load %adt, %adt* %app_res_ptr10
  store %adt %app_res11, %adt* %scrut
  %switch_tag_ptr = getelementptr inbounds %adt, %adt* %scrut, i32 0, i32 0
  %scrut_data = getelementptr inbounds %adt, %adt* %scrut, i32 0, i32 1
  %switch_tag = load i64, i64* %switch_tag_ptr
  switch i64 %switch_tag, label %default [
    i64 0, label %case
  ]

default:                                          ; preds = %entry
  call void @__die__()
  br label %case_continue

case:                                             ; preds = %entry
  %scrut_data_deref = load i8*, i8** %scrut_data
  %cons_cast = bitcast i8* %scrut_data_deref to %Tuple*
  %cons_destruct_ptr = getelementptr inbounds %Tuple, %Tuple* %cons_cast, i32 0,
  %cons_destruct = load %boxt, %boxt* %cons_destruct_ptr
  %cons_destruct_ptr12 = getelementptr inbounds %Tuple, %Tuple* %cons_cast, i32
  %cons_destruct13 = load %boxt, %boxt* %cons_destruct_ptr12
  %app_lhs14 = alloca %clos
  %clos_fn_ptr15 = getelementptr inbounds %clos, %clos* %app_lhs14, i32 0, i32 0
  store i8* bitcast (%adt* (i8*, i8**)* @printFile3192161587922682755 to i8*), i
```

```
%malloccall16 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i
%env_aloc17 = bitcast i8* %malloccall16 to [1 x i8*]*
%raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc1
%malloccall18 = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1
%closarg_ptr = bitcast i8* %malloccall18 to %boxt*
store %boxt %cons_destruct13, %boxt* %closarg_ptr
%raw_arg_ptr = bitcast %boxt* %closarg_ptr to i8*
store i8* %raw_arg_ptr, i8** %raw_closarg_ptr_ptr
%env19 = getelementptr inbounds %clos, %clos* %app_lhs14, i32 0, i32 1
%env_ptr_raw20 = bitcast [1 x i8*]* %env_aloc17 to i8**
store i8** %env_ptr_raw20, i8*** %env19
%app_rhs21 = alloca %adt
%app_lhs22 = alloca %clos
%app_lhs23 = alloca %clos
%clos_fn_ptr24 = getelementptr inbounds %clos, %clos* %app_lhs23, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @printString to i8*), i8** %clos_fn_ptr
%malloccall25 = tail call i8* @malloc(i32 0)
%env_aloc26 = bitcast i8* %malloccall25 to [0 x i8*]*
%env27 = getelementptr inbounds %clos, %clos* %app_lhs23, i32 0, i32 1
%env_ptr_raw28 = bitcast [0 x i8*]* %env_aloc26 to i8**
store i8** %env_ptr_raw28, i8*** %env27
%app_rhs29 = alloca i64
store i64 0, i64* %app_rhs29
%raw_app_rhs30 = bitcast i64* %app_rhs29 to i8*
%raw_fn_ptr_ptr31 = getelementptr inbounds %clos, %clos* %app_lhs23, i32 0, i3
%raw_fn_ptr32 = load i8*, i8** %raw_fn_ptr_ptr31
%fn_ptr33 = bitcast i8* %raw_fn_ptr32 to %clos* (i8*, i8**)*
%args_ptr34 = getelementptr inbounds %clos, %clos* %app_lhs23, i32 0, i32 1
%args35 = load i8**, i8*** %args_ptr34
%app_res_ptr36 = call %clos* %fn_ptr33(i8* %raw_app_rhs30, i8** %args35)
%app_res37 = load %clos, %clos* %app_res_ptr36
store %clos %app_res37, %clos* %app_lhs22
%app_rhs38 = alloca %adt
%boxed = alloca %boxt
store %boxt %cons_destruct, %boxt* %boxed
%unbox_ptr = bitcast %boxt* %boxed to %adt*
%unbox_val = load %adt, %adt* %unbox_ptr
store %adt %unbox_val, %adt* %app_rhs38
```

```
  %raw_app_rhs39 = bitcast %adt* %app_rhs38 to i8*
  %raw_fn_ptr_ptr40 = getelementptr inbounds %clos, %clos* %app_lhs22, i32 0, i3
  %raw_fn_ptr41 = load i8*, i8** %raw_fn_ptr_ptr40
  %fn_ptr42 = bitcast i8* %raw_fn_ptr41 to %adt* (i8*, i8**)*
  %args_ptr43 = getelementptr inbounds %clos, %clos* %app_lhs22, i32 0, i32 1
  %args44 = load i8**, i8*** %args_ptr43
  %app_res_ptr45 = call %adt* %fn_ptr42(i8* %raw_app_rhs39, i8** %args44)
  %app_res46 = load %adt, %adt* %app_res_ptr45
  store %adt %app_res46, %adt* %app_rhs21
  %raw_app_rhs47 = bitcast %adt* %app_rhs21 to i8*
  %raw_fn_ptr_ptr48 = getelementptr inbounds %clos, %clos* %app_lhs14, i32 0, i3
  %raw_fn_ptr49 = load i8*, i8** %raw_fn_ptr_ptr48
  %fn_ptr50 = bitcast i8* %raw_fn_ptr49 to %adt* (i8*, i8**)*
  %args_ptr51 = getelementptr inbounds %clos, %clos* %app_lhs14, i32 0, i32 1
  %args52 = load i8**, i8*** %args_ptr51
  %app_res_ptr53 = call %adt* %fn_ptr50(i8* %raw_app_rhs47, i8** %args52)
  %app_res54 = load %adt, %adt* %app_res_ptr53
  store %adt %app_res54, %adt* %rval_ptr
  br label %case_continue

case_continue:                                    ; preds = %case, %default
  ret %adt* %rval_ptr
}

define %adt* @printFile3192161587922682755(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %app_lhs = alloca %clos
  %app_lhs1 = alloca %clos
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @closeFile to i8*), i8** %clos_fn_ptr
  %malloccall2 = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall2 to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  %app_rhs = alloca i64
```

```
  store i64 0, i64* %app_rhs
  %raw_app_rhs = bitcast i64* %app_rhs to i8*
  %raw_fn_ptr_ptr = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 0
  %raw_fn_ptr = load i8*, i8** %raw_fn_ptr_ptr
  %fn_ptr = bitcast i8* %raw_fn_ptr to %clos* (i8*, i8**)*
  %args_ptr = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 1
  %args = load i8**, i8*** %args_ptr
  %app_res_ptr = call %clos* %fn_ptr(i8* %raw_app_rhs, i8** %args)
  %app_res = load %clos, %clos* %app_res_ptr
  store %clos %app_res, %clos* %app_lhs
  %app_rhs3 = alloca %adt
  %boxed = alloca %boxt
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg_ptr = bitcast i8* %raw_arg_ptr to %boxt*
  %arg = load %boxt, %boxt* %arg_ptr
  store %boxt %arg, %boxt* %boxed
  %unbox_ptr = bitcast %boxt* %boxed to %adt*
  %unbox_val = load %adt, %adt* %unbox_ptr
  store %adt %unbox_val, %adt* %app_rhs3
  %raw_app_rhs4 = bitcast %adt* %app_rhs3 to i8*
  %raw_fn_ptr_ptr5 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 0
  %raw_fn_ptr6 = load i8*, i8** %raw_fn_ptr_ptr5
  %fn_ptr7 = bitcast i8* %raw_fn_ptr6 to %adt* (i8*, i8**)*
  %args_ptr8 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
  %args9 = load i8**, i8*** %args_ptr8
  %app_res_ptr10 = call %adt* %fn_ptr7(i8* %raw_app_rhs4, i8** %args9)
  %app_res11 = load %adt, %adt* %app_res_ptr10
  store %adt %app_res11, %adt* %rval_ptr
  ret %adt* %rval_ptr
}

define %clos* @C(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
  %rval_ptr = bitcast i8* %malloccall to %clos*
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 0
```

```
  store i8* bitcast (%adt* (i8*, i8**)* @C7985179176134664640 to i8*), i8** %clo
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1
  %env_aloc = bitcast i8* %malloccall1 to [1 x i8*]*
  %raw_closarg_ptr_ptr = getelementptr inbounds [1 x i8*], [1 x i8*]* %env_aloc,
  %closarg_ptr = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
  %arg = load i8, i8* %0
  store i8 %arg, i8* %closarg_ptr
  store i8* %closarg_ptr, i8** %raw_closarg_ptr_ptr
  %env = getelementptr inbounds %clos, %clos* %rval_ptr, i32 0, i32 1
  %env_ptr_raw = bitcast [1 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  ret %clos* %rval_ptr
}

define %adt* @C7985179176134664640(i8* %0, i8** %1) {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt, %
  %rval_ptr = bitcast i8* %malloccall to %adt*
  %carg_aloc = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8* nu
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (%adt* getelementptr (%adt,
  %carg_aloc3 = bitcast i8* %malloccall2 to %adt*
  %arg_gep_ptr = bitcast i8** %1 to [1 x i8*]*
  %raw_arg_ptr_ptr = getelementptr [1 x i8*], [1 x i8*]* %arg_gep_ptr, i32 0, i3
  %raw_arg_ptr = load i8*, i8** %raw_arg_ptr_ptr
  %arg = load i8, i8* %raw_arg_ptr
  store i8 %arg, i8* %carg_aloc
  %arg_ptr = bitcast i8* %0 to %adt*
  %arg4 = load %adt, %adt* %arg_ptr
  store %adt %arg4, %adt* %carg_aloc3
  %malloccall5 = tail call i8* @malloc(i32 ptrtoint (%C* getelementptr (%C, %C*
  %cons = bitcast i8* %malloccall5 to %C*
  %carg_aloc_val = load i8, i8* %carg_aloc
  %carg = getelementptr inbounds %C, %C* %cons, i32 0, i32 0
  store i8 %carg_aloc_val, i8* %carg
  %carg_aloc_val6 = load %adt, %adt* %carg_aloc3
  %carg7 = getelementptr inbounds %C, %C* %cons, i32 0, i32 1
  store %adt %carg_aloc_val6, %adt* %carg7
  %cons_vptr = bitcast %C* %cons to i8*
```

```
  %tag = getelementptr inbounds %adt, %adt* %rval_ptr, i32 0, i32 0
  %data_ptr = getelementptr inbounds %adt, %adt* %rval_ptr, i32 0, i32 1
  store i64 1, i64* %tag
  store i8* %cons_vptr, i8** %data_ptr
  ret %adt* %rval_ptr
}

declare i1 @__prim__unop__not(i1)

declare i64 @__prim__unop__neg(i64)

declare i1 @__prim__binop__or(i1, i1)

declare i1 @__prim__binop__and(i1, i1)

declare i1 @__prim__binop__neq(i64, i64)

declare i1 @__prim__binop__lt(i64, i64)

declare i1 @__prim__binop__leq(i64, i64)

declare i1 @__prim__binop__eq(i64, i64)

declare i1 @__prim__binop__gt(i64, i64)

declare i1 @__prim__binop__geq(i64, i64)

declare i64 @__prim__binop__divide(i64, i64)

declare i64 @__prim__binop__times(i64, i64)

declare i64 @__prim__binop__minus(i64, i64)

declare i64 @__prim__binop__plus(i64, i64)

declare %adt @close_file(%adt)

declare %adt @read_file(%adt)
```

```
declare %adt @open_file(%adt, %adt)

declare %adt @set_bit(%adt, i64, i8)

declare %adt @prim_drop(%adt)

declare %adt @prim_alloc(i64)

declare %adt @print_string(%adt)

declare i64 @print_int(i64)

declare noalias i8* @malloc(i32)

define i64 @main() {
entry:
  %ret = alloca i64
  %boxed = alloca %boxt
  %app_lhs = alloca %clos
  %app_lhs1 = alloca %clos
  %app_lhs2 = alloca %clos
  %clos_fn_ptr = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 0
  store i8* bitcast (%clos* (i8*, i8**)* @const to i8*), i8** %clos_fn_ptr
  %malloccall = tail call i8* @malloc(i32 0)
  %env_aloc = bitcast i8* %malloccall to [0 x i8*]*
  %env = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 1
  %env_ptr_raw = bitcast [0 x i8*]* %env_aloc to i8**
  store i8** %env_ptr_raw, i8*** %env
  %app_rhs = alloca i64
  store i64 0, i64* %app_rhs
  %raw_app_rhs = bitcast i64* %app_rhs to i8*
  %raw_fn_ptr_ptr = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 0
  %raw_fn_ptr = load i8*, i8** %raw_fn_ptr_ptr
  %fn_ptr = bitcast i8* %raw_fn_ptr to %clos* (i8*, i8**)*
  %args_ptr = getelementptr inbounds %clos, %clos* %app_lhs2, i32 0, i32 1
  %args = load i8**, i8*** %args_ptr
  %app_res_ptr = call %clos* %fn_ptr(i8* %raw_app_rhs, i8** %args)
```

```
%app_res = load %clos, %clos* %app_res_ptr
store %clos %app_res, %clos* %app_lhs1
%app_rhs3 = alloca %boxt
%unbox = alloca i64
store i64 0, i64* %unbox
%box_ptr = bitcast i64* %unbox to %boxt*
%box_val = load %boxt, %boxt* %box_ptr
store %boxt %box_val, %boxt* %app_rhs3
%raw_app_rhs4 = bitcast %boxt* %app_rhs3 to i8*
%raw_fn_ptr_ptr5 = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32
%raw_fn_ptr6 = load i8*, i8** %raw_fn_ptr_ptr5
%fn_ptr7 = bitcast i8* %raw_fn_ptr6 to %clos* (i8*, i8**)*
%args_ptr8 = getelementptr inbounds %clos, %clos* %app_lhs1, i32 0, i32 1
%args9 = load i8**, i8*** %args_ptr8
%app_res_ptr10 = call %clos* %fn_ptr7(i8* %raw_app_rhs4, i8** %args9)
%app_res11 = load %clos, %clos* %app_res_ptr10
store %clos %app_res11, %clos* %app_lhs
%app_rhs12 = alloca %boxt
%unbox13 = alloca %adt
%app_lhs14 = alloca %clos
%app_lhs15 = alloca %clos
%app_lhs16 = alloca %clos
%clos_fn_ptr17 = getelementptr inbounds %clos, %clos* %app_lhs16, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @printFile to i8*), i8** %clos_fn_ptr17
%malloccall18 = tail call i8* @malloc(i32 0)
%env_aloc19 = bitcast i8* %malloccall18 to [0 x i8*]*
%env20 = getelementptr inbounds %clos, %clos* %app_lhs16, i32 0, i32 1
%env_ptr_raw21 = bitcast [0 x i8*]* %env_aloc19 to i8**
store i8** %env_ptr_raw21, i8*** %env20
%app_rhs22 = alloca i64
store i64 0, i64* %app_rhs22
%raw_app_rhs23 = bitcast i64* %app_rhs22 to i8*
%raw_fn_ptr_ptr24 = getelementptr inbounds %clos, %clos* %app_lhs16, i32 0, i3
%raw_fn_ptr25 = load i8*, i8** %raw_fn_ptr_ptr24
%fn_ptr26 = bitcast i8* %raw_fn_ptr25 to %clos* (i8*, i8**)*
%args_ptr27 = getelementptr inbounds %clos, %clos* %app_lhs16, i32 0, i32 1
%args28 = load i8**, i8*** %args_ptr27
%app_res_ptr29 = call %clos* %fn_ptr26(i8* %raw_app_rhs23, i8** %args28)
```

```
%app_res30 = load %clos, %clos* %app_res_ptr29
store %clos %app_res30, %clos* %app_lhs15
%app_rhs31 = alloca %adt
%app_lhs32 = alloca %clos
%app_lhs33 = alloca %clos
%clos_fn_ptr34 = getelementptr inbounds %clos, %clos* %app_lhs33, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr34
%malloccall35 = tail call i8* @malloc(i32 0)
%env_aloc36 = bitcast i8* %malloccall35 to [0 x i8*]*
%env37 = getelementptr inbounds %clos, %clos* %app_lhs33, i32 0, i32 1
%env_ptr_raw38 = bitcast [0 x i8*]* %env_aloc36 to i8**
store i8** %env_ptr_raw38, i8*** %env37
%app_rhs39 = alloca i8
store i8 46, i8* %app_rhs39
%raw_fn_ptr_ptr40 = getelementptr inbounds %clos, %clos* %app_lhs33, i32 0, i3
%raw_fn_ptr41 = load i8*, i8** %raw_fn_ptr_ptr40
%fn_ptr42 = bitcast i8* %raw_fn_ptr41 to %clos* (i8*, i8**)*
%args_ptr43 = getelementptr inbounds %clos, %clos* %app_lhs33, i32 0, i32 1
%args44 = load i8**, i8*** %args_ptr43
%app_res_ptr45 = call %clos* %fn_ptr42(i8* %app_rhs39, i8** %args44)
%app_res46 = load %clos, %clos* %app_res_ptr45
store %clos %app_res46, %clos* %app_lhs32
%app_rhs47 = alloca %adt
%app_lhs48 = alloca %clos
%app_lhs49 = alloca %clos
%clos_fn_ptr50 = getelementptr inbounds %clos, %clos* %app_lhs49, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr50
%malloccall51 = tail call i8* @malloc(i32 0)
%env_aloc52 = bitcast i8* %malloccall51 to [0 x i8*]*
%env53 = getelementptr inbounds %clos, %clos* %app_lhs49, i32 0, i32 1
%env_ptr_raw54 = bitcast [0 x i8*]* %env_aloc52 to i8**
store i8** %env_ptr_raw54, i8*** %env53
%app_rhs55 = alloca i8
store i8 47, i8* %app_rhs55
%raw_fn_ptr_ptr56 = getelementptr inbounds %clos, %clos* %app_lhs49, i32 0, i3
%raw_fn_ptr57 = load i8*, i8** %raw_fn_ptr_ptr56
%fn_ptr58 = bitcast i8* %raw_fn_ptr57 to %clos* (i8*, i8**)*
%args_ptr59 = getelementptr inbounds %clos, %clos* %app_lhs49, i32 0, i32 1
```

```
%args60 = load i8**, i8*** %args_ptr59
%app_res_ptr61 = call %clos* %fn_ptr58(i8* %app_rhs55, i8** %args60)
%app_res62 = load %clos, %clos* %app_res_ptr61
store %clos %app_res62, %clos* %app_lhs48
%app_rhs63 = alloca %adt
%app_lhs64 = alloca %clos
%app_lhs65 = alloca %clos
%clos_fn_ptr66 = getelementptr inbounds %clos, %clos* %app_lhs65, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr66
%malloccall67 = tail call i8* @malloc(i32 0)
%env_aloc68 = bitcast i8* %malloccall67 to [0 x i8*]*
%env69 = getelementptr inbounds %clos, %clos* %app_lhs65, i32 0, i32 1
%env_ptr_raw70 = bitcast [0 x i8*]* %env_aloc68 to i8**
store i8** %env_ptr_raw70, i8*** %env69
%app_rhs71 = alloca i8
store i8 114, i8* %app_rhs71
%raw_fn_ptr_ptr72 = getelementptr inbounds %clos, %clos* %app_lhs65, i32 0, i3
%raw_fn_ptr73 = load i8*, i8** %raw_fn_ptr_ptr72
%fn_ptr74 = bitcast i8* %raw_fn_ptr73 to %clos* (i8*, i8**)*
%args_ptr75 = getelementptr inbounds %clos, %clos* %app_lhs65, i32 0, i32 1
%args76 = load i8**, i8*** %args_ptr75
%app_res_ptr77 = call %clos* %fn_ptr74(i8* %app_rhs71, i8** %args76)
%app_res78 = load %clos, %clos* %app_res_ptr77
store %clos %app_res78, %clos* %app_lhs64
%app_rhs79 = alloca %adt
%app_lhs80 = alloca %clos
%app_lhs81 = alloca %clos
%clos_fn_ptr82 = getelementptr inbounds %clos, %clos* %app_lhs81, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr82
%malloccall83 = tail call i8* @malloc(i32 0)
%env_aloc84 = bitcast i8* %malloccall83 to [0 x i8*]*
%env85 = getelementptr inbounds %clos, %clos* %app_lhs81, i32 0, i32 1
%env_ptr_raw86 = bitcast [0 x i8*]* %env_aloc84 to i8**
store i8** %env_ptr_raw86, i8*** %env85
%app_rhs87 = alloca i8
store i8 101, i8* %app_rhs87
%raw_fn_ptr_ptr88 = getelementptr inbounds %clos, %clos* %app_lhs81, i32 0, i3
%raw_fn_ptr89 = load i8*, i8** %raw_fn_ptr_ptr88
```

```
%fn_ptr90 = bitcast i8* %raw_fn_ptr89 to %clos* (i8*, i8**)*
%args_ptr91 = getelementptr inbounds %clos, %clos* %app_lhs81, i32 0, i32 1
%args92 = load i8**, i8*** %args_ptr91
%app_res_ptr93 = call %clos* %fn_ptr90(i8* %app_rhs87, i8** %args92)
%app_res94 = load %clos, %clos* %app_res_ptr93
store %clos %app_res94, %clos* %app_lhs80
%app_rhs95 = alloca %adt
%app_lhs96 = alloca %clos
%app_lhs97 = alloca %clos
%clos_fn_ptr98 = getelementptr inbounds %clos, %clos* %app_lhs97, i32 0, i32 0
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr98
%malloccall99 = tail call i8* @malloc(i32 0)
%env_aloc100 = bitcast i8* %malloccall99 to [0 x i8*]*
%env101 = getelementptr inbounds %clos, %clos* %app_lhs97, i32 0, i32 1
%env_ptr_raw102 = bitcast [0 x i8*]* %env_aloc100 to i8**
store i8** %env_ptr_raw102, i8*** %env101
%app_rhs103 = alloca i8
store i8 103, i8* %app_rhs103
%raw_fn_ptr_ptr104 = getelementptr inbounds %clos, %clos* %app_lhs97, i32 0, i
%raw_fn_ptr105 = load i8*, i8** %raw_fn_ptr_ptr104
%fn_ptr106 = bitcast i8* %raw_fn_ptr105 to %clos* (i8*, i8**)*
%args_ptr107 = getelementptr inbounds %clos, %clos* %app_lhs97, i32 0, i32 1
%args108 = load i8**, i8*** %args_ptr107
%app_res_ptr109 = call %clos* %fn_ptr106(i8* %app_rhs103, i8** %args108)
%app_res110 = load %clos, %clos* %app_res_ptr109
store %clos %app_res110, %clos* %app_lhs96
%app_rhs111 = alloca %adt
%app_lhs112 = alloca %clos
%app_lhs113 = alloca %clos
%clos_fn_ptr114 = getelementptr inbounds %clos, %clos* %app_lhs113, i32 0, i32
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr114
%malloccall115 = tail call i8* @malloc(i32 0)
%env_aloc116 = bitcast i8* %malloccall115 to [0 x i8*]*
%env117 = getelementptr inbounds %clos, %clos* %app_lhs113, i32 0, i32 1
%env_ptr_raw118 = bitcast [0 x i8*]* %env_aloc116 to i8**
store i8** %env_ptr_raw118, i8*** %env117
%app_rhs119 = alloca i8
store i8 45, i8* %app_rhs119
```

```
%raw_fn_ptr_ptr120 = getelementptr inbounds %clos, %clos* %app_lhs113, i32 0,
%raw_fn_ptr121 = load i8*, i8** %raw_fn_ptr_ptr120
%fn_ptr122 = bitcast i8* %raw_fn_ptr121 to %clos* (i8*, i8**)*
%args_ptr123 = getelementptr inbounds %clos, %clos* %app_lhs113, i32 0, i32 1
%args124 = load i8**, i8*** %args_ptr123
%app_res_ptr125 = call %clos* %fn_ptr122(i8* %app_rhs119, i8** %args124)
%app_res126 = load %clos, %clos* %app_res_ptr125
store %clos %app_res126, %clos* %app_lhs112
%app_rhs127 = alloca %adt
%app_lhs128 = alloca %clos
%app_lhs129 = alloca %clos
%clos_fn_ptr130 = getelementptr inbounds %clos, %clos* %app_lhs129, i32 0, i32
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr130
%malloccall131 = tail call i8* @malloc(i32 0)
%env_aloc132 = bitcast i8* %malloccall131 to [0 x i8*]*
%env133 = getelementptr inbounds %clos, %clos* %app_lhs129, i32 0, i32 1
%env_ptr_raw134 = bitcast [0 x i8*]* %env_aloc132 to i8**
store i8** %env_ptr_raw134, i8*** %env133
%app_rhs135 = alloca i8
store i8 116, i8* %app_rhs135
%raw_fn_ptr_ptr136 = getelementptr inbounds %clos, %clos* %app_lhs129, i32 0,
%raw_fn_ptr137 = load i8*, i8** %raw_fn_ptr_ptr136
%fn_ptr138 = bitcast i8* %raw_fn_ptr137 to %clos* (i8*, i8**)*
%args_ptr139 = getelementptr inbounds %clos, %clos* %app_lhs129, i32 0, i32 1
%args140 = load i8**, i8*** %args_ptr139
%app_res_ptr141 = call %clos* %fn_ptr138(i8* %app_rhs135, i8** %args140)
%app_res142 = load %clos, %clos* %app_res_ptr141
store %clos %app_res142, %clos* %app_lhs128
%app_rhs143 = alloca %adt
%app_lhs144 = alloca %clos
%app_lhs145 = alloca %clos
%clos_fn_ptr146 = getelementptr inbounds %clos, %clos* %app_lhs145, i32 0, i32
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr146
%malloccall147 = tail call i8* @malloc(i32 0)
%env_aloc148 = bitcast i8* %malloccall147 to [0 x i8*]*
%env149 = getelementptr inbounds %clos, %clos* %app_lhs145, i32 0, i32 1
%env_ptr_raw150 = bitcast [0 x i8*]* %env_aloc148 to i8**
store i8** %env_ptr_raw150, i8*** %env149
```

```
%app_rhs151 = alloca i8
store i8 101, i8* %app_rhs151
%raw_fn_ptr_ptr152 = getelementptr inbounds %clos, %clos* %app_lhs145, i32 0,
%raw_fn_ptr153 = load i8*, i8** %raw_fn_ptr_ptr152
%fn_ptr154 = bitcast i8* %raw_fn_ptr153 to %clos* (i8*, i8**)*
%args_ptr155 = getelementptr inbounds %clos, %clos* %app_lhs145, i32 0, i32 1
%args156 = load i8**, i8*** %args_ptr155
%app_res_ptr157 = call %clos* %fn_ptr154(i8* %app_rhs151, i8** %args156)
%app_res158 = load %clos, %clos* %app_res_ptr157
store %clos %app_res158, %clos* %app_lhs144
%app_rhs159 = alloca %adt
%app_lhs160 = alloca %clos
%app_lhs161 = alloca %clos
%clos_fn_ptr162 = getelementptr inbounds %clos, %clos* %app_lhs161, i32 0, i32
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr162
%malloccall163 = tail call i8* @malloc(i32 0)
%env_aloc164 = bitcast i8* %malloccall163 to [0 x i8*]*
%env165 = getelementptr inbounds %clos, %clos* %app_lhs161, i32 0, i32 1
%env_ptr_raw166 = bitcast [0 x i8*]* %env_aloc164 to i8**
store i8** %env_ptr_raw166, i8*** %env165
%app_rhs167 = alloca i8
store i8 115, i8* %app_rhs167
%raw_fn_ptr_ptr168 = getelementptr inbounds %clos, %clos* %app_lhs161, i32 0,
%raw_fn_ptr169 = load i8*, i8** %raw_fn_ptr_ptr168
%fn_ptr170 = bitcast i8* %raw_fn_ptr169 to %clos* (i8*, i8**)*
%args_ptr171 = getelementptr inbounds %clos, %clos* %app_lhs161, i32 0, i32 1
%args172 = load i8**, i8*** %args_ptr171
%app_res_ptr173 = call %clos* %fn_ptr170(i8* %app_rhs167, i8** %args172)
%app_res174 = load %clos, %clos* %app_res_ptr173
store %clos %app_res174, %clos* %app_lhs160
%app_rhs175 = alloca %adt
%app_lhs176 = alloca %clos
%app_lhs177 = alloca %clos
%clos_fn_ptr178 = getelementptr inbounds %clos, %clos* %app_lhs177, i32 0, i32
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr178
%malloccall179 = tail call i8* @malloc(i32 0)
%env_aloc180 = bitcast i8* %malloccall179 to [0 x i8*]*
%env181 = getelementptr inbounds %clos, %clos* %app_lhs177, i32 0, i32 1
```

```
%env_ptr_raw182 = bitcast [0 x i8*]* %env_aloc180 to i8**
store i8** %env_ptr_raw182, i8*** %env181
%app_rhs183 = alloca i8
store i8 116, i8* %app_rhs183
%raw_fn_ptr_ptr184 = getelementptr inbounds %clos, %clos* %app_lhs177, i32 0,
%raw_fn_ptr185 = load i8*, i8** %raw_fn_ptr_ptr184
%fn_ptr186 = bitcast i8* %raw_fn_ptr185 to %clos* (i8*, i8**)*
%args_ptr187 = getelementptr inbounds %clos, %clos* %app_lhs177, i32 0, i32 1
%args188 = load i8**, i8*** %args_ptr187
%app_res_ptr189 = call %clos* %fn_ptr186(i8* %app_rhs183, i8** %args188)
%app_res190 = load %clos, %clos* %app_res_ptr189
store %clos %app_res190, %clos* %app_lhs176
%app_rhs191 = alloca %adt
%app_lhs192 = alloca %clos
%app_lhs193 = alloca %clos
%clos_fn_ptr194 = getelementptr inbounds %clos, %clos* %app_lhs193, i32 0, i32
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr194
%malloccall195 = tail call i8* @malloc(i32 0)
%env_aloc196 = bitcast i8* %malloccall195 to [0 x i8*]*
%env197 = getelementptr inbounds %clos, %clos* %app_lhs193, i32 0, i32 1
%env_ptr_raw198 = bitcast [0 x i8*]* %env_aloc196 to i8**
store i8** %env_ptr_raw198, i8*** %env197
%app_rhs199 = alloca i8
store i8 115, i8* %app_rhs199
%raw_fn_ptr_ptr200 = getelementptr inbounds %clos, %clos* %app_lhs193, i32 0,
%raw_fn_ptr201 = load i8*, i8** %raw_fn_ptr_ptr200
%fn_ptr202 = bitcast i8* %raw_fn_ptr201 to %clos* (i8*, i8**)*
%args_ptr203 = getelementptr inbounds %clos, %clos* %app_lhs193, i32 0, i32 1
%args204 = load i8**, i8*** %args_ptr203
%app_res_ptr205 = call %clos* %fn_ptr202(i8* %app_rhs199, i8** %args204)
%app_res206 = load %clos, %clos* %app_res_ptr205
store %clos %app_res206, %clos* %app_lhs192
%app_rhs207 = alloca %adt
%app_lhs208 = alloca %clos
%app_lhs209 = alloca %clos
%clos_fn_ptr210 = getelementptr inbounds %clos, %clos* %app_lhs209, i32 0, i32
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr210
%malloccall211 = tail call i8* @malloc(i32 0)
```

```
%env_aloc212 = bitcast i8* %malloccall211 to [0 x i8*]*
%env213 = getelementptr inbounds %clos, %clos* %app_lhs209, i32 0, i32 1
%env_ptr_raw214 = bitcast [0 x i8*]* %env_aloc212 to i8**
store i8** %env_ptr_raw214, i8*** %env213
%app_rhs215 = alloca i8
store i8 47, i8* %app_rhs215
%raw_fn_ptr_ptr216 = getelementptr inbounds %clos, %clos* %app_lhs209, i32 0,
%raw_fn_ptr217 = load i8*, i8** %raw_fn_ptr_ptr216
%fn_ptr218 = bitcast i8* %raw_fn_ptr217 to %clos* (i8*, i8**)*
%args_ptr219 = getelementptr inbounds %clos, %clos* %app_lhs209, i32 0, i32 1
%args220 = load i8**, i8*** %args_ptr219
%app_res_ptr221 = call %clos* %fn_ptr218(i8* %app_rhs215, i8** %args220)
%app_res222 = load %clos, %clos* %app_res_ptr221
store %clos %app_res222, %clos* %app_lhs208
%app_rhs223 = alloca %adt
%app_lhs224 = alloca %clos
%app_lhs225 = alloca %clos
%clos_fn_ptr226 = getelementptr inbounds %clos, %clos* %app_lhs225, i32 0, i32
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr226
%malloccall227 = tail call i8* @malloc(i32 0)
%env_aloc228 = bitcast i8* %malloccall227 to [0 x i8*]*
%env229 = getelementptr inbounds %clos, %clos* %app_lhs225, i32 0, i32 1
%env_ptr_raw230 = bitcast [0 x i8*]* %env_aloc228 to i8**
store i8** %env_ptr_raw230, i8*** %env229
%app_rhs231 = alloca i8
store i8 102, i8* %app_rhs231
%raw_fn_ptr_ptr232 = getelementptr inbounds %clos, %clos* %app_lhs225, i32 0,
%raw_fn_ptr233 = load i8*, i8** %raw_fn_ptr_ptr232
%fn_ptr234 = bitcast i8* %raw_fn_ptr233 to %clos* (i8*, i8**)*
%args_ptr235 = getelementptr inbounds %clos, %clos* %app_lhs225, i32 0, i32 1
%args236 = load i8**, i8*** %args_ptr235
%app_res_ptr237 = call %clos* %fn_ptr234(i8* %app_rhs231, i8** %args236)
%app_res238 = load %clos, %clos* %app_res_ptr237
store %clos %app_res238, %clos* %app_lhs224
%app_rhs239 = alloca %adt
%app_lhs240 = alloca %clos
%app_lhs241 = alloca %clos
%clos_fn_ptr242 = getelementptr inbounds %clos, %clos* %app_lhs241, i32 0, i32
```

```
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr242
%malloccall243 = tail call i8* @malloc(i32 0)
%env_aloc244 = bitcast i8* %malloccall243 to [0 x i8*]*
%env245 = getelementptr inbounds %clos, %clos* %app_lhs241, i32 0, i32 1
%env_ptr_raw246 = bitcast [0 x i8*]* %env_aloc244 to i8**
store i8** %env_ptr_raw246, i8*** %env245
%app_rhs247 = alloca i8
store i8 105, i8* %app_rhs247
%raw_fn_ptr_ptr248 = getelementptr inbounds %clos, %clos* %app_lhs241, i32 0,
%raw_fn_ptr249 = load i8*, i8** %raw_fn_ptr_ptr248
%fn_ptr250 = bitcast i8* %raw_fn_ptr249 to %clos* (i8*, i8**)*
%args_ptr251 = getelementptr inbounds %clos, %clos* %app_lhs241, i32 0, i32 1
%args252 = load i8**, i8*** %args_ptr251
%app_res_ptr253 = call %clos* %fn_ptr250(i8* %app_rhs247, i8** %args252)
%app_res254 = load %clos, %clos* %app_res_ptr253
store %clos %app_res254, %clos* %app_lhs240
%app_rhs255 = alloca %adt
%app_lhs256 = alloca %clos
%app_lhs257 = alloca %clos
%clos_fn_ptr258 = getelementptr inbounds %clos, %clos* %app_lhs257, i32 0, i32
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr258
%malloccall259 = tail call i8* @malloc(i32 0)
%env_aloc260 = bitcast i8* %malloccall259 to [0 x i8*]*
%env261 = getelementptr inbounds %clos, %clos* %app_lhs257, i32 0, i32 1
%env_ptr_raw262 = bitcast [0 x i8*]* %env_aloc260 to i8**
store i8** %env_ptr_raw262, i8*** %env261
%app_rhs263 = alloca i8
store i8 108, i8* %app_rhs263
%raw_fn_ptr_ptr264 = getelementptr inbounds %clos, %clos* %app_lhs257, i32 0,
%raw_fn_ptr265 = load i8*, i8** %raw_fn_ptr_ptr264
%fn_ptr266 = bitcast i8* %raw_fn_ptr265 to %clos* (i8*, i8**)*
%args_ptr267 = getelementptr inbounds %clos, %clos* %app_lhs257, i32 0, i32 1
%args268 = load i8**, i8*** %args_ptr267
%app_res_ptr269 = call %clos* %fn_ptr266(i8* %app_rhs263, i8** %args268)
%app_res270 = load %clos, %clos* %app_res_ptr269
store %clos %app_res270, %clos* %app_lhs256
%app_rhs271 = alloca %adt
%app_lhs272 = alloca %clos
```

```
%app_lhs273 = alloca %clos
%clos_fn_ptr274 = getelementptr inbounds %clos, %clos* %app_lhs273, i32 0, i32
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr274
%malloccall275 = tail call i8* @malloc(i32 0)
%env_aloc276 = bitcast i8* %malloccall275 to [0 x i8*]*
%env277 = getelementptr inbounds %clos, %clos* %app_lhs273, i32 0, i32 1
%env_ptr_raw278 = bitcast [0 x i8*]* %env_aloc276 to i8**
store i8** %env_ptr_raw278, i8*** %env277
%app_rhs279 = alloca i8
store i8 101, i8* %app_rhs279
%raw_fn_ptr_ptr280 = getelementptr inbounds %clos, %clos* %app_lhs273, i32 0,
%raw_fn_ptr281 = load i8*, i8** %raw_fn_ptr_ptr280
%fn_ptr282 = bitcast i8* %raw_fn_ptr281 to %clos* (i8*, i8**)*
%args_ptr283 = getelementptr inbounds %clos, %clos* %app_lhs273, i32 0, i32 1
%args284 = load i8**, i8*** %args_ptr283
%app_res_ptr285 = call %clos* %fn_ptr282(i8* %app_rhs279, i8** %args284)
%app_res286 = load %clos, %clos* %app_res_ptr285
store %clos %app_res286, %clos* %app_lhs272
%app_rhs287 = alloca %adt
%app_lhs288 = alloca %clos
%app_lhs289 = alloca %clos
%clos_fn_ptr290 = getelementptr inbounds %clos, %clos* %app_lhs289, i32 0, i32
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr290
%malloccall291 = tail call i8* @malloc(i32 0)
%env_aloc292 = bitcast i8* %malloccall291 to [0 x i8*]*
%env293 = getelementptr inbounds %clos, %clos* %app_lhs289, i32 0, i32 1
%env_ptr_raw294 = bitcast [0 x i8*]* %env_aloc292 to i8**
store i8** %env_ptr_raw294, i8*** %env293
%app_rhs295 = alloca i8
store i8 46, i8* %app_rhs295
%raw_fn_ptr_ptr296 = getelementptr inbounds %clos, %clos* %app_lhs289, i32 0,
%raw_fn_ptr297 = load i8*, i8** %raw_fn_ptr_ptr296
%fn_ptr298 = bitcast i8* %raw_fn_ptr297 to %clos* (i8*, i8**)*
%args_ptr299 = getelementptr inbounds %clos, %clos* %app_lhs289, i32 0, i32 1
%args300 = load i8**, i8*** %args_ptr299
%app_res_ptr301 = call %clos* %fn_ptr298(i8* %app_rhs295, i8** %args300)
%app_res302 = load %clos, %clos* %app_res_ptr301
store %clos %app_res302, %clos* %app_lhs288
```

```
%app_rhs303 = alloca %adt
%app_lhs304 = alloca %clos
%app_lhs305 = alloca %clos
%clos_fn_ptr306 = getelementptr inbounds %clos, %clos* %app_lhs305, i32 0, i32
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr306
%malloccall307 = tail call i8* @malloc(i32 0)
%env_aloc308 = bitcast i8* %malloccall307 to [0 x i8*]*
%env309 = getelementptr inbounds %clos, %clos* %app_lhs305, i32 0, i32 1
%env_ptr_raw310 = bitcast [0 x i8*]* %env_aloc308 to i8**
store i8** %env_ptr_raw310, i8*** %env309
%app_rhs311 = alloca i8
store i8 116, i8* %app_rhs311
%raw_fn_ptr_ptr312 = getelementptr inbounds %clos, %clos* %app_lhs305, i32 0,
%raw_fn_ptr313 = load i8*, i8** %raw_fn_ptr_ptr312
%fn_ptr314 = bitcast i8* %raw_fn_ptr313 to %clos* (i8*, i8**)*
%args_ptr315 = getelementptr inbounds %clos, %clos* %app_lhs305, i32 0, i32 1
%args316 = load i8**, i8*** %args_ptr315
%app_res_ptr317 = call %clos* %fn_ptr314(i8* %app_rhs311, i8** %args316)
%app_res318 = load %clos, %clos* %app_res_ptr317
store %clos %app_res318, %clos* %app_lhs304
%app_rhs319 = alloca %adt
%app_lhs320 = alloca %clos
%app_lhs321 = alloca %clos
%clos_fn_ptr322 = getelementptr inbounds %clos, %clos* %app_lhs321, i32 0, i32
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr322
%malloccall323 = tail call i8* @malloc(i32 0)
%env_aloc324 = bitcast i8* %malloccall323 to [0 x i8*]*
%env325 = getelementptr inbounds %clos, %clos* %app_lhs321, i32 0, i32 1
%env_ptr_raw326 = bitcast [0 x i8*]* %env_aloc324 to i8**
store i8** %env_ptr_raw326, i8*** %env325
%app_rhs327 = alloca i8
store i8 120, i8* %app_rhs327
%raw_fn_ptr_ptr328 = getelementptr inbounds %clos, %clos* %app_lhs321, i32 0,
%raw_fn_ptr329 = load i8*, i8** %raw_fn_ptr_ptr328
%fn_ptr330 = bitcast i8* %raw_fn_ptr329 to %clos* (i8*, i8**)*
%args_ptr331 = getelementptr inbounds %clos, %clos* %app_lhs321, i32 0, i32 1
%args332 = load i8**, i8*** %args_ptr331
%app_res_ptr333 = call %clos* %fn_ptr330(i8* %app_rhs327, i8** %args332)
```

```
%app_res334 = load %clos, %clos* %app_res_ptr333
store %clos %app_res334, %clos* %app_lhs320
%app_rhs335 = alloca %adt
%app_lhs336 = alloca %clos
%app_lhs337 = alloca %clos
%clos_fn_ptr338 = getelementptr inbounds %clos, %clos* %app_lhs337, i32 0, i32
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr338
%malloccall339 = tail call i8* @malloc(i32 0)
%env_aloc340 = bitcast i8* %malloccall339 to [0 x i8*]*
%env341 = getelementptr inbounds %clos, %clos* %app_lhs337, i32 0, i32 1
%env_ptr_raw342 = bitcast [0 x i8*]* %env_aloc340 to i8**
store i8** %env_ptr_raw342, i8*** %env341
%app_rhs343 = alloca i8
store i8 116, i8* %app_rhs343
%raw_fn_ptr_ptr344 = getelementptr inbounds %clos, %clos* %app_lhs337, i32 0,
%raw_fn_ptr345 = load i8*, i8** %raw_fn_ptr_ptr344
%fn_ptr346 = bitcast i8* %raw_fn_ptr345 to %clos* (i8*, i8**)*
%args_ptr347 = getelementptr inbounds %clos, %clos* %app_lhs337, i32 0, i32 1
%args348 = load i8**, i8*** %args_ptr347
%app_res_ptr349 = call %clos* %fn_ptr346(i8* %app_rhs343, i8** %args348)
%app_res350 = load %clos, %clos* %app_res_ptr349
store %clos %app_res350, %clos* %app_lhs336
%app_rhs351 = alloca %adt
%malloccall352 = tail call i8* @malloc(i32 0)
%cons = bitcast i8* %malloccall352 to %E*
%cons_vptr = bitcast %E* %cons to i8*
%tag = getelementptr inbounds %adt, %adt* %app_rhs351, i32 0, i32 0
%data_ptr = getelementptr inbounds %adt, %adt* %app_rhs351, i32 0, i32 1
store i64 0, i64* %tag
store i8* %cons_vptr, i8** %data_ptr
%raw_app_rhs353 = bitcast %adt* %app_rhs351 to i8*
%raw_fn_ptr_ptr354 = getelementptr inbounds %clos, %clos* %app_lhs336, i32 0,
%raw_fn_ptr355 = load i8*, i8** %raw_fn_ptr_ptr354
%fn_ptr356 = bitcast i8* %raw_fn_ptr355 to %adt* (i8*, i8**)*
%args_ptr357 = getelementptr inbounds %clos, %clos* %app_lhs336, i32 0, i32 1
%args358 = load i8**, i8*** %args_ptr357
%app_res_ptr359 = call %adt* %fn_ptr356(i8* %raw_app_rhs353, i8** %args358)
%app_res360 = load %adt, %adt* %app_res_ptr359
```

```
store %adt %app_res360, %adt* %app_rhs335
%raw_app_rhs361 = bitcast %adt* %app_rhs335 to i8*
%raw_fn_ptr_ptr362 = getelementptr inbounds %clos, %clos* %app_lhs320, i32 0,
%raw_fn_ptr363 = load i8*, i8** %raw_fn_ptr_ptr362
%fn_ptr364 = bitcast i8* %raw_fn_ptr363 to %adt* (i8*, i8**)*
%args_ptr365 = getelementptr inbounds %clos, %clos* %app_lhs320, i32 0, i32 1
%args366 = load i8**, i8*** %args_ptr365
%app_res_ptr367 = call %adt* %fn_ptr364(i8* %raw_app_rhs361, i8** %args366)
%app_res368 = load %adt, %adt* %app_res_ptr367
store %adt %app_res368, %adt* %app_rhs319
%raw_app_rhs369 = bitcast %adt* %app_rhs319 to i8*
%raw_fn_ptr_ptr370 = getelementptr inbounds %clos, %clos* %app_lhs304, i32 0,
%raw_fn_ptr371 = load i8*, i8** %raw_fn_ptr_ptr370
%fn_ptr372 = bitcast i8* %raw_fn_ptr371 to %adt* (i8*, i8**)*
%args_ptr373 = getelementptr inbounds %clos, %clos* %app_lhs304, i32 0, i32 1
%args374 = load i8**, i8*** %args_ptr373
%app_res_ptr375 = call %adt* %fn_ptr372(i8* %raw_app_rhs369, i8** %args374)
%app_res376 = load %adt, %adt* %app_res_ptr375
store %adt %app_res376, %adt* %app_rhs303
%raw_app_rhs377 = bitcast %adt* %app_rhs303 to i8*
%raw_fn_ptr_ptr378 = getelementptr inbounds %clos, %clos* %app_lhs288, i32 0,
%raw_fn_ptr379 = load i8*, i8** %raw_fn_ptr_ptr378
%fn_ptr380 = bitcast i8* %raw_fn_ptr379 to %adt* (i8*, i8**)*
%args_ptr381 = getelementptr inbounds %clos, %clos* %app_lhs288, i32 0, i32 1
%args382 = load i8**, i8*** %args_ptr381
%app_res_ptr383 = call %adt* %fn_ptr380(i8* %raw_app_rhs377, i8** %args382)
%app_res384 = load %adt, %adt* %app_res_ptr383
store %adt %app_res384, %adt* %app_rhs287
%raw_app_rhs385 = bitcast %adt* %app_rhs287 to i8*
%raw_fn_ptr_ptr386 = getelementptr inbounds %clos, %clos* %app_lhs272, i32 0,
%raw_fn_ptr387 = load i8*, i8** %raw_fn_ptr_ptr386
%fn_ptr388 = bitcast i8* %raw_fn_ptr387 to %adt* (i8*, i8**)*
%args_ptr389 = getelementptr inbounds %clos, %clos* %app_lhs272, i32 0, i32 1
%args390 = load i8**, i8*** %args_ptr389
%app_res_ptr391 = call %adt* %fn_ptr388(i8* %raw_app_rhs385, i8** %args390)
%app_res392 = load %adt, %adt* %app_res_ptr391
store %adt %app_res392, %adt* %app_rhs271
%raw_app_rhs393 = bitcast %adt* %app_rhs271 to i8*
```

```
%raw_fn_ptr_ptr394 = getelementptr inbounds %clos, %clos* %app_lhs256, i32 0,
%raw_fn_ptr395 = load i8*, i8** %raw_fn_ptr_ptr394
%fn_ptr396 = bitcast i8* %raw_fn_ptr395 to %adt* (i8*, i8**)*
%args_ptr397 = getelementptr inbounds %clos, %clos* %app_lhs256, i32 0, i32 1
%args398 = load i8**, i8*** %args_ptr397
%app_res_ptr399 = call %adt* %fn_ptr396(i8* %raw_app_rhs393, i8** %args398)
%app_res400 = load %adt, %adt* %app_res_ptr399
store %adt %app_res400, %adt* %app_rhs255
%raw_app_rhs401 = bitcast %adt* %app_rhs255 to i8*
%raw_fn_ptr_ptr402 = getelementptr inbounds %clos, %clos* %app_lhs240, i32 0,
%raw_fn_ptr403 = load i8*, i8** %raw_fn_ptr_ptr402
%fn_ptr404 = bitcast i8* %raw_fn_ptr403 to %adt* (i8*, i8**)*
%args_ptr405 = getelementptr inbounds %clos, %clos* %app_lhs240, i32 0, i32 1
%args406 = load i8**, i8*** %args_ptr405
%app_res_ptr407 = call %adt* %fn_ptr404(i8* %raw_app_rhs401, i8** %args406)
%app_res408 = load %adt, %adt* %app_res_ptr407
store %adt %app_res408, %adt* %app_rhs239
%raw_app_rhs409 = bitcast %adt* %app_rhs239 to i8*
%raw_fn_ptr_ptr410 = getelementptr inbounds %clos, %clos* %app_lhs224, i32 0,
%raw_fn_ptr411 = load i8*, i8** %raw_fn_ptr_ptr410
%fn_ptr412 = bitcast i8* %raw_fn_ptr411 to %adt* (i8*, i8**)*
%args_ptr413 = getelementptr inbounds %clos, %clos* %app_lhs224, i32 0, i32 1
%args414 = load i8**, i8*** %args_ptr413
%app_res_ptr415 = call %adt* %fn_ptr412(i8* %raw_app_rhs409, i8** %args414)
%app_res416 = load %adt, %adt* %app_res_ptr415
store %adt %app_res416, %adt* %app_rhs223
%raw_app_rhs417 = bitcast %adt* %app_rhs223 to i8*
%raw_fn_ptr_ptr418 = getelementptr inbounds %clos, %clos* %app_lhs208, i32 0,
%raw_fn_ptr419 = load i8*, i8** %raw_fn_ptr_ptr418
%fn_ptr420 = bitcast i8* %raw_fn_ptr419 to %adt* (i8*, i8**)*
%args_ptr421 = getelementptr inbounds %clos, %clos* %app_lhs208, i32 0, i32 1
%args422 = load i8**, i8*** %args_ptr421
%app_res_ptr423 = call %adt* %fn_ptr420(i8* %raw_app_rhs417, i8** %args422)
%app_res424 = load %adt, %adt* %app_res_ptr423
store %adt %app_res424, %adt* %app_rhs207
%raw_app_rhs425 = bitcast %adt* %app_rhs207 to i8*
%raw_fn_ptr_ptr426 = getelementptr inbounds %clos, %clos* %app_lhs192, i32 0,
%raw_fn_ptr427 = load i8*, i8** %raw_fn_ptr_ptr426
```

```
%fn_ptr428 = bitcast i8* %raw_fn_ptr427 to %adt* (i8*, i8**)*
%args_ptr429 = getelementptr inbounds %clos, %clos* %app_lhs192, i32 0, i32 1
%args430 = load i8**, i8*** %args_ptr429
%app_res_ptr431 = call %adt* %fn_ptr428(i8* %raw_app_rhs425, i8** %args430)
%app_res432 = load %adt, %adt* %app_res_ptr431
store %adt %app_res432, %adt* %app_rhs191
%raw_app_rhs433 = bitcast %adt* %app_rhs191 to i8*
%raw_fn_ptr_ptr434 = getelementptr inbounds %clos, %clos* %app_lhs176, i32 0,
%raw_fn_ptr435 = load i8*, i8** %raw_fn_ptr_ptr434
%fn_ptr436 = bitcast i8* %raw_fn_ptr435 to %adt* (i8*, i8**)*
%args_ptr437 = getelementptr inbounds %clos, %clos* %app_lhs176, i32 0, i32 1
%args438 = load i8**, i8*** %args_ptr437
%app_res_ptr439 = call %adt* %fn_ptr436(i8* %raw_app_rhs433, i8** %args438)
%app_res440 = load %adt, %adt* %app_res_ptr439
store %adt %app_res440, %adt* %app_rhs175
%raw_app_rhs441 = bitcast %adt* %app_rhs175 to i8*
%raw_fn_ptr_ptr442 = getelementptr inbounds %clos, %clos* %app_lhs160, i32 0,
%raw_fn_ptr443 = load i8*, i8** %raw_fn_ptr_ptr442
%fn_ptr444 = bitcast i8* %raw_fn_ptr443 to %adt* (i8*, i8**)*
%args_ptr445 = getelementptr inbounds %clos, %clos* %app_lhs160, i32 0, i32 1
%args446 = load i8**, i8*** %args_ptr445
%app_res_ptr447 = call %adt* %fn_ptr444(i8* %raw_app_rhs441, i8** %args446)
%app_res448 = load %adt, %adt* %app_res_ptr447
store %adt %app_res448, %adt* %app_rhs159
%raw_app_rhs449 = bitcast %adt* %app_rhs159 to i8*
%raw_fn_ptr_ptr450 = getelementptr inbounds %clos, %clos* %app_lhs144, i32 0,
%raw_fn_ptr451 = load i8*, i8** %raw_fn_ptr_ptr450
%fn_ptr452 = bitcast i8* %raw_fn_ptr451 to %adt* (i8*, i8**)*
%args_ptr453 = getelementptr inbounds %clos, %clos* %app_lhs144, i32 0, i32 1
%args454 = load i8**, i8*** %args_ptr453
%app_res_ptr455 = call %adt* %fn_ptr452(i8* %raw_app_rhs449, i8** %args454)
%app_res456 = load %adt, %adt* %app_res_ptr455
store %adt %app_res456, %adt* %app_rhs143
%raw_app_rhs457 = bitcast %adt* %app_rhs143 to i8*
%raw_fn_ptr_ptr458 = getelementptr inbounds %clos, %clos* %app_lhs128, i32 0,
%raw_fn_ptr459 = load i8*, i8** %raw_fn_ptr_ptr458
%fn_ptr460 = bitcast i8* %raw_fn_ptr459 to %adt* (i8*, i8**)*
%args_ptr461 = getelementptr inbounds %clos, %clos* %app_lhs128, i32 0, i32 1
```

```
%args462 = load i8**, i8*** %args_ptr461
%app_res_ptr463 = call %adt* %fn_ptr460(i8* %raw_app_rhs457, i8** %args462)
%app_res464 = load %adt, %adt* %app_res_ptr463
store %adt %app_res464, %adt* %app_rhs127
%raw_app_rhs465 = bitcast %adt* %app_rhs127 to i8*
%raw_fn_ptr_ptr466 = getelementptr inbounds %clos, %clos* %app_lhs112, i32 0,
%raw_fn_ptr467 = load i8*, i8** %raw_fn_ptr_ptr466
%fn_ptr468 = bitcast i8* %raw_fn_ptr467 to %adt* (i8*, i8**)*
%args_ptr469 = getelementptr inbounds %clos, %clos* %app_lhs112, i32 0, i32 1
%args470 = load i8**, i8*** %args_ptr469
%app_res_ptr471 = call %adt* %fn_ptr468(i8* %raw_app_rhs465, i8** %args470)
%app_res472 = load %adt, %adt* %app_res_ptr471
store %adt %app_res472, %adt* %app_rhs111
%raw_app_rhs473 = bitcast %adt* %app_rhs111 to i8*
%raw_fn_ptr_ptr474 = getelementptr inbounds %clos, %clos* %app_lhs96, i32 0, i
%raw_fn_ptr475 = load i8*, i8** %raw_fn_ptr_ptr474
%fn_ptr476 = bitcast i8* %raw_fn_ptr475 to %adt* (i8*, i8**)*
%args_ptr477 = getelementptr inbounds %clos, %clos* %app_lhs96, i32 0, i32 1
%args478 = load i8**, i8*** %args_ptr477
%app_res_ptr479 = call %adt* %fn_ptr476(i8* %raw_app_rhs473, i8** %args478)
%app_res480 = load %adt, %adt* %app_res_ptr479
store %adt %app_res480, %adt* %app_rhs95
%raw_app_rhs481 = bitcast %adt* %app_rhs95 to i8*
%raw_fn_ptr_ptr482 = getelementptr inbounds %clos, %clos* %app_lhs80, i32 0, i
%raw_fn_ptr483 = load i8*, i8** %raw_fn_ptr_ptr482
%fn_ptr484 = bitcast i8* %raw_fn_ptr483 to %adt* (i8*, i8**)*
%args_ptr485 = getelementptr inbounds %clos, %clos* %app_lhs80, i32 0, i32 1
%args486 = load i8**, i8*** %args_ptr485
%app_res_ptr487 = call %adt* %fn_ptr484(i8* %raw_app_rhs481, i8** %args486)
%app_res488 = load %adt, %adt* %app_res_ptr487
store %adt %app_res488, %adt* %app_rhs79
%raw_app_rhs489 = bitcast %adt* %app_rhs79 to i8*
%raw_fn_ptr_ptr490 = getelementptr inbounds %clos, %clos* %app_lhs64, i32 0, i
%raw_fn_ptr491 = load i8*, i8** %raw_fn_ptr_ptr490
%fn_ptr492 = bitcast i8* %raw_fn_ptr491 to %adt* (i8*, i8**)*
%args_ptr493 = getelementptr inbounds %clos, %clos* %app_lhs64, i32 0, i32 1
%args494 = load i8**, i8*** %args_ptr493
%app_res_ptr495 = call %adt* %fn_ptr492(i8* %raw_app_rhs489, i8** %args494)
```

```
%app_res496 = load %adt, %adt* %app_res_ptr495
store %adt %app_res496, %adt* %app_rhs63
%raw_app_rhs497 = bitcast %adt* %app_rhs63 to i8*
%raw_fn_ptr_ptr498 = getelementptr inbounds %clos, %clos* %app_lhs48, i32 0, i
%raw_fn_ptr499 = load i8*, i8** %raw_fn_ptr_ptr498
%fn_ptr500 = bitcast i8* %raw_fn_ptr499 to %adt* (i8*, i8**)*
%args_ptr501 = getelementptr inbounds %clos, %clos* %app_lhs48, i32 0, i32 1
%args502 = load i8**, i8*** %args_ptr501
%app_res_ptr503 = call %adt* %fn_ptr500(i8* %raw_app_rhs497, i8** %args502)
%app_res504 = load %adt, %adt* %app_res_ptr503
store %adt %app_res504, %adt* %app_rhs47
%raw_app_rhs505 = bitcast %adt* %app_rhs47 to i8*
%raw_fn_ptr_ptr506 = getelementptr inbounds %clos, %clos* %app_lhs32, i32 0, i
%raw_fn_ptr507 = load i8*, i8** %raw_fn_ptr_ptr506
%fn_ptr508 = bitcast i8* %raw_fn_ptr507 to %adt* (i8*, i8**)*
%args_ptr509 = getelementptr inbounds %clos, %clos* %app_lhs32, i32 0, i32 1
%args510 = load i8**, i8*** %args_ptr509
%app_res_ptr511 = call %adt* %fn_ptr508(i8* %raw_app_rhs505, i8** %args510)
%app_res512 = load %adt, %adt* %app_res_ptr511
store %adt %app_res512, %adt* %app_rhs31
%raw_app_rhs513 = bitcast %adt* %app_rhs31 to i8*
%raw_fn_ptr_ptr514 = getelementptr inbounds %clos, %clos* %app_lhs15, i32 0, i
%raw_fn_ptr515 = load i8*, i8** %raw_fn_ptr_ptr514
%fn_ptr516 = bitcast i8* %raw_fn_ptr515 to %clos* (i8*, i8**)*
%args_ptr517 = getelementptr inbounds %clos, %clos* %app_lhs15, i32 0, i32 1
%args518 = load i8**, i8*** %args_ptr517
%app_res_ptr519 = call %clos* %fn_ptr516(i8* %raw_app_rhs513, i8** %args518)
%app_res520 = load %clos, %clos* %app_res_ptr519
store %clos %app_res520, %clos* %app_lhs14
%app_rhs521 = alloca %adt
%app_lhs522 = alloca %clos
%app_lhs523 = alloca %clos
%clos_fn_ptr524 = getelementptr inbounds %clos, %clos* %app_lhs523, i32 0, i32
store i8* bitcast (%clos* (i8*, i8**)* @C to i8*), i8** %clos_fn_ptr524
%malloccall525 = tail call i8* @malloc(i32 0)
%env_aloc526 = bitcast i8* %malloccall525 to [0 x i8*]*
%env527 = getelementptr inbounds %clos, %clos* %app_lhs523, i32 0, i32 1
%env_ptr_raw528 = bitcast [0 x i8*]* %env_aloc526 to i8**
```

```
store i8** %env_ptr_raw528, i8*** %env527
%app_rhs529 = alloca i8
store i8 114, i8* %app_rhs529
%raw_fn_ptr_ptr530 = getelementptr inbounds %clos, %clos* %app_lhs523, i32 0,
%raw_fn_ptr531 = load i8*, i8** %raw_fn_ptr_ptr530
%fn_ptr532 = bitcast i8* %raw_fn_ptr531 to %clos* (i8*, i8**)*
%args_ptr533 = getelementptr inbounds %clos, %clos* %app_lhs523, i32 0, i32 1
%args534 = load i8**, i8*** %args_ptr533
%app_res_ptr535 = call %clos* %fn_ptr532(i8* %app_rhs529, i8** %args534)
%app_res536 = load %clos, %clos* %app_res_ptr535
store %clos %app_res536, %clos* %app_lhs522
%app_rhs537 = alloca %adt
%malloccall538 = tail call i8* @malloc(i32 0)
%cons539 = bitcast i8* %malloccall538 to %E*
%cons_vptr540 = bitcast %E* %cons539 to i8*
%tag541 = getelementptr inbounds %adt, %adt* %app_rhs537, i32 0, i32 0
%data_ptr542 = getelementptr inbounds %adt, %adt* %app_rhs537, i32 0, i32 1
store i64 0, i64* %tag541
store i8* %cons_vptr540, i8** %data_ptr542
%raw_app_rhs543 = bitcast %adt* %app_rhs537 to i8*
%raw_fn_ptr_ptr544 = getelementptr inbounds %clos, %clos* %app_lhs522, i32 0,
%raw_fn_ptr545 = load i8*, i8** %raw_fn_ptr_ptr544
%fn_ptr546 = bitcast i8* %raw_fn_ptr545 to %adt* (i8*, i8**)*
%args_ptr547 = getelementptr inbounds %clos, %clos* %app_lhs522, i32 0, i32 1
%args548 = load i8**, i8*** %args_ptr547
%app_res_ptr549 = call %adt* %fn_ptr546(i8* %raw_app_rhs543, i8** %args548)
%app_res550 = load %adt, %adt* %app_res_ptr549
store %adt %app_res550, %adt* %app_rhs521
%raw_app_rhs551 = bitcast %adt* %app_rhs521 to i8*
%raw_fn_ptr_ptr552 = getelementptr inbounds %clos, %clos* %app_lhs14, i32 0, i
%raw_fn_ptr553 = load i8*, i8** %raw_fn_ptr_ptr552
%fn_ptr554 = bitcast i8* %raw_fn_ptr553 to %adt* (i8*, i8**)*
%args_ptr555 = getelementptr inbounds %clos, %clos* %app_lhs14, i32 0, i32 1
%args556 = load i8**, i8*** %args_ptr555
%app_res_ptr557 = call %adt* %fn_ptr554(i8* %raw_app_rhs551, i8** %args556)
%app_res558 = load %adt, %adt* %app_res_ptr557
store %adt %app_res558, %adt* %unbox13
%box_ptr559 = bitcast %adt* %unbox13 to %boxt*
```

```
  %box_val560 = load %boxt, %boxt* %box_ptr559
  store %boxt %box_val560, %boxt* %app_rhs12
  %raw_app_rhs561 = bitcast %boxt* %app_rhs12 to i8*
  %raw_fn_ptr_ptr562 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32
  %raw_fn_ptr563 = load i8*, i8** %raw_fn_ptr_ptr562
  %fn_ptr564 = bitcast i8* %raw_fn_ptr563 to %boxt* (i8*, i8**)*
  %args_ptr565 = getelementptr inbounds %clos, %clos* %app_lhs, i32 0, i32 1
  %args566 = load i8**, i8*** %args_ptr565
  %app_res_ptr567 = call %boxt* %fn_ptr564(i8* %raw_app_rhs561, i8** %args566)
  %app_res568 = load %boxt, %boxt* %app_res_ptr567
  store %boxt %app_res568, %boxt* %boxed
  %unbox_ptr = bitcast %boxt* %boxed to i64*
  %unbox_val = load i64, i64* %unbox_ptr
  store i64 %unbox_val, i64* %ret
  %retval = load i64, i64* %ret
  ret i64 %retval
}
```

6.5 Team Distribution

As we were a team of three we each worked on almost all of the compiler and the tests are no different. Sophia made initial implementations of the regression testing scripts and wrote tests. Jay built the githooks, put the tests inside of the docker container, wrote tests and helped with the final implementation of the regression testing scipts. Ben helped write the testing scripts and wrote tests.

7 Lessons Learned

7.1 Sophia

I learned how complicated programming languages actually are. The fact that languages like C and Haskell exist is sort of baffling to me given how much work it was to do ours. I learned a lot about type theory and lambdas and the phrase "De Bruijn indices" will forever be floating in the back of my mind. I also realized how important test-driven development is. With something as complicated as a compiler it's important to take Edward's advice and build things up little by little otherwise you are pretty much screwed. My advice for future groups is to try

and be realistic about what you can accomplish in the semester, and to recognize that you're going to need to learn a lot of what you end up doing for your project outside of class. Also remember that Edwards and the TAs are great resources if you're completely lost on how to do something.

7.2 Ben

I learned so much about type theory this semester to do the project, probably even more than what I learned from the class itself. In fact, most of the work that went into this project was me thinking I understood something, coding it up, discovering I don't actually know it, and then having to rewrite it with my updated understanding. The one biggest piece of advice for groups later on trying to implement a functional programming language is to limit the feature set. The amount of code writing and re-writing you will have to do will generally mean that every feature will take at least twice as long as expected to implement. Additionally, every feature adds complexity as they all have to work and interact with each other. As for more general advice, it's crucial to the success of your project to take advantage of office hours and meeting times with your groups and to start early.

7.3 Jay

I think my most important learning was type theory. I took Edward's Parallel Functional Programming class the semester before I took PLT and I thought I understood typing. No, no, no. This project really helped me get inside a lot of the actual typing rules as we were reading papers and looking at different implementations of type checking and type inference throughout the semester working on the project. I am still by no means an expert but I definitely have a better understanding of how functional languages work under the hood and how they relate to type theory.

My advice for future teams is 1. start early. This is a big project and it takes a lot of time, we worked on it early and I feel like there was definitely more to get done. 2. Do not implement a functional programming language unless you really love functional languages or you are willing to work your butt off. There is a lot of really great learning to be done from this project and I think I gained a lot, however, there were a lot of late nights working on this project while I tried to understand why I was doing this to myself. 3. Go to your TA / Edwards office hours each week for advice from the pros. Not only did this keep us on track for

the semester, but Edwards almost always had the answers we were looking for an explained things elegantly. Take advantage.

8 Appendix

8.1 Work Log

* These commits may not be representative of the amount of actual work distribution. Because of the complexity of the type checker, we often met as a group over VS Code Live share and committed from one computer.

```
commit 58f1b8c8458c66a18e759f76d8bee7ef84fce3db
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sun Apr 25 23:08:50 2021 -0400

    changed syntax for lets and added final tests

commit 49f29262eeb13fa7f19a0e9cd55e00e1d52acf7b
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sun Apr 25 22:11:27 2021 -0400

    specialization for let when recursive and other fixes

commit 6c5e7ee5cc01ced1fd748dbc48d42ef326f46909
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sun Apr 25 17:16:13 2021 -0400

    added to demos and fixed debruijn index issue in typechecker

commit 8bbde5f132da62cd83935c8dbd90c93684df4f29
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sun Apr 25 09:53:35 2021 -0500

    added more tests

commit 466699d3bee2285a2d9c2eae659b1582ad9f0bb6
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sun Apr 25 10:49:03 2021 -0400
```

```
    added demos

commit 8895af4c14c31ce37ffedd337f8b9dd1510ebcae
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sun Apr 25 00:23:07 2021 -0500

    added more tests

commit 00cdf72bc9d145335c71b01ea892432a192da687
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sat Apr 24 21:55:37 2021 -0500

    added testing

commit a3a50a16b36a1b6752a98758f4ea64784557e53b
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sat Apr 24 18:19:12 2021 -0400

    moved lib.c and created compose_fail test

commit c98af31dcfcad4205ca921ef8a4527d083f56385
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sat Apr 24 18:02:15 2021 -0400

    second mega commit

commit 3c6bc7716a460b5b113ba55bdceb9532449c51ea
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sat Apr 24 15:40:39 2021 -0500

    executable compiler and linting

commit 4f09161550a551556814c716221299481ca226ff
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sat Apr 24 13:06:33 2021 -0500

    added lingoc compiler in src
```

```
commit 861a1d16990f6a9c9db1e41edc7afb3ce17f842b
Merge: 50ecbd2 d504357
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sat Apr 24 11:53:51 2021 -0500

    Merge branch 'main' of github.com:jaykarp/lingo into main

commit 50ecbd24eaa2bd2d3c629f721f4d450bdb32ddb9
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sat Apr 24 11:53:27 2021 -0500

    added cleaning after hook

commit 440617c1e9ab0880af0d06ac8117feebb4cbfe8e
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sat Apr 24 11:52:27 2021 -0500

    updated hook2

commit 837bb3a76d1870504eaecc6e9c02506c0a47a482
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sat Apr 24 11:51:18 2021 -0500

    updated hook

commit d5043577663b7901d25046246a6a01a6fee23aef
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sat Apr 24 12:36:35 2021 -0400

    linearity order relation and small parser fix

commit 6d22e2867bc0d3c90333258939667ba7e8568000
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sat Apr 24 11:01:39 2021 -0500

    added failing out of testing
```

```
commit 1e8040a357c62d7095993d2384eb1905208168db
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sat Apr 24 04:36:54 2021 -0400

    fixed linearity checking and issue with boxing interacting with higher order

commit 7156dc05f1bebdba5741e5a7591ce82008393fa0
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Thu Apr 22 18:17:22 2021 -0500

    ok jay is pushing to main

commit 66bf94e67ad5450d254d05c1071fad4e82550fc0
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Thu Apr 22 17:20:35 2021 -0500

    got rid of constraints in typechecker

commit 5c0c5bfd00b4408a12fef9d1c6f5f14a63458ad2
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Thu Apr 22 16:23:00 2021 -0500

    pre-typechecker linearity fix

commit d4e14e5e4bb56a92306ac48ac9a606641940303b
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Thu Apr 22 15:22:00 2021 -0500

    added colors to testing and fixed name conflicts with single quote asm

commit 7f3fd237acf0a48582af4f6da558fa12bb35f5cd
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Thu Apr 22 15:05:00 2021 -0500

    added die in cases and let

commit 196f262bfc45a7a15ea64ca57405501f2830a679
Author: Jay Karp <jaykarp2@gmail.com>
```

```
Date:   Thu Apr 22 11:43:45 2021 -0500

    added failing tests support

commit 1eae66b39bb3581c2901d7c9685ebc94232121a4
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Thu Apr 22 11:34:55 2021 -0500

    added random uid to end of function name

commit 77a283eebcd52e5ce47cc10d63f5ac6f0f4437ae
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Thu Apr 22 11:11:40 2021 -0500

    updated to ocaml 11 uuids also question mark

commit e92125ff4b133b1e53a802d4bcdc6fc4e2248a9c
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Tue Apr 20 00:25:14 2021 -0400

    updated TODO

commit 4a440c79bf20653ec198d0caec4f9a9e04589d8e
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Tue Apr 20 00:19:10 2021 -0400

    added to .gitignore in reg-tests

commit 90dcbd172dd479bf6a14c1909aacbc8108a8cea4
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Tue Apr 20 00:17:23 2021 -0400

    global constants now handled

commit 3f6acc5bf684cbc3b097a27b3b6151075718de66
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Mon Apr 19 23:28:24 2021 -0400
```

```
    updated malloc example

commit d7f87bf09a2cc121a5296d363dc8329427beb461
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Mon Apr 19 23:06:39 2021 -0400

    malloc example and lots of fixes

commit b060035931a1677f992e887e3f53f4ad90687e90
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Mon Apr 19 03:46:34 2021 -0400

    monomorphization fixed

commit 71c9c07cc03aff6fd407c27f71e578e168dc5217
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sun Apr 18 22:34:38 2021 -0400

    foreign function calls now work

commit 4a21d44292f083bc39118baaaab754aa182e8248
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sun Apr 18 20:25:26 2021 -0500

    adding fail cases

commit 46d4c72c878fc2ab6cac0c1fcdcb7cb667f3430e
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sun Apr 18 20:22:29 2021 -0500

    finished testing

commit 6e8fbcec34d3a8ab3aa88072c33f84810a2c1b6d
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sun Apr 18 20:07:33 2021 -0500

    severely improved testing
```

```
commit 3196761f7ef81e7c856d4bba6be7b7ba35b4a6be
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sun Apr 18 18:44:29 2021 -0500

    updating tests

commit 673aed44bfd7fa417766a40ef480426c6102ff9b
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sat Apr 17 02:22:13 2021 -0400

    updated TODO

commit 5632e77332c9ec4ccea5c3792252146b2fda2d1b
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sat Apr 17 02:14:43 2021 -0400

    fixed small closure issue

commit 5d1f23d7b1953b29995a0c6a75ea063585b11bb7
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Fri Apr 16 02:20:20 2021 -0400

    added binop and unop operations and fac example

commit 2f436f7fd9559a58cc2389ff769d280278abde5e
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Fri Apr 16 00:51:31 2021 -0400

    fixed mono for cases

commit 57c8d2062735f2d8fd3cd7dbeca14f54dae0fa8d
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Thu Apr 15 16:51:04 2021 -0400

    fixed codegen closure malloc bug

commit e1dd4ec8dccabf1cd5a2d1b148b9d6226780fc7d
Author: Benjamin Flin <brf2117@columbia.edu>
```

```
Date:   Thu Apr 15 03:33:25 2021 -0400

    mega commit for code generation

commit 31a9664656adec086ba0f12c3d33101a7dd805b4
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Wed Apr 14 23:43:37 2021 -0400

    closure conversion initial implementation

commit 2cbe9782bb23e354be238c7c43e15f78af3fc20a
Merge: 3e5e931 22cf3e7
Author: benjaminflin <brf2117@columbia.edu>
Date:   Mon Apr 12 20:01:12 2021 -0400

    Merge pull request #11 from jaykarp/mono

    Mono

commit 22cf3e78bd5f201e6a6982e610e0f0aa82595d9e
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Mon Apr 12 20:00:27 2021 -0400

    added datadef conversion

commit 3e5e931281b3bf61ce84425c5418086f7c781b9d
Merge: 7d81587 abee6c4
Author: jaykarp <45463412+jaykarp@users.noreply.github.com>
Date:   Mon Apr 12 14:56:54 2021 -0500

    Merge pull request #10 from jaykarp/codegen

    Codegen

commit abee6c4158c3063dfa8ce5c67f445d4ac6044332
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Mon Apr 12 15:30:58 2021 -0400
```

```
    finished code generation

commit 7c6440889f0492ae0ee87009d68b1d59b91dd09e
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Fri Apr 9 19:04:18 2021 -0400

    fixed typo

commit ca40fb25ce33477a8b4acbb738eb120b91eaf846
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Fri Apr 9 19:03:28 2021 -0400

    fixed construction

commit bf864f76de118405cb821c6804855de90e7521ac
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Fri Apr 9 18:30:06 2021 -0400

    application and construction

commit ee4a58fd5fbd6f1b71ab10fd1a899bc5e35b4a36
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Tue Apr 6 01:15:59 2021 -0700

    some codegen implemented

commit 3aebbe2874f3304bb1b0fa10f9fa90ed4666c815
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Mon Apr 5 15:16:29 2021 -0700

    renamed Box

commit 2107972687c6d174eebd5d5a045974c169403e70
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sat Apr 3 22:15:39 2021 -0700

    initial (most likely buggy) implementation
```

```
commit 7d815878d33cf6d3fa14984ada5c63310a1bd3f5
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Fri Apr 2 15:53:08 2021 -0700

    fixed broken typechecking test

commit c528eefc56cc04b834bf279be2ae1d3aa3befd5d
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Fri Apr 2 15:51:38 2021 -0700

    added sast generation from sast branch

commit a35e163e13fed010779d4891ec51ad2e7182e735
Author: Sophia Kolak <sophiakolak.sk@gmail.com>
Date:   Fri Apr 2 00:06:25 2021 -0400

    fixed fail condition

commit bccdfe327fe5e2753f75b07c9b507fe41cc0b986
Merge: 923caea 37b1721
Author: Sophia Kolak <sophiakolak.sk@gmail.com>
Date:   Thu Apr 1 14:03:11 2021 -0400

    Merge branch 'main' of https://github.com/jaykarp/lingo into main

commit 923caead9bcfdc395853a24a0f3bc6f8512d39f7
Author: Sophia Kolak <sophiakolak.sk@gmail.com>
Date:   Thu Apr 1 14:00:11 2021 -0400

    .ml file for run helloworld, helloworld+sample out

commit 7d5a10f38b0680fbb7ea7eceff1bed03e3ff591b
Author: Sophia Kolak <sophiakolak.sk@gmail.com>
Date:   Thu Apr 1 13:59:52 2021 -0400

    script to execute reg-tests

commit 37b172129739234657d5dc4917811352a02195bc
```

```
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Wed Mar 31 22:06:40 2021 -0700

    added closure defs to closed ast

commit 35b98997b52e16258dc45bd863ce911925c4cd35
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Wed Mar 31 21:56:34 2021 -0700

    modified closed ast and added example

commit 9c74ba232cef343cdaf5746fc07b3c810d302e47
Merge: 2f08cb5 c1ba098
Author: Sophia Kolak <sophiakolak.sk@gmail.com>
Date:   Tue Mar 30 21:36:15 2021 -0400

    Merge branch 'main' of https://github.com/jaykarp/lingo into main

commit c1ba098692c3966fce4c833e4450f867d6ca6ddd
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Tue Mar 30 15:13:36 2021 -0700

    updated closed ast

commit 81d44a85256178342d26dc6d4b3f6927b063b098
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Tue Mar 30 15:12:08 2021 -0700

    updated closed ast and added third closure example

commit 2f08cb58b36b8311804db47ac4dbba100df7c59d
Merge: 221e889 7fa2ab7
Author: Sophia Kolak <sophiakolak.sk@gmail.com>
Date:   Tue Mar 30 10:35:54 2021 -0400

    Merge branch 'main' of https://github.com/jaykarp/lingo into main

commit 7fa2ab7c10bebc814317429c6fe2f90ad734ebd4
```

```
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Mon Mar 29 17:00:05 2021 -0700

    added closed ast

commit 221e8899f1b5ba5e0ac22cd13f5a037699d0e705
Merge: a5eac5e 0606ad0
Author: Sophia Kolak <sophiakolak.sk@gmail.com>
Date:   Thu Mar 25 15:21:50 2021 -0400

    Merge branch 'main' of https://github.com/jaykarp/lingo into main

commit 0606ad0ca89137406275b166ae1aed414db93c8d
Merge: 9289a0a 967396f
Author: jaykarp <45463412+jaykarp@users.noreply.github.com>
Date:   Mon Mar 22 20:50:58 2021 -0500

    Merge pull request #9 from jaykarp/typecheck-cases

    Typecheck cases

commit 967396f9c0c27d91e81d389daadc2b8d55b5dbfe
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Mon Mar 22 18:50:20 2021 -0700

    changed gadt_eq test and docker-compose

commit dc178591262644acaf13bbe9efc36dd0fa6901d1
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Mon Mar 22 18:18:14 2021 -0700

    case expressions typecheck correctly

commit fc6ea1f4b8ecee7f7ece7ce0ac26404d7d54bcfe
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sun Mar 21 14:08:53 2021 -0700

    cases downgraded to checkable and proper equality checking for GADTs
```

```
commit 2b0c790b682016d28266352709c5a48e3f8f2881
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sat Mar 20 21:15:17 2021 -0700

    typechecking cases now complete

commit 9289a0ae4001afba45f4d0228fc123c5b853e810
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Thu Mar 18 21:22:15 2021 -0500

    added tag instead of t

commit a5eac5e57af6886b117616994f76f308e47b1b1c
Merge: cd90e45 067dffb
Author: Sophia Kolak <sophiakolak.sk@gmail.com>
Date:   Thu Mar 18 22:21:22 2021 -0400

    Merge branch 'main' of https://github.com/jaykarp/lingo into main

commit 067dffb8759c789e818d0c1ab7c715b8f75d919f
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Thu Mar 18 21:20:55 2021 -0500

    updated readme

commit cd90e45ca7193a03e2a056517cd02f91ec834ee9
Merge: 7139a61 ae45deb
Author: Sophia Kolak <sophiakolak.sk@gmail.com>
Date:   Thu Mar 18 22:19:25 2021 -0400

    Merge branch 'main' of https://github.com/jaykarp/lingo into main

commit ae45deb83362c2142b20eff625ab2df710604137
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Thu Mar 18 21:15:22 2021 -0500

    added test_all
```

```
commit 7139a6118d33bfd3affffe3bd624821a1849690f
Merge: eca8a61 617a35e
Author: Sophia Kolak <sophiakolak.sk@gmail.com>
Date:   Thu Mar 18 22:14:44 2021 -0400

    Merge branch 'main' of https://github.com/jaykarp/lingo into main

commit 617a35ee350efdd19f576091a0377cdb37e469a6
Merge: 25d250a 0c20120
Author: jaykarp <45463412+jaykarp@users.noreply.github.com>
Date:   Thu Mar 18 21:13:15 2021 -0500

    Merge pull request #8 from jaykarp/tests

    initial hello_world test

commit 25d250a6cc5e1cecae7e78d1d573af92000a36cd
Merge: cdc5ddd a7df4c6
Author: jaykarp <45463412+jaykarp@users.noreply.github.com>
Date:   Thu Mar 18 21:12:44 2021 -0500

    Merge pull request #7 from jaykarp/docker

    added docker automated test script

commit 0c201203b202e3e679fcc0bebdb1836eda331bfc
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Thu Mar 18 19:12:42 2021 -0700

    initial hello_world test

commit a7df4c6e4b3cd4adff3430b5bd6f900a9b256864
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Thu Mar 18 20:56:11 2021 -0500

    added docker automated test script
```

```
commit eca8a6135b52e060589a1edcee435e970ec9a529
Merge: 9757e88 cdc5ddd
Author: Sophia Kolak <sophiakolak.sk@gmail.com>
Date:   Thu Mar 18 20:22:14 2021 -0400

    Merge branch 'main' of https://github.com/jaykarp/lingo into main

commit cdc5dddfce28cb3a8aea18062a862b75d246729e
Merge: e07a836 9e792bb
Author: jaykarp <45463412+jaykarp@users.noreply.github.com>
Date:   Thu Mar 18 19:15:32 2021 -0500

    Merge pull request #6 from jaykarp/codegen

    hello world

commit 9e792bb8fb9e7a897f2bbe02f2bbd3c7cfff80be
Merge: 065771d e07a836
Author: jaykarp <45463412+jaykarp@users.noreply.github.com>
Date:   Thu Mar 18 19:15:27 2021 -0500

    Merge branch 'main' into codegen

commit e07a836b95fc0670ee289625d86439a2996c54b8
Merge: d3b7b93 d107429
Author: jaykarp <45463412+jaykarp@users.noreply.github.com>
Date:   Thu Mar 18 19:13:57 2021 -0500

    Merge pull request #5 from jaykarp/core

    Core

commit 9757e88402edef4a27388f96d78f8d174e79f55d
Merge: 3e91072 d3b7b93
Author: Sophia Kolak <sophiakolak.sk@gmail.com>
Date:   Thu Mar 18 14:56:20 2021 -0400

    Merge branch 'main' of https://github.com/jaykarp/lingo into main
```

```
commit d10742975fbaa8ac21ff9b88410dd1aabb1b92ff
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Wed Mar 17 22:24:16 2021 -0700

    small parser fix and type substitution in cases

commit 407eb969ffcb1c2e03b8685280babd86792782b4
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Wed Mar 17 13:25:39 2021 -0700

    small parser fixes

commit e708db2f8414dc025864e7c450dba5a2f8b23577
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Wed Mar 17 00:05:00 2021 -0700

    letdef typechecking and small bug fix in conversion

commit a412724dcdfc8035f4ceaa49dcfcef477b744425
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Tue Mar 16 22:18:31 2021 -0700

    partial implementation typechecking

commit 90b6e48199f4931b1fcbdaa57774b2bb3ae168d4
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Tue Mar 16 00:59:55 2021 -0700

    kind checking complete

commit d25992368a2ce73fddf6de67f73b15d5a656bc53
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Mon Mar 15 19:08:54 2021 -0700

    conversion using debrujin indices

commit 08e15480d2372a1e502fcae6906517d87a4bb844
```

```
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Mon Mar 15 17:15:58 2021 -0700

    small cleanup

commit deceb628b3d886c02909e42e844f64b7a01a7280
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sun Mar 14 22:58:34 2021 -0700

    fixed small linearity checking issue in lambda

commit 267580b1acd4ea6c5c758cdeafe145e94702dd74
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sun Mar 14 22:32:29 2021 -0700

    first program checked

commit 065771dea9870f714c750fc49745aa955adb5212
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sun Mar 14 16:23:43 2021 -0700

    hello world

commit 269ff19f745462424d543505c93b5862eab9cb7d
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sun Mar 14 13:19:45 2021 -0700

    annotations and if stmts typechecking

commit ad3d7c7dfc336c80f23884a114148aec773cddea
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Tue Mar 9 23:22:04 2021 -0800

    changes to let expr and more typechecking

commit 232df6e5b5e10af4edf3cda4b5d7663cac68cfdc
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Mon Mar 8 13:15:40 2021 -0800
```

```
    small parser fixes

commit 71455a2c47c59950fc125d4cffcc9ec522ddb08e
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Mon Mar 8 12:39:24 2021 -0800

    small fixes to conversion

commit 7ea9a10bf2b5cdb05a6e45e5f4c09f0da0015986
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Mon Mar 8 12:32:57 2021 -0800

    intial implementation of conversion to core

commit 59e5ecf4adda87578a7138b74ff874f1ee8a06e9
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Sun Mar 7 13:39:29 2021 -0800

    major parser changes

commit 26df3957b4c1e12b4704d99f74d778dcd6febde2
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sat Mar 6 21:30:18 2021 -0600

    working on cases

commit 302336f314283a0badc39b774fd89e0094d8bc26
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sat Feb 27 17:33:05 2021 -0600

    typechecker let

commit 42b4d759e55572c1fc1d3af8278d3823dc22ffda
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sat Feb 27 16:35:39 2021 -0600

    working on typechecker
```

```
commit d3b7b93268c33bf09611ea0330d0248d4d38cbf4
Merge: 1bc6425 3b1b76b
Author: jaykarp <45463412+jaykarp@users.noreply.github.com>
Date:   Sat Feb 27 14:17:44 2021 -0600

    Merge pull request #4 from jaykarp/Scanner

    Scanner

commit 3b1b76b8fcbc50f4905fe1acb2d78796876ea86f
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sat Feb 27 14:17:18 2021 -0600

    first pass of scanner done

commit babfa1dd3180569e24c64e8084c8c4153830aa7f
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Tue Feb 23 23:27:05 2021 -0600

    added src9

commit 3e910727f0911a756e98a25818b288d477d55d02
Merge: 1aac614 9a4c585
Author: Sophia Kolak <sophiakolak.sk@gmail.com>
Date:   Mon Feb 22 22:39:05 2021 -0500

    Merge branch 'Scanner' of https://github.com/jaykarp/lingo into main

commit 9a4c5857fd0b7f94ebe70736cb2e0e8cdaf9d724
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Mon Feb 22 19:02:47 2021 -0600

    writing lrm with parser

commit 1aac614e618294df89b09c130d23e39c65c263ec
Merge: d36b3cc 1bc6425
Author: Sophia Kolak <sophiakolak.sk@gmail.com>
```

```
Date:   Mon Feb 22 12:50:58 2021 -0500

    Merge branch 'main' of https://github.com/jaykarp/lingo into main
    oi

commit 703e861b0451bb5eb2fee915497d8746b22589d1
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sat Feb 20 17:24:38 2021 -0600

    still more to work on for parser and pretty printer

commit ad86c0f4ddb8d6dd24897606b971d8f4e42ebfb2
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sat Feb 20 15:25:16 2021 -0600

    seems that the scanner is done

commit e832999cb137be8a1632e160904dfa380fd2ac68
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Fri Feb 19 22:04:10 2021 -0800

    fixed some parser issues

commit a60dd492b2f6b7239132ef9610c831523e8e1a2e
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Fri Feb 19 21:29:11 2021 -0800

    initial parser implementation finished

commit 7fd77c53ce988e13c3c541189c71dddceac6f8e0
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Fri Feb 19 19:36:39 2021 -0600

    hunting shift reductions

commit d36b3cc882cb0dd758f73abe9dfb22c579d7c005
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Thu Feb 18 22:35:10 2021 -0800
```

```
    partial parsing of lambdas and let expressions

commit b16c12897ed10a074e1773fb793e7bc83a629b96
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Thu Feb 18 22:37:45 2021 -0600

    working on scanner and parser

commit 4aa49db8f6a1311feeaad970bb650c5fffe223a3
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Thu Feb 18 21:35:11 2021 -0600

    added second closure and lrm.lingo examples

commit 1bc6425fb88f3c8b4edf91dc038d51a64ea72a9c
Merge: 0008820 1b67d67
Author: jaykarp <45463412+jaykarp@users.noreply.github.com>
Date:   Thu Feb 18 14:02:27 2021 -0600

    Merge pull request #3 from jaykarp/Scanner

    Scanner

commit 000882041d0c8e35d7e5846634acebb945c07f76
Merge: 40c6f4b a62ffd6
Author: jaykarp <45463412+jaykarp@users.noreply.github.com>
Date:   Thu Feb 18 14:02:10 2021 -0600

    Merge pull request #2 from jaykarp/ClosureConverstion

    Closure converstion

commit 1b67d67a25aef0983225dbd18f50338f1cffd118
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Wed Feb 17 23:42:47 2021 -0600

    testing parser imports in lingo
```

```
commit a3e1203070b1b75e45415286f296eba7fbb0e88e
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Wed Feb 17 23:28:14 2021 -0600

    added dune configuration for parser monke

commit 7ffa1ed8844626e9e24dbd4c09154ae06aa9967a
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Wed Feb 17 22:28:32 2021 -0600

    added more closures and explained to sophia

commit a62ffd6619ca322f5cca5197818066b31cc6a8e5
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sun Feb 14 17:41:36 2021 -0600

    fixed comments

commit c835cc97eff9b5cac849744a12c96f9ec05af9f8
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sun Feb 14 17:36:46 2021 -0600

    added compose example

commit 40c6f4b1f877c8b328f9360bdf62e2285e03ef46
Merge: 5938a64 85a5ac7
Author: jaykarp <45463412+jaykarp@users.noreply.github.com>
Date:   Sun Feb 14 15:18:51 2021 -0600

    Merge pull request #1 from jaykarp/Docker

    Docker

commit 85a5ac7e5dc372ece0f5b23f1ec33adb988e0ba1
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sun Feb 14 12:31:56 2021 -0600
```

```
    cleaned dockerfile

commit 64dd39d7ba297837dc89e09eb8b221792a64f112
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sun Feb 14 12:09:57 2021 -0600

    added docker building and updated readme

commit dfd6a1fd57ee1c3691dec786a1d9f3c98735fd0a
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Sat Feb 13 21:32:55 2021 -0600

    updated one line to add f to RWS

commit 3937c237caf7ed35d43ba711eedf32054f87d46d
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Fri Feb 5 20:43:59 2021 -0600

    added subst and simp

commit e1eddfabd8403798cb9e0d23e553d93af6e162d0
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Fri Feb 5 19:54:49 2021 -0600

    fixed match

commit 451334166b56800a0261b0d7cf8b7651079c16a5
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Fri Feb 5 00:49:13 2021 -0800

    initial impl of var, lam, app, and base expr

commit 69baf5875f4351926a22ca4011a1bf6d030d395e
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Thu Feb 4 23:58:26 2021 -0800

    more typechecker definitions
```

```
commit aa9481dce01ff0d8fbcdfaffed08486eb2f87442
Author: Benjamin Flin <brf2117@columbia.edu>
Date:   Thu Feb 4 15:50:42 2021 -0800

    preliminary type definitions for typechecker

commit 5938a64861ed5fd575e2fae4f02425bad0479a71
Author: Jay Karp <jaykarp2@gmail.com>
Date:   Wed Feb 3 23:41:26 2021 -0600

    Dune Initialization

commit 3ab527d4e93c99e07923ad51e52c43f134337176
Author: jaykarp <45463412+jaykarp@users.noreply.github.com>
Date:   Fri Jan 15 16:50:12 2021 -0600

    Update LICENSE

commit c92e5cea6243efc2bf18911bf278d6da298c58e8
Author: jaykarp <45463412+jaykarp@users.noreply.github.com>
Date:   Fri Jan 15 16:33:06 2021 -0600

    Initial commit
```

8.2 Source Files

8.2.1 lib/parser/scanner.mll

```
(*
Authors: Ben, Sophia, Jay
Ocamllex Scanner *)

{ open Parser }
let digit = ['0'-'9']
let digits = digit+
let lchar = ['a'-'z']
let uchar = ['A'-'Z']
let schar = [ ^ '"' ]
```

```
let cchar = [ ^ '\'' ]

rule tokenize = parse
  [' ' '\t' '\r' '\n']                { tokenize lexbuf }
| "(*"                                { comment lexbuf }
| "!="                                { NEQ }
| "<="                                { LEQ }
| '<'                                 { LT }
| ">="                                { GEQ }
| '>'                                 { GT }
| "=="                                { EQ }
| '='                                 { ASSIGN }
| "||"                                { OR }
| "&&"                                { AND }
| "!"                                 { NOT }
| "()"                                { UNIT }
| ':'                                 { COLON }
| ';'                                 { SEMICOLON }
| '_'                                 { WILDCARD }
| '\\'                                { BACKSLASH }
| '`'                                 { BACKTICK }
| '.'                                 { DOT }
| '('                                 { LPAREN }
| ')'                                 { RPAREN }
| '@'                                 { FORALL }
| '#'                                 { FORALLM }
| '+'                                 { PLUS }
| '-'                                 { DASH }
| '*'                                 { STAR }
| '/'                                 { SLASH }
| "if"                                { IF }
| "then"                              { THEN }
| "else"                              { ELSE }
| "let"                               { LET }
| "in"                                { IN }
| "of"                                { OF }
| "data"                              { DATA }
| "case"                              { CASE }
```

```
| "where"                                  { WHERE }
| "true"                                   { BOOL(true) }
| "false"                                  { BOOL(false) }
| "Unr"                                    { UNR }
| "One"                                    { ONE }
| digits as lxm                            { LITERAL(int_of_string lxm) }
| '"' ((schar*) as lxm) '"'                { STRING (lxm) }
| '\'' (cchar as lxm) '\''
                                           { CHAR(lxm) }
| '\'' ((cchar*) as lxm) '\''              { CHAR(String.get (Scanf.unescaped lxm)
| lchar+ (digit | lchar | uchar | '_')* ('\'')* as lxm
                                           { LID(String.map (fun x -> if x = '\'' t
| uchar+ (digit | lchar | uchar | '_')* ('\'')* as lxm
                                           { UID(String.map (fun x -> if x = '\'' t
| eof                                      { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char ))}

and comment = parse
  "*)"  { tokenize lexbuf }
| _     { comment lexbuf }
```

8.2.2 lib/parser/parser.mly

```
%{
(* Authors: Sophia, Ben, Jay *)
open Ast
let build_string str =
    let explode s = List.init (String.length s) (String.get s) in
    List.fold_left (fun cons ch -> App(App(Construction "C", Infer (Char ch)), I
%}

%token COLON SEMICOLON WILDCARD BACKSLASH LPAREN RPAREN UNIT BACKTICK DOT
%token PLUS DASH STAR SLASH ASSIGN EQ NEQ LT LEQ GT GEQ OR AND NOT
%token IF THEN ELSE LET IN OF DATA CASE WHERE FORALL FORALLM
%token UNR ONE

%token <int>    LITERAL
%token <bool>   BOOL
```

```
%token <string> UID LID
%token <char>   CHAR
%token <string> STRING

%token  EOF

%left   OR
%left   AND
%left   EQ NEQ
%left   LT GT LEQ GEQ
%left   PLUS DASH
%left   STAR SLASH
%right  NOT

%start  program
%type   <Ast.program> program

%%

program:
| def { [$1] }
| def program { $1::$2 }

def:
| let_def { $1 }
| data_def { $1 }
| let_decl { $1 }

let_decl:
| LID COLON ty SEMICOLON          { LetDecl($1, $3) }

data_def:
| DATA UID data_param_list WHERE cons_list { DataDef($2, $3, $5) }
| DATA UID WHERE cons_list                 { DataDef($2, [], $4) }

data_param_list:
| data_param { [$1] }
| data_param data_param_list { $1::$2 }
```

```
data_param:
| FORALL LID    { TypeParam($2) }
| LID           { TypeParam($1) }
| FORALLM LID   { MultParam($2) }

cons_list:
|                 { [] }
| cons cons_list { $1::$2 }

cons:
| UID COLON ty SEMICOLON { Cons($1, $3) }

let_def:
| LID COLON ty ASSIGN check_expr SEMICOLON           { LetDef($1, [], $3, $5) }
| LID name_list COLON ty ASSIGN check_expr SEMICOLON { LetDef($1, $2, $4, $6) }

lambda:
| BACKSLASH LID DOT check_expr                { Lam($2, $4) }
| LPAREN lambda RPAREN                        { $2 }

check_expr:
| lambda                                              { $1 }
| CASE infer_expr OF case_alts                        { Case($2, $4) }
| infer_expr                                          { Infer($1) }

infer_expr:
| IF infer_expr THEN infer_expr ELSE infer_expr     { If($2, $4, $6) }
| NOT infer_expr                                    { App(Unop(Not), Infer($2))
| DASH infer_expr %prec NOT                         { App(Unop(Neg), Infer($2))
| app_term                                          { $1 }
| let_expr                                          { $1 }
| bin_operation                                     { $1 }
| app_term lambda                                   { App($1, $2) }

let_expr:
| LET FORALLM mult LID COLON ty ASSIGN check_expr IN infer_expr
| LET LID COLON ty ASSIGN check_expr IN infer_expr
```

```
| LET FORALLM mult LID name_list COLON ty ASSIGN check_expr IN infer_expr
| LET LID name_list COLON ty ASSIGN check_expr IN infer_expr


app_term:
| atomic_term                                       { $1 }
| app_term atomic_term                             { App($1, Infer($2)) }
| app_term BACKTICK LID BACKTICK atomic_term       { App(App(Var($3), Infer($1)

atomic_term:
| LPAREN infer_expr RPAREN      { $2 }
| LPAREN check_expr COLON ty RPAREN
                                { Ann($2, $4) }
| UNIT                          { Construction("Unit") }
| LID                           { Var($1) }
| UID                           { Construction($1) }
| LITERAL                       { Int($1) }
| BOOL                          { Bool($1) }
| CHAR                          { Char($1) }
| STRING                        { build_string ($1) }
| FORALL atomic_ty              { Type($2) }
| FORALLM atomic_mult           { Mult($2) }

case_alts:
| case_alt           { [$1] }
| case_alt case_alts  { $1::$2 }

case_alt:
| UID name_list DASH GT infer_expr SEMICOLON   { Destructor($1, $2, $5) }
| UID DASH GT infer_expr SEMICOLON             { Destructor($1, [], $4) }
| WILDCARD DASH GT infer_expr SEMICOLON        { Wildcard($4) }

bin_operation:
| bterm PLUS bterm      { App(App(Binop(Plus), Infer($1)), Infer($3)) }
| bterm DASH bterm      { App(App(Binop(Minus), Infer($1)), Infer($3)) }
| bterm SLASH bterm     { App(App(Binop(Divide), Infer($1)), Infer($3)) }
| bterm STAR bterm      { App(App(Binop(Times), Infer($1)), Infer($3)) }
| bterm OR bterm        { App(App(Binop(Or), Infer($1)), Infer($3)) }
```

```
| bterm AND bterm       { App(App(Binop(And), Infer($1)), Infer($3)) }
| bterm EQ bterm        { App(App(Binop(Eq), Infer($1)), Infer($3)) }
| bterm NEQ bterm       { App(App(Binop(Neq), Infer($1)), Infer($3)) }
| bterm LEQ bterm       { App(App(Binop(Leq), Infer($1)), Infer($3)) }
| bterm GT bterm        { App(App(Binop(Gt), Infer($1)), Infer($3)) }
| bterm LT bterm        { App(App(Binop(Lt), Infer($1)), Infer($3)) }
| bterm GEQ bterm       { App(App(Binop(Geq), Infer($1)), Infer($3)) }


bterm:
| app_term           { $1 }
| bin_operation      { $1 }

ty:
| FORALL LID ty      { Forall($2, $3) }
| FORALLM LID ty     { ForallM($2, $3) }
| mapp_ty arrow ty   { Arr($2, $1, $3) }
| mapp_ty            { $1 }

mapp_ty:
| mapp_ty FORALLM mult { InstM($1, $3) }
| mapp_ty FORALL atomic_ty { Inst($1, $3) }
| app_ty { $1 }

app_ty:
| app_ty atomic_ty    { Inst($1, $2) }
| atomic_ty           { $1 }

atomic_ty:
| UNIT { TName "Unit" }
| LID { TVar $1 }
| UID { TName $1 }
| LPAREN ty RPAREN  { $2 }

arrow:
| DASH GT { Unr }
| DASH STAR { One }
| DASH mult GT { $2 }
```

```
mult:
| mult STAR mult                { MTimes($1, $3) }
| atomic_mult                   { $1 }

atomic_mult:
| LPAREN mult RPAREN { $2 }
| LID { MVar($1) }
| UNR { Unr }
| ONE { One }

name_list:
| LID { [$1] }
| LID name_list { $1::$2 }
```

8.2.3 lib/parser/ast.ml

```
(* Authors: Jay, Sophia, Ben *)

type mult
  = One
  | Unr
  | MVar of string
  | MTimes of mult * mult

type name = string

type ty
  = TVar of name
  | TName of name
  | Arr of mult * ty * ty
  | Inst of ty * ty
  | InstM of ty * mult
  | Forall of name * ty
  | ForallM of name * ty

type binop
  = Lt | Gt | Leq | Geq | Neq | Eq | Plus | Minus | Divide | Times | And | Or
```

```
type unop = Not | Neg

type check_expr
  = Lam of name * check_expr
  | Case of infer_expr * case_alt list
  | Infer of infer_expr
and infer_expr
  = Var of name
  | Binop of binop
  | Unop of unop
  | Let of name * mult * name list * ty * check_expr * infer_expr
  | Type of ty
  | Mult of mult
  | App of infer_expr * check_expr
  | Construction of name
  | If of infer_expr * infer_expr * infer_expr
  | Ann of check_expr * ty
  | Int of int
  | Char of char
  | Bool of bool
and case_alt
  = Destructor of name * name list * infer_expr
  | Wildcard of infer_expr

type cons_def
  = Cons of name * ty

type data_param
  = MultParam of name
  | TypeParam of name

type def
  = LetDef of name * name list * ty * check_expr
  | DataDef of name * data_param list * cons_def list
  | LetDecl of name * ty

type program = def list
```

8.2.4 lib/core/conversion.ml

```
(* Authors: Ben *)
open Parse
module Tc = Typecheck

exception NotImplemented
exception UnboundVariable of string
exception ExpectedArrow of Ast.ty
exception TypeInWrongPlace of Ast.ty
exception MultInWrongPlace of Ast.mult
exception ImpredicativeType

let index_of y xs =
  let rec index_of' y i = function
  | []      -> raise Not_found
  | (x::xs) -> if x = y then i else index_of' y (i + 1) xs
  in
  index_of' y 0 xs

let rec convert_mult dbmap = function
| Ast.One -> Tc.One
| Ast.Unr -> Tc.Unr
| Ast.MTimes (a, b) -> Tc.MTimes (convert_mult dbmap a, convert_mult dbmap b)
| Ast.MVar name ->
  try
    Tc.MVar (index_of name dbmap)
with Not_found -> raise (UnboundVariable name)

let tname_to_ty params = function
| "Int" -> Tc.BaseT Tc.IntT
| "Bool" -> Tc.BaseT Tc.BoolT
| "Char" -> Tc.BaseT Tc.CharT
| x -> Tc.DataTy (x, params)

let rec convert_ty dbmap ty =
  let rec convert_ty' dbmap params = function
  | Ast.TName name -> tname_to_ty params name
```

```
  | Ast.TVar name -> (
    try
      Tc.TVar (index_of name dbmap)
    with Not_found -> raise (UnboundVariable name)
    )

  | Ast.Arr (mult, in_ty, out_ty) ->
    Tc.Arr ( convert_mult dbmap mult,
             convert_ty' dbmap params in_ty,
             convert_ty' dbmap params out_ty
           )

  | Ast.Inst (to_inst, with_ty) ->
    convert_ty' dbmap ((Type (convert_ty dbmap with_ty))::params) to_inst

  | Ast.InstM (to_inst, with_mult) ->
    convert_ty' dbmap ((Mult (convert_mult dbmap with_mult))::params) to_inst

  | Ast.Forall (name, ty) ->
    Tc.Forall (convert_ty' (name::dbmap) params ty)

  | Ast.ForallM (name, ty) ->
    Tc.ForallM (convert_ty' (name::dbmap) params ty)
  in
  convert_ty' dbmap [] ty

let convert_binop = function
| Ast.Leq -> Tc.Leq
| Ast.Geq -> Tc.Leq
| Ast.Neq -> Tc.Neq
| Ast.Eq -> Tc.Eq
| Ast.Lt -> Tc.Lt
| Ast.Gt -> Tc.Gt
| Ast.And -> Tc.And
| Ast.Or -> Tc.Or
| Ast.Times -> Tc.Times
| Ast.Divide -> Tc.Divide
| Ast.Plus -> Tc.Plus
```

```
| Ast.Minus -> Tc.Minus

let convert_unop = function
| Ast.Not -> Tc.Not
| Ast.Neg -> Tc.Neg


let rec convert_check_expr dbmap = function
| Ast.Lam (name, cexpr) -> Tc.Lam (convert_check_expr (name::dbmap) cexpr)
| Ast.Case (iexpr, ca_list) ->
  Tc.Case ( convert_infer_expr dbmap iexpr,
            List.map (convert_case_alt dbmap) ca_list
          )
| Ast.Infer iexpr       -> Tc.Infer (convert_infer_expr dbmap iexpr)

and convert_infer_expr dbmap = function
| Ast.Var name -> (
  try
    Tc.DbIndex (index_of name dbmap)
  with Not_found -> Tc.Global name
  )
| Ast.Binop binop ->
  Tc.Binop (convert_binop binop)

| Ast.Unop unop ->
  Tc.Unop (convert_unop unop)

| Ast.Let (name, mult, name_list, ty, cexpr, iexpr) ->
  (match ty with
  | Ast.Arr (_, _, _) ->
    Tc.Let ( convert_mult [] mult,
             convert_ty [] ty,
             (name_list_to_lam (name::dbmap) cexpr name_list),
             convert_infer_expr (name::dbmap) iexpr
           )
  | _ ->
    Tc.Let ( convert_mult [] mult,
             convert_ty [] ty,
```

```
            name_list_to_lam dbmap cexpr name_list,
            convert_infer_expr (name::dbmap) iexpr
          )
  )

| Ast.App (iexpr, Ast.Infer (Ast.Type ty)) ->
  Tc.TApp (convert_infer_expr dbmap iexpr, convert_ty dbmap ty)

| Ast.App (iexpr, Ast.Infer (Ast.Mult mult)) ->
  Tc.MApp (convert_infer_expr dbmap iexpr, convert_mult dbmap mult)

| Ast.App (iexpr, cexpr) ->
  Tc.App (convert_infer_expr dbmap iexpr, convert_check_expr dbmap cexpr)

| Ast.Type ty -> raise (TypeInWrongPlace ty)

| Ast.Mult mult -> raise (MultInWrongPlace mult)

| Ast.Construction name -> Tc.Construction name



| Ast.Ann (cexpr, ty) ->
  Tc.Ann (convert_check_expr dbmap cexpr, convert_ty [] ty)

| Ast.If (iexpr, iexpr_a, iexpr_b) ->
  Tc.If ( convert_infer_expr dbmap iexpr,
          convert_infer_expr dbmap iexpr_a,
          convert_infer_expr dbmap iexpr_b
        )

| Ast.Int i -> Tc.Int i

| Ast.Char c -> Tc.Char c

| Ast.Bool b -> Tc.Bool b

and convert_case_alt dbmap = function
```

```
| Ast.Destructor (name, name_list, iexpr) ->
  Tc.Destructor ( name,
                  List.length name_list,
                  convert_infer_expr ((List.rev name_list) @ dbmap) iexpr
                )

| Ast.Wildcard (iexpr) ->
  Tc.Wildcard (convert_infer_expr dbmap iexpr)
and name_list_to_lam dbmap cexpr = function
| [] -> convert_check_expr dbmap cexpr
| x::xs -> Tc.Lam (name_list_to_lam (x::dbmap) cexpr xs)


let convert_data_param = function
| Ast.MultParam _ -> Tc.MultParam
| Ast.TypeParam _ -> Tc.TypeParam

let data_param_name = function
| Ast.MultParam name -> name
| Ast.TypeParam name -> name

let rec convert_cons_params dbmap = function
| Ast.Arr (mult, in_ty, out_ty) ->
  (convert_mult dbmap mult, convert_ty dbmap in_ty) :: convert_cons_params dbmap
| _ -> []

let convert_cons_ty_params dbmap ty =
  let rec ty_params' = function
  | Ast.Inst (to_inst, ty) ->
    Tc.Type (convert_ty dbmap ty) :: ty_params' to_inst
  | Ast.InstM (to_inst, mult) ->
    Tc.Mult (convert_mult dbmap mult) :: ty_params' to_inst
  | Ast.Forall _  -> raise ImpredicativeType
  | Ast.ForallM _ -> raise ImpredicativeType
  | _ -> []
  in
  let rec ty_params = function
  | Ast.Arr (_, _, out_ty) -> ty_params out_ty
```

```
  | Forall _  -> raise ImpredicativeType
  | ForallM _ -> raise ImpredicativeType
  | ty -> ty_params' ty
  in
  List.rev (ty_params ty)


let convert_cons_def dbmap (Ast.Cons (name, ty))
  = (name, convert_cons_params dbmap ty, convert_cons_ty_params dbmap ty)

let convert_def = function
  | Ast.LetDef (name, args, ty, cexpr) ->
    Tc.LetDef (name, convert_ty [] ty, name_list_to_lam [] cexpr args)
  | Ast.DataDef (name, dp_list, cd_list) ->
    Tc.DataDef ( name,
                 List.map convert_data_param dp_list,
                 let dbmap = List.rev (List.map data_param_name dp_list) in
                 List.map (convert_cons_def dbmap) cd_list
               )
  | Ast.LetDecl (name, ty) -> Tc.LetDecl (name, convert_ty [] ty)
let convert = List.map convert_def
```

8.2.5 lib/core/typecheck.ml

```
(* Authors: Ben, Jay *)
type global       = string
type dbindex      = int
type num_abstr    = int
type constr_index = int
type uniq_varname = int

type mult
  = One
  | Unr
  | MVar of dbindex
  | MTimes of mult * mult

type kind
```

```
  = KType
  | KMult
  | KUnknown of uniq_varname
  | KArr of kind * kind

type base_ty = BoolT | CharT | IntT

type ty
  = BaseT of base_ty
  | DataTy of global * param list
  | TVar of dbindex
  | Arr of mult * ty * ty
  | Forall of ty
  | ForallM of ty
and param
  = Mult of mult
  | Type of ty
type binop
  = Lt | Gt | Leq | Geq | Neq | Eq | Plus | Minus | Divide | Times | And | Or

type unop = Not | Neg

type check_expr
  = Lam of check_expr
  | Case of infer_expr * case_alt list
  | Infer of infer_expr
and infer_expr
  = DbIndex of dbindex
  | Global of global
  | Binop of binop
  | Unop of unop
  | Let of mult * ty * check_expr * infer_expr
  | App of infer_expr * check_expr
  | MApp of infer_expr * mult
  | TApp of infer_expr * ty
  | Construction of global
  | If of infer_expr * infer_expr * infer_expr
  | Ann of check_expr * ty
```

```
  | Int of int
  | Char of char
  | Bool of bool
and case_alt
  = Destructor of global * num_abstr * infer_expr
  | Wildcard of infer_expr

type cons_def = global * (mult * ty) list * param list

type data_param
  = MultParam
  | TypeParam

type data_def = global * data_param list * cons_def list

type def
  = LetDef of global * ty * check_expr
  | DataDef of data_def
  | LetDecl of global * ty

type env =
  {
    local_env  : ty list;
    global_env : (global * ty) list;
    kind_env   : (global * kind) list;
    data_env   : data_def list;
    lam_count   : int;
  }

type program = def list

module Sast = struct
  module Syntax = struct
    type sty =
    | BaseT of base_ty
    | DataSty of global * sty list
    | TVar of dbindex
    | Arr of sty * sty
```

```
  | Forall of sty

  type ast
  = Lam     of ast * sty * sty
  | TLam    of ast * sty
  | Case    of ast * sty * case_alt list * sty
  | DbIndex of dbindex * sty
  | Global  of global * sty
  | Binop   of binop * sty
  | Unop    of unop * sty
  | Let     of ast * sty * ast * sty
  | App     of ast * sty * ast * sty * sty
  | TApp    of ast * sty * sty
  | Construction
            of global * sty
  | If      of ast * ast * ast * sty
  | Int     of int
  | Char    of char
  | Bool    of bool
  and case_alt
  = Destructor of global * num_abstr * ast * sty
  | Wildcard   of ast * sty

  type cons_def = global * sty list
  type data_def = global * num_abstr * cons_def list

  type def
  = LetDef of global * sty * ast
  | DataDef of data_def
  | LetDecl of global * sty

  type program = def list
end
module S = Syntax


let rec shift_sty cutoff amount = function
| S.DataSty (global, stys) -> S.DataSty (global, List.map (shift_sty cutoff am
```

```
| S.TVar n -> S.TVar(if n < cutoff then n else n + amount)
| S.Arr (in_sty, out_sty) -> S.Arr (shift_sty cutoff amount in_sty, shift_sty
| S.Forall ty -> S.Forall (shift_sty (cutoff + 1) amount ty)
| t -> t
and shift cutoff amount = function
| S.Lam (ast, sty1, sty2) -> S.Lam (shift (cutoff + 1) amount ast, shift_sty c
| S.Case (ast, sty, calts, sty2) -> S.Case (shift cutoff amount ast, shift_sty
| S.DbIndex (n, ty) -> S.DbIndex ((if n < cutoff then n else n + amount), (shi
| S.Global (name, sty) -> S.Global(name, shift_sty cutoff amount sty)
| S.Binop (binop, sty) -> S.Binop (binop, shift_sty cutoff amount sty)
| S.Unop (unop, sty) -> S.Unop (unop, shift_sty cutoff amount sty)
| S.Let (in_ast, in_sty, out_ast, out_sty) ->
    (match in_sty with
    | S.Arr (_, _) ->
      S.Let (
              shift (cutoff + 1) amount in_ast,
              shift_sty cutoff amount in_sty,
              shift (cutoff + 1) amount out_ast,
              shift_sty (cutoff + 1) amount out_sty
            )
    | _ -> S.Let (
              shift (cutoff + 1) amount in_ast,
              shift_sty cutoff amount in_sty,
              shift (cutoff) amount out_ast,
              shift_sty (cutoff + 1) amount out_sty
    ))
| S.TApp (ast, app_sty, out_sty) -> S.TApp(shift cutoff amount ast, shift_sty
| S.Construction (name, sty) -> S.Construction (name, shift_sty cutoff amount
| S.If(ast1, ast2, ast3, sty) -> S.If(shift cutoff amount ast1, shift cutoff a
| t -> t
and shift_case_alt cutoff amount = function
| S.Destructor(name, num_abstr, ast, sty) -> S.Destructor(name, num_abstr, shi
| S.Wildcard(ast, sty) -> S.Wildcard(shift cutoff amount ast, shift_sty cutoff

let ty_to_sty ty =
  let rec tts cutoff = function
  | BaseT base_ty -> S.BaseT base_ty
  | DataTy (name, param_list) -> S.DataSty (name, List.concat @@ List.map (par
```

```
    | TVar dbindex -> S.TVar dbindex
    | Arr (_, in_ty, out_ty) -> S.Arr (tts cutoff in_ty, tts cutoff out_ty)
    | Forall ty -> S.Forall (tts (cutoff + 1) ty)
    | ForallM ty -> shift_sty cutoff (-1) (tts cutoff ty)
    and param_to_ty cutoff = function
    | Type ty -> [tts cutoff ty]
    | _ -> []
    in tts 0 ty

  let convert_ty_param = function
  | Type t -> [ty_to_sty t]
  | _ -> []
  let convert_cons_def ty_params (global, params, _) =
    let rec convert_params cutoff params = function
    | (TypeParam)::xs ->  convert_params (cutoff + 1) params xs
    | _::xs -> convert_params cutoff (List.map (shift_sty cutoff (-1)) params) x
    | [] -> params
    in
    let sparams = List.map ty_to_sty (List.map snd params) in
    global, convert_params 0 sparams ty_params
  let convert_data_def (name, params, cd_list) =
    let rec num_ty_params = function
    | (TypeParam)::xs -> 1 + num_ty_params xs
    | _::xs -> num_ty_params xs
    | [] -> 0 in
    name, num_ty_params params, List.map (convert_cons_def params) cd_list
  include S
end

exception NotImplemented
exception ExpectedAbs of ty
exception ExpectedArr of ty
exception ExpectedForall of ty
exception ExpectedForallM of ty
exception TypeMismatch of ty * ty
exception MultMismatch of mult * mult
exception VarNotFound of global
exception UnknownConstructor of global
```

```
exception ExpectedDataTy of ty
exception ArgumentLengthMismatch of num_abstr * num_abstr
exception CaseResultMismatch of ty
exception ParamMismatch of param * param
exception InternalTypechekerError

let rec string_of_mult = function
| One -> "One"
| Unr -> "Unr"
| MTimes (a, b) -> string_of_mult a ^ "*" ^ string_of_mult b
| MVar a -> "#" ^ string_of_int a

let rec string_of_ty = function
| BaseT (IntT) -> "Int"
| BaseT (BoolT) ->"Bool"
| BaseT (CharT) -> "Char"
| DataTy (ty, params) -> ty ^ " " ^ List.fold_left (fun s p -> s ^ " " ^ string_
| TVar t -> "#" ^ string_of_int t
| Arr (m, s, t) -> "(" ^ string_of_ty s ^ " -" ^ string_of_mult m ^ "> " ^ strin
| Forall (t) -> "@ (" ^ string_of_ty t ^ ")"
| ForallM (t) -> "# (" ^ string_of_ty t ^ ")"
and string_of_param = function
| Type ty -> string_of_ty ty
| Mult mult -> string_of_mult mult


let string_of_uenv =
  List.fold_left (fun s (i, m) -> s ^ string_of_int i ^ ": " ^ string_of_mult m

module KindChecker = struct

  exception UnknownDataType of global
  exception UnknownTypeParam of dbindex
  exception OccursCheck of constr_index * kind
  exception UnificationError of kind * kind

  let lookup_data_def name dd_list =
    let finder name_to_find (name, _, _) =
```

```
    name_to_find = name
  in
  (try
    List.find (finder name) dd_list
    with Not_found -> raise (UnknownDataType name))


type kenv =
{
  kind_env   : kind list;
  data_env   : data_def list;
}

module IntMap = Map.Make (Int)

let rec apply_subst s = function
| KUnknown i ->
  (match (IntMap.find_opt i s) with
  | Some k -> k
  | None -> KUnknown i)
| KArr (k1, k2) -> KArr (apply_subst s k1, apply_subst s k2)
| k -> k

let apply_subst_env s env = { env with kind_env = List.map (apply_subst s) env
let compose_subst s1 s2 = IntMap.union (fun _ x _ -> Some x) (IntMap.map (appl

let rec unify_kind k1 k2 =
  let rec fvs = function
  | KUnknown i -> [i]
  | KArr (k1, k2) -> fvs k1 @ fvs k2
  | _ -> []
  in
  let occurs_check i k =
    match List.find_opt (fun x -> x = i) (fvs k) with
    | Some _ -> raise (OccursCheck (i, k))
    | None -> ()
  in
  let bind i k =
```

```
    if (KUnknown i) = k then IntMap.empty else
    (occurs_check i k; IntMap.singleton i k)
  in
  match (k1, k2) with
| KUnknown i, k -> bind i k
| k, KUnknown i -> bind i k
| KArr (k1, k2), KArr (k3, k4) ->
  let s1 = unify_kind k1 k3 in
  let s2 = unify_kind (apply_subst s1 k2) (apply_subst s1 k4) in
  compose_subst s2 s1
| k1, k2 -> if k1 = k2 then IntMap.empty else raise (UnificationError (k1, k2)

let rec infer_kind env i = function
| BaseT _ -> KType, IntMap.empty, i

| DataTy (_, _) ->
  (* TODO: Infer kinds of params from use in each constructor *)
  KType, IntMap.empty, i

| TVar idx -> (
  try
    List.nth env.kind_env idx, IntMap.empty, i
  with _ -> raise (UnknownTypeParam idx)
)

| Arr (_, from_ty, to_ty) ->
  let k1, s1, i = infer_kind env i from_ty in
  let k2, s2, i = infer_kind (apply_subst_env s1 env) i to_ty in
  KArr (KArr (k1, KMult), k2), s2, i

| Forall ty ->
  let k, s, i' = infer_kind { env with kind_env = (KUnknown i)::env.kind_env }
  (KArr (apply_subst s (KUnknown i), k)), s, i'

| ForallM ty ->
  let k, s, i = infer_kind { env with kind_env = KMult::env.kind_env } i ty in
  (KArr (KMult, k)), s, i
and kind_param env i = function
```

```
  | Type ty -> infer_kind env i ty
  | Mult _ -> KMult, IntMap.empty, i
end

let cond c e = if c then () else raise e

let rec extract_data_env = function
| (LetDef _)::defs -> extract_data_env defs
| (DataDef data_def)::defs -> data_def :: extract_data_env defs
| _::defs -> extract_data_env defs
| _ -> []

let rec extract_global_env = function
| (LetDef (global, ty, _))::defs -> (global,ty) :: extract_global_env defs
| (DataDef _)::defs -> extract_global_env defs
| (LetDecl (global, ty))::defs -> (global, ty) :: extract_global_env defs
| _ -> []

let env_to_kenv env : KindChecker.kenv = { kind_env = []; data_env = env.data_en
let extend_env ty env = { env with local_env = ty :: env.local_env }

let rec lookup_uenv i = function
| (j, mult)::env -> if i = j then mult else lookup_uenv i env
| [] -> Unr

let lookup_constructor name { data_env; _ } =
  let rec add_abs c = function
  | TypeParam::xs -> Forall (add_abs c xs)
  | MultParam::xs -> ForallM (add_abs c xs)
  | [] -> c
  in
  let rec cons_ty' name ty_params = function
  | ((mult, in_ty)::xs) -> Arr (mult, in_ty, cons_ty' name ty_params xs)
  | [] -> DataTy (name, ty_params)
  in
  let cons_ty name params ty_params = add_abs (cons_ty' name ty_params params)
  in
  let cd_list =
```

```
    List.concat (List.map (
      fun (dname,dp_list,cd_list) ->
        List.map (fun
          (cname, params, ty_params) ->
            (cname, cons_ty dname params ty_params dp_list)) cd_list
    ) data_env)
  in
  try List.assoc name cd_list with _ -> raise (UnknownConstructor name)

let rec dec_uenv = function
| (i, mult)::env -> if i = 0 then dec_uenv env else (i-1, mult)::dec_uenv env
| [] -> []

let scale_usage mult = List.map (fun (x, m) -> (x, MTimes (mult, m)))

let rec union_with f l = function
| [] -> l
| (x, v1)::xs ->
  (match List.assoc_opt x l with
  | Some v2 -> (x, f v1 v2)::(union_with f (List.remove_assoc x l) xs)
  | None -> (x, v1)::(union_with f l xs))

let simp =
  let rec simp' = function
  | MTimes (a, One)  -> simp' a
  | MTimes (One, a)  -> simp' a
  | MTimes (Unr, _)  -> Unr
  | MTimes (_, Unr)  -> Unr
  | a -> a
  in
  let rec simp = function
  | MTimes (a, b)    -> simp' (MTimes (simp a, simp b))
  | a                -> simp' a
  in simp

let add_usage x y = union_with (fun _ _ -> Unr) x y
let multiply_usage x y = union_with (fun a b -> MTimes (a, b)) x y
```

```
let rec shift_mult cutoff amount = function
| MTimes (a, b) -> MTimes (shift_mult cutoff amount a, shift_mult cutoff amount
| MVar n -> MVar (if n < cutoff then n else n + amount)
| m -> m
let rec shift_ty cutoff amount = function
| Arr (mult, in_ty, out_ty) ->
  Arr ( shift_mult cutoff amount mult,
        shift_ty cutoff amount in_ty,
        shift_ty cutoff amount out_ty
      )
| Forall ty -> Forall (shift_ty (cutoff + 1) amount ty)
| ForallM ty -> ForallM (shift_ty (cutoff + 1) amount ty)
| TVar n -> TVar (if n < cutoff then n else n + amount)
| DataTy (name, params) -> DataTy (name, List.map (shift_param cutoff amount) pa
| t -> t
and shift_param cutoff amount = function
| Type ty -> Type (shift_ty cutoff amount ty)
| Mult mult -> Mult (shift_mult cutoff amount mult)

let rec subst_mult_mult subst mult = match (subst, mult) with
| (i, m), MVar j -> if i = j then m else MVar j
| s, MTimes (a, b) -> MTimes (subst_mult_mult s a, subst_mult_mult s b)
| _, m -> m

let rec subst_mult_ty subst ty = match (subst, ty) with
| s, Arr (m, t, t') -> Arr (subst_mult_mult s m, subst_mult_ty s t, subst_mult_t
| (i, m), Forall t -> Forall (subst_mult_ty (i+1, shift_mult 0 1 m) t)
| (i, m), ForallM t -> ForallM (subst_mult_ty (i+1, shift_mult 0 1 m) t)
| s, DataTy (name, params) -> DataTy (name, List.map (subst_mult_param s) params
| _, t -> t
and subst_mult_param s = function
| Type ty -> Type (subst_mult_ty s ty)
| Mult mult -> Mult (subst_mult_mult s mult)

let rec subst_ty_ty subst ty = match (subst, ty) with
| (i, t), TVar j -> if i = j then t else TVar j
| s, Arr (m, t, t') -> Arr (m, subst_ty_ty s t, subst_ty_ty s t')
```

```
| (i, t), Forall t' -> Forall (subst_ty_ty (i+1, shift_ty 0 1 t) t')
| (i, t), ForallM t' -> ForallM (subst_ty_ty (i+1, shift_ty 0 1 t) t')
| s, DataTy (name, params) -> DataTy (name, List.map (subst_ty_param s) params)
| _, t -> t
and subst_ty_param s = function
| Type ty -> Type (subst_ty_ty s ty)
| m -> m

let subst_param_ty (i, p) ty = match p with
| Type ty'-> subst_ty_ty (i, ty') ty
| Mult mult -> subst_mult_ty (i, mult) ty
let subst_param_mult (i, p) mult = match p with
| Mult mult' -> subst_mult_mult (i, mult') mult
| _ -> mult
let subst_param_list_ty substs ty =
  List.fold_left (fun ty s -> subst_param_ty s ty) ty substs
let subst_param_list_mult substs mult =
  List.fold_left (fun mult s -> subst_param_mult s mult) mult substs

let lookup_constructor_def name { data_env; _ } =
  let cd_aggregate_list =
    List.concat (List.map (
      fun (_,_,cd_list) -> List.map
        (fun (name, params, ty_params) -> name, (params, ty_params)) cd_list
    ) data_env)
  in
  try List.assoc name cd_aggregate_list with _ -> raise (UnknownConstructor name

let rec unify_left ty ty' = match (ty, ty') with
| Arr (m, in_ty, out_ty), Arr (m', in_ty', out_ty') ->
  let s_in = unify_left in_ty in_ty' in
  let s_out = unify_left out_ty out_ty' in
  cond (m = m') (MultMismatch (m, m'));
  union_with (fun t t' ->
    cond (t = t') (TypeMismatch (t, t')); t'
  ) s_in s_out
| DataTy (name, params), DataTy (name', params') ->
  cond (name = name') (TypeMismatch (ty, ty'));
```

```
  let unify_param (p, p') = (match (p, p') with
  | Type t, Type t' -> unify_left t t'
  | Mult m, Mult m' -> cond (m = m') (MultMismatch (m, m')); []
  | _, _ -> raise @@ ParamMismatch (p, p'))
  in
  let unify_params s p = (unify_param p) @ s
  in
  List.fold_left unify_params [] (List.combine params params')
| TVar n, _ -> [n, ty']
| _, TVar _ -> raise @@ TypeMismatch (ty, ty')
| _ -> cond (ty = ty') (TypeMismatch (ty, ty')); []

module S = Sast

let rec is_polymorphic = function
| One -> false
| Unr -> false
| MVar _ -> true
| MTimes (a, b) -> is_polymorphic a || is_polymorphic b

let check_constraint actual_mult expected_mult =
  let rec leq a b = match (a, b) with
  | MTimes (MTimes (a, b), c), MTimes (MTimes (a', b'), c') -> assoc_leq a b c a
  | MTimes (a,  MTimes (b, c)), MTimes (MTimes (a', b'), c') -> assoc_leq a b c
  | MTimes (MTimes (a, b), c), MTimes (a',  MTimes (b', c')) -> assoc_leq a b c
  | MTimes (a,  MTimes (b, c)), MTimes (a',  MTimes (b', c')) -> assoc_leq a b c
  | MTimes (((MVar _) as a), ((MVar _) as b)), MTimes (((MVar _) as a'), ((MVar
  | MTimes (a, b), ((MVar _) as c) -> (leq a c && leq b c)
  | a, MTimes (b, MTimes (c, d)) -> leq a (MTimes (b, c)) || leq a (MTimes (b, d
  | a, MTimes (MTimes (b, c), d) -> leq a (MTimes (b, c)) || leq a (MTimes (b, d
  | a, MTimes (b, c) -> leq a b || leq a c
  | (MVar a, MVar b) -> a = b
  | (_, Unr) -> true
  | (Unr, _) -> false
  | (One, One) -> true
  | (One, _) -> true
  | (_, One) -> false
  and assoc_leq a b c a' b' c' =
```

```
    (leq a a' && leq b b' && leq c c') ||
    (leq a a' && leq b c' && leq c b') ||
    (leq a b' && leq b c' && leq c a') ||
    (leq a b' && leq b a' && leq c c') ||
    (leq a c' && leq b b' && leq c a') ||
    (leq a c' && leq b a' && leq c b')
  in
  let a, b = simp actual_mult, simp expected_mult in
  if leq a b then
    ()
  else
    raise @@ MultMismatch (a, b)

let rec check env ty = function
| Lam cexpr ->
  (match ty with
    | Arr (expected_mult, in_ty, out_ty) ->
      let env = { env with lam_count = env.lam_count + 1 } in
      let uenv, sexpr
        = check (extend_env in_ty env) out_ty cexpr
      in
      let actual_mult = lookup_uenv 0 uenv in
      check_constraint actual_mult expected_mult;
      dec_uenv uenv, S.Lam (sexpr, S.ty_to_sty in_ty, S.ty_to_sty out_ty)
    | Forall (ty') ->
      let uenv, sexpr = check env ty' cexpr in
      dec_uenv uenv, S.TLam (sexpr, S.ty_to_sty ty)
    | ForallM (ty') ->
      let uenv, sexpr = check env ty' cexpr in
      dec_uenv uenv, S.shift 0 (-1) sexpr
    | ty -> raise @@ ExpectedAbs ty
  )
| Case (iexpr, calts) ->
  let ty_scrut, uenv,  sscrut = infer env iexpr in
  let infer_calt = function
    | Destructor (name, len, rhs) -> (
      let range n = List.init n (fun x -> x) in
      (* Get parameters and type paramaters from constructor *)
```

```
let calt_params, _
  = lookup_constructor_def name env in
let scrut_ty_params = (match ty_scrut with
| DataTy (_, params) -> params
| _ -> raise @@ ExpectedDataTy ty_scrut)
in
(* Refine type of parameters according to the scrutinee *)
let calt_params
  = List.map (
      fun (m, t) ->
        let substs = List.combine (range (List.length scrut_ty_params)) (L
        in
        subst_param_list_mult substs m,
        subst_param_list_ty substs t) (List.rev calt_params)
in

(* Infer type of rhs given the params introduced by the case alt *)
let mults = List.map fst calt_params in
let types = List.map snd calt_params in
let actual_ty, uenv, srhs
  = infer ({ env with local_env = types @ env.local_env }) rhs
in
(* Check to make sure that the number of arguments
   bound matches the number of arguments given in
   the definition of the constructor *)
cond (len = List.length types) (ArgumentLengthMismatch (len, List.length t
(* Unify expected type with actual (GADT) *)
let substs = unify_left ty actual_ty in
let subst_list_ty_ty s t = List.fold_left (fun t s -> subst_ty_ty s t) t s
in
let expected_ty
  = subst_list_ty_ty substs ty in
(* Check equality of rhs to the expected type *)
cond (expected_ty = actual_ty) (TypeMismatch (expected_ty, actual_ty));
(* Check multiplicity constraints based on usage env in rhs of case alt *)
let check_constr (idx, expected_mult) =
  let actual_mult = lookup_uenv idx uenv in
  check_constraint actual_mult expected_mult
```

```
      in
      List.iter check_constr (List.combine (range len) mults);
      (* Remove all bound variables in usage environment *)
      let uenv = List.fold_left (fun u _ -> dec_uenv u) uenv (range len) in
      uenv, S.Destructor (name, len, srhs, S.ty_to_sty actual_ty)
    )
    | Wildcard rhs ->
      let uenv, srhs = check env ty (Infer rhs) in
      uenv, S.Wildcard (srhs, S.ty_to_sty ty)
  in
  (* Add up all constraints and usage environments from each case alt *)
  let rhs_res = List.map infer_calt calts in
  let uenv_rhs = List.concat (List.map fst rhs_res) in
  let scalts = List.map snd rhs_res in
  add_usage uenv uenv_rhs, S.Case (sscrut, S.ty_to_sty ty_scrut, scalts, S.ty_to
| Infer iexpr ->
  let ty', uenv, sexpr
    = infer env iexpr
  in
  cond (ty = ty') (TypeMismatch (ty, ty'));
  uenv, sexpr
and infer env = function
| DbIndex idx -> let ty = List.nth env.local_env idx in
  ty, [(idx, One)], S.DbIndex (idx, S.ty_to_sty ty)

| Global global ->
  (
    try
      let ty = List.assoc global env.global_env in
      ty, [], S.Global (global, S.ty_to_sty ty)
    with _ -> raise (VarNotFound global)
  )

| Binop binop ->
  (match binop with
  | And | Or ->
    let ty = Arr (One, BaseT BoolT, Arr (One, BaseT BoolT, BaseT BoolT)) in
    ty, [], S.Binop (binop, S.ty_to_sty ty)
```

```
  | Plus | Minus | Times | Divide ->
    let ty = Arr (One, BaseT IntT, Arr (One, BaseT IntT, BaseT IntT)) in
    ty, [], S.Binop (binop, S.ty_to_sty ty)
  | _ ->
    let ty = Arr (One, BaseT IntT, Arr (One, BaseT IntT, BaseT BoolT)) in
    ty, [], S.Binop (binop, S.ty_to_sty ty))
| Unop unop ->
  (match unop with
  | Not ->
    let ty = Arr (One, BaseT BoolT, BaseT BoolT) in
    ty, [], S.Unop (unop, S.ty_to_sty ty)
  | Neg ->
    let ty = Arr (One, BaseT IntT, BaseT IntT) in
    ty , [], S.Unop (unop, S.ty_to_sty ty))

| Let (mult, ty, cexpr, iexpr) ->
  (match ty with
  | Arr (_, _, _) ->
    let uenv1, sexpr1
      = check (extend_env ty env) ty cexpr
    in
    let ty', uenv2, sexpr2
      = infer (extend_env ty env) iexpr
    in
    let actual_mult = lookup_uenv 0 uenv2 in
    check_constraint actual_mult mult;
    ty', add_usage (dec_uenv uenv2) (scale_usage mult (dec_uenv uenv1)), S.Let (
  | _ ->
    let uenv1, sexpr1
      = check env ty cexpr
    in
    let ty', uenv2, sexpr2
      = infer (extend_env ty env) iexpr
    in
    let actual_mult = lookup_uenv 0 uenv2 in
    check_constraint actual_mult mult;
    ty', add_usage (dec_uenv uenv2) (scale_usage mult uenv1), S.Let (sexpr1, S.t
  )
```

```
| App (iexpr, cexpr) ->
  let ty, uenv1, sexpr1 = infer env iexpr in
  (match ty with
  | Arr (mult, in_ty, out_ty) ->
    let uenv2, sexpr2 = check env in_ty cexpr in
    out_ty, add_usage uenv1 (scale_usage mult uenv2), S.App (sexpr1, S.ty_to_sty
  | t -> raise @@ ExpectedArr t)

| MApp (iexpr, mult) ->
  let lhs_ty, uenv, sexpr1 = infer env iexpr in
  let mult = shift_mult 0 (-env.lam_count) mult in
  (match lhs_ty with
  | ForallM ty ->
    shift_ty 0 (-1) (subst_mult_ty (0, shift_mult 0 1 mult) ty), uenv, sexpr1
  | t -> raise @@ ExpectedForallM t)

| TApp (iexpr, ty) ->
  let lhs_ty, uenv, sexpr = infer env iexpr in
  let ty = shift_ty 0 (-env.lam_count) ty in
  (match lhs_ty with
  | Forall ty' ->
    let out_ty = shift_ty 0 (-1) (subst_ty_ty (0, shift_ty 0 1 ty) ty') in
    out_ty, uenv, S.TApp (sexpr, S.ty_to_sty ty, S.ty_to_sty out_ty)
  | t -> raise @@ ExpectedForall t)

| Construction name ->
  let ty = lookup_constructor name env in
  ty, [], S.Construction (name, S.ty_to_sty ty)

| If (iexpr_0, iexpr_1, iexpr_2) ->
  let ty, uenv, sexpr0 = infer env iexpr_0 in
  cond (ty = BaseT BoolT) (TypeMismatch (ty, (BaseT BoolT)));
  let ty1, uenv1, sexpr1 = infer env iexpr_1 in
  let ty2, uenv2, sexpr2 = infer env iexpr_2 in
  cond (ty1 = ty2) (TypeMismatch (ty1, ty2));
  ty1, add_usage uenv (multiply_usage uenv1 uenv2), S.If (sexpr0, sexpr1, sexpr2
```

```
| Ann (cexpr, ty) ->
  let uenv, sexpr = check env ty cexpr in
  ty, uenv, sexpr

| Int i -> BaseT IntT, [], S.Int i

| Char c -> BaseT CharT, [], S.Char c

| Bool b -> BaseT BoolT, [], S.Bool b


let check_def env = function
| LetDef (name, ty, cexpr) ->
  let _ = KindChecker.infer_kind (env_to_kenv env) 0 ty in
  let _, sexpr = check env ty cexpr in
  S.LetDef (name, S.ty_to_sty ty, sexpr)
| DataDef dd -> S.DataDef (S.convert_data_def dd)
| LetDecl (name, ty) -> S.LetDecl (name, S.ty_to_sty ty)
let check_prog prog =
  try
    let data_env = extract_data_env prog in
    let global_env = extract_global_env prog in
    let env = { local_env = [];
                global_env = global_env;
                data_env = data_env;
                kind_env = [];
                lam_count = 0;
              } in
    List.map (check_def env) prog
  with
  | MultMismatch (a, b) as self ->
    Printf.eprintf "%s\n" ("Multiplicity Mismatch: " ^ string_of_mult a ^ " > "
    raise self
  | TypeMismatch (a, b) as self ->
    Printf.eprintf "%s\n" ("Type Mismatch: " ^ string_of_ty a ^ " /= " ^ string_
    raise self
```

8.2.6 lib/mono/conversion.ml

```
(* Authors: Ben, Jay *)
module Sast = Core.Typecheck.Sast
module C = Core.Typecheck
module S = Sast

open Mast
exception NotImplemented
exception MonoError


let rec convert_sty = function
| S.BaseT C.BoolT -> BoolT
| S.BaseT C.IntT -> IntT
| S.BaseT C.CharT -> CharT
| S.DataSty (global, _) -> DataTy global
| S.TVar _ -> BoxT
| Forall sty -> convert_sty sty
| S.Arr (in_sty, out_sty) -> Arr (convert_sty in_sty, convert_sty out_sty)

let rec string_of_ty = function
| IntT -> "Int"
| CharT -> "Char"
| BoolT -> "Bool"
| DataTy name -> name
| TVar dbindex -> "#" ^ string_of_int dbindex
| BoxT -> "Box"
| Arr (in_ty, out_ty) -> string_of_ty in_ty ^ " -> " ^ string_of_ty out_ty

let reconcile expr exp_ty gen_ty =
  match (exp_ty, gen_ty) with
  | (BoxT, BoxT) ->  BoxT, expr
  | (_, BoxT) -> exp_ty, Unbox (expr, exp_ty)
  | (BoxT, _) -> exp_ty, Box (expr, gen_ty)
  | _ -> gen_ty, expr
```

```
let convert_sexpr datadefs =
  let rec convert_sexpr tys = function
  | S.Lam (sexpr, in_sty, out_sty) ->
    let in_ty = convert_sty in_sty in
    let gen_out_ty, expr = convert_sexpr (in_ty::tys) sexpr in
    let exp_out_ty = convert_sty out_sty in
    let out_ty, expr = reconcile expr exp_out_ty gen_out_ty in
    Arr (in_ty, out_ty), Lam (expr, in_ty, out_ty)

  | S.TLam (sexpr, sty) ->
    let exp_ty = convert_sty sty in
    let gen_ty, expr = convert_sexpr tys (S.shift 0 (-1) sexpr) in
    reconcile expr exp_ty gen_ty

  | S.Case (sscrut, scrut_sty, ca_list, out_sty) ->
    let exp_scrut_ty = convert_sty scrut_sty in
    let gen_scrut_ty, scrut = convert_sexpr tys sscrut in
    let scrut_ty, scrut = reconcile scrut exp_scrut_ty gen_scrut_ty in
    let out_ty, calts = List.fold_left_map (convert_calt tys) (convert_sty out_s
    out_ty, Case (scrut, scrut_ty, calts, out_ty)

  | S.DbIndex (idx, sty) ->
    (match List.nth_opt tys idx with
    | Some ty ->
      reconcile (DbIndex (idx, ty)) (convert_sty sty) ty
    | None -> convert_sty sty, DbIndex (idx, convert_sty sty))
  | S.Global (global, sty) ->
    convert_sty sty, Global (global, convert_sty sty)
  | S.Binop (op, sty) ->
    convert_sty sty, Binop (op, convert_sty sty)
  | S.Unop (op, sty) ->
    convert_sty sty, Unop (op, convert_sty sty)
  | S.Let (sexpr1, sty1, sexpr2, sty2) ->
    let exp_ty1, exp_ty2 = convert_sty sty1, convert_sty sty2 in
    let gen_ty1, expr1 = convert_sexpr (exp_ty1::tys) sexpr1 in
    let gen_ty2, expr2 = convert_sexpr (exp_ty1::tys) sexpr2 in
    let ty1, expr1 = reconcile expr1 exp_ty1 gen_ty1 in
    let ty2, expr2 = reconcile expr2 exp_ty2 gen_ty2 in
```

```
  ty2, Let (expr1, ty1, expr2, ty2)

| S.App (sexpr1, _, sexpr2, in_sty, out_sty) ->
  let gen_arr_ty, expr1 = convert_sexpr tys sexpr1 in
  let _, expr2 = convert_sexpr tys sexpr2 in
  let exp_in_ty = convert_sty in_sty in
  let exp_out_ty = convert_sty out_sty in
  (match gen_arr_ty with
  | Arr (gen_in_ty, gen_out_ty) ->
      let ty2, expr2 = reconcile expr2 gen_in_ty exp_in_ty in
      reconcile (App (expr1, gen_arr_ty, expr2, ty2, gen_out_ty)) exp_out_ty g
  | _ -> raise MonoError
  )
| S.TApp (sexpr, _, sty) ->
  let exp_ty = convert_sty sty in
  let gen_ty, expr = convert_sexpr tys sexpr in
  reconcile expr exp_ty gen_ty
| S.Construction (global, sty) ->
  convert_sty sty, Construction (global, convert_sty sty)
| S.If (sexpr1, sexpr2, sexpr3, out_sty) ->
  let exp_out_ty = convert_sty out_sty in
  let ty1, expr1 = convert_sexpr tys sexpr1 in
  let _, expr1 = reconcile expr1 BoolT ty1 in
  let ty2, expr2 = convert_sexpr tys sexpr2 in
  let ty2, expr2 = reconcile expr2 exp_out_ty ty2 in
  let ty3, expr3 = convert_sexpr tys sexpr3 in
  let out_ty, expr3 = reconcile expr3 ty2 ty3 in
  out_ty, If(expr1, expr2, expr3, exp_out_ty)
| S.Int i -> IntT, Int i
| S.Bool b -> BoolT, Bool b
| S.Char c -> CharT, Char c
and convert_calt tys exp_ty = function
| S.Destructor (name, num_abstr, sexpr, _) ->
  let all_cons = List.concat (List.map snd datadefs) in
  let tys' = List.assoc name all_cons in
  let gen_ty, expr = convert_sexpr (List.rev tys' @ tys) sexpr in
  let ty, expr = reconcile expr exp_ty gen_ty in
  ty, Destructor (name, num_abstr, expr, ty)
```

```
  | S.Wildcard (sexpr, _) ->
    let gen_ty, expr = convert_sexpr tys sexpr in
    let ty, expr = reconcile expr exp_ty gen_ty in
    ty, Wildcard (expr, ty)
  in convert_sexpr []

let convert_prog sprog =
  let initial_sprog = {
    letdefs = [];
    main = Int 0;
    datadefs = [];
    decls = [];
  } in
  let rec datadefs = function
  | (S.DataDef (global, _, cd_list))::xs ->
      let datadef =
        global, List.map (fun (name, l) -> name, List.map convert_sty l) cd_list
      in datadef :: datadefs xs
  | _::xs -> datadefs xs
  | [] -> []
  in
  let datadefs = datadefs sprog in
  let to_prog mprog = function
  | S.LetDef (global, _, sexpr) ->
    if global = "main" then
      {mprog with main = snd @@ convert_sexpr datadefs sexpr}
    else
      let ty, expr = convert_sexpr datadefs sexpr in
      {mprog with letdefs = ((global, ty, expr)) :: mprog.letdefs }
  | S.DataDef (global, _, cd_list) ->
    let datadef =
      global, List.map (fun (name, l) -> name, List.map convert_sty l) cd_list
    in
    {mprog with datadefs = datadef::mprog.datadefs }
  | S.LetDecl (global, sty) ->
    let decl = (global, convert_sty sty) in
    { mprog with decls = decl::mprog.decls }
```

```
  in
  List.fold_left to_prog initial_sprog sprog
```

8.2.7 lib/mono/mast.ml

```
(* Authors: Ben *)
module Tc = Core.Typecheck
type binop = Tc.binop
type unop = Tc.unop
type global = string
type dbindex = int
type num_abstr = int

type mty =
  | IntT
  | CharT
  | BoolT
  | DataTy of global
  | TVar of dbindex
  | BoxT
  | Arr of mty * mty

type mexpr
  = Lam     of mexpr * mty * mty
  | Case    of mexpr * mty * case_alt list * mty
  | DbIndex of dbindex * mty
  | Global  of global * mty
  | Binop   of binop * mty
  | Unop    of unop * mty
  | Let     of mexpr * mty * mexpr * mty
  | App     of mexpr * mty * mexpr * mty * mty
  | Construction
            of global * mty
  | Box     of mexpr * mty
  | Unbox   of mexpr * mty
  | If      of mexpr * mexpr * mexpr * mty
  | Int     of int
  | Char    of char
```

```
  | Bool    of bool
and case_alt
  = Destructor of global * num_abstr * mexpr * mty
  | Wildcard   of mexpr * mty

type cons_def = global * mty list
type data_def = global * cons_def list

type program = {
  main : mexpr;
  letdefs : (global * mty * mexpr) list;
  datadefs: data_def list;
  decls: (global * mty) list;
}
```

8.2.8 lib/closure/conversion.ml

```
(* Authors: Ben, Jay *)
module M = Mono.Mast

open Cast
exception NotImplemented
exception ClosureError
type partial_prog = {
  funs: fundef list;
}
module Tc = Core.Typecheck
let (<<<) f g x = f (g x)

let string_of_ty = function
| CIntT -> "Int"
| CBoolT -> "Bool"
| CharT -> "Char"
| CDataTy cname -> cname
| CClosT -> "Clos"
| BoxT -> "Box"

module SM = Map.Make (String)
```

```
let vars = ref SM.empty
let unique_name name =
  match SM.find_opt name !vars with
  | Some i -> vars := SM.add name (Random.int64 (Int64.max_int)) !vars; name ^ I
  | None -> vars := SM.add name (Random.int64 (Int64.max_int)) !vars; name
let convert_mty = function
| M.IntT -> CIntT
| M.CharT -> CharT
| M.BoolT -> CBoolT
| M.DataTy name -> CDataTy name
| M.BoxT -> BoxT
| M.Arr _ -> CClosT
| _ -> raise ClosureError

let beta_reduce mexpr1 mexpr2 =
  let rec shift cutoff amount = function
  | M.Lam (mexpr, in_mty, out_mty) ->
    M.Lam (shift (cutoff + 1) amount mexpr, in_mty, out_mty)
  | M.DbIndex(idx, ty) as self ->
    if idx < cutoff then self else M.DbIndex(idx + amount, ty)
  | M.Case (mscrut, mty, mcalts, out_mty) ->
    M.Case (shift cutoff amount mscrut, mty, List.map (shift_case cutoff amount)
  | M.Let (mexpr1, mty1, mexpr2, mty2) ->
    (match mty1 with
    | M.Arr (_, _) ->
      M.Let (shift (cutoff + 1) amount mexpr1, mty1, shift (cutoff + 1) amount m
    | _ ->
      M.Let (shift cutoff amount mexpr1, mty1, shift (cutoff + 1) amount mexpr2,
    )

  | M.Box (mexpr, mty) ->
    M.Box (shift cutoff amount mexpr, mty)
  | M.Unbox (mexpr, mty) ->
    M.Unbox (shift cutoff amount mexpr, mty)
  | M.If (mexpr1, mexpr2, mexpr3, mty) ->
    M.If (shift cutoff amount mexpr1, shift cutoff amount mexpr2, shift cutoff a
  | M.App (mexpr1, mty1, mexpr2, mty2, out_mty) ->
    M.App (shift cutoff amount mexpr1, mty1, shift cutoff amount mexpr2, mty2, o
```

```
| a -> a
and shift_case cutoff amount = function
| M.Destructor (name, num_abstr, mexpr, out_mty) ->
  M.Destructor (name, num_abstr, shift (cutoff + num_abstr) amount mexpr, out_
| M.Wildcard (mexpr, ty) ->
  M.Wildcard (shift cutoff amount mexpr, ty)
in
let rec subst expr idx = function
| M.Lam (mexpr, in_mty, out_mty) ->
  M.Lam (subst (shift 0 1 expr) (idx + 1) mexpr, in_mty, out_mty)
| M.DbIndex(i, _) as self ->
  if i == idx then expr else self
| M.Case (mscrut, mty, mcalts, out_mty) ->
  M.Case (subst expr idx mscrut, mty, List.map (subst_case expr idx) mcalts, o
| M.Let (mexpr1, mty1, mexpr2, mty2) ->
  (match mty1 with
  | M.Arr (_, _) ->
    M.Let (subst (shift 0 1 expr) (idx + 1) mexpr1, mty1, subst (shift 0 1 exp
  | _ ->
    M.Let (subst expr idx mexpr1, mty1, subst (shift 0 1 expr) (idx + 1) mexpr
| M.Box (mexpr, mty) ->
  M.Box (subst expr idx mexpr, mty)
| M.Unbox (mexpr, mty) ->
  M.Unbox (subst expr idx mexpr, mty)
| M.If (mexpr1, mexpr2, mexpr3, mty) ->
  M.If (subst expr idx mexpr1, subst expr idx mexpr2, subst expr idx mexpr3, m
| M.App (mexpr1, mty1, mexpr2, mty2, out_mty) ->
  M.App (subst expr idx mexpr1, mty1, subst expr idx mexpr2, mty2, out_mty)
| a -> a
and subst_case expr idx = function
| M.Destructor (name, num_abstr, mexpr, out_mty) ->
  M.Destructor (name, num_abstr, subst (shift 0 num_abstr expr) (idx + num_abs
| M.Wildcard (mexpr, ty) ->
  M.Wildcard (subst expr idx mexpr, ty)
in
shift 0 (-1) (subst (shift 0 1 mexpr2) 0 mexpr1)
```

```
module T = struct
  type t = (cindex * cty)
  let compare : t -> t -> int = fun x y -> compare (fst x) (fst y)
end
module S = Set.Make (T)
let free_vars =
  let dec_by c =
    S.filter_map (fun (idx,ty) -> if idx - c < 0 then None else Some (idx-c, ty)
  in
  let rec free_vars = function
    | M.Lam (mexpr, _, _) -> dec_by 1 @@ free_vars mexpr
    | M.Case (mexpr, _, calt_list, _) ->
      S.union (free_vars mexpr)
        @@ List.fold_left S.union S.empty (List.map free_vars_calt calt_list)
    | M.DbIndex (idx, mty) -> S.singleton (idx, convert_mty mty)
    | M.Let (mexpr1, mty1, mexpr2, _) ->
      (match mty1 with
      | M.Arr (_, _) -> S.union (dec_by 1 @@ free_vars mexpr1) (dec_by 1 @@ free
      | _ -> S.union (free_vars mexpr1) (dec_by 1 @@ free_vars mexpr2))
    | M.App (mexpr1, _, mexpr2, _, _) ->
      S.union (free_vars mexpr1) (free_vars mexpr2)
    | M.Box (mexpr, _) -> free_vars mexpr
    | M.Unbox (mexpr, _) -> free_vars mexpr
    | M.If (mexpr1, mexpr2, mexpr3, _) ->
      S.union (S.union (free_vars mexpr1)
      (free_vars mexpr2)) @@ free_vars mexpr3
    | _ -> S.empty
    and free_vars_calt = function
    | M.Destructor (_, num_abstr, mexpr, _) ->
      dec_by num_abstr @@ free_vars mexpr
    | M.Wildcard (mexpr, _) -> free_vars mexpr
  in S.elements <<< free_vars

let string_of_binop = function
| Tc.Plus -> "plus"
| Tc.Minus -> "minus"
| Tc.Times -> "times"
```

```
| Tc.Divide -> "divide"
| Tc.Geq -> "geq"
| Tc.Gt -> "gt"
| Tc.Eq -> "eq"
| Tc.Leq -> "leq"
| Tc.Lt -> "lt"
| Tc.Neq -> "neq"
| Tc.And -> "and"
| Tc.Or -> "or"
let string_of_unop = function
| Tc.Not -> "not"
| Tc.Neg -> "neg"

let funs = ref []
let convert_mexpr name (prog : M.program) expr =
 let rec convert = function
  | M.Lam (mexpr, _in_mty, out_mty) as self ->
    let name = unique_name ("fn_" ^ name) in
    let expr = convert mexpr in
    let fvs = free_vars self in
    let args = List.map (fun (i,t) -> CArg (i, t)) @@ fvs in
    let fv_tys = List.map snd fvs in
    let global = name, List.length fvs + 1, expr, convert_mty out_mty in
    funs := global::!funs;
    CClos (name, args, fv_tys)
  | M.Case (mscrut, scrut_mty, ca_list, out_mty) ->
    let scrut = convert mscrut in
    let ca_list = List.map convert_calt ca_list in
    CCase (scrut, convert_mty scrut_mty, ca_list, convert_mty out_mty)
  | M.DbIndex (idx, ty) ->
    CArg (idx, convert_mty ty)
  | M.Global (name, mty) -> (
    let ty = convert_mty mty in
    match List.assoc_opt name prog.decls with
    | Some _ -> CClos ("__" ^ name ^ "__", [], [])
    | None -> CApp (CClos (name, [], []), CClosT, CInt 0, CIntT, ty))
  | M.App (mexpr1, mty1, mexpr2, mty2, out_mty) ->
    let expr1 = convert mexpr1 in
```

```
  let expr2 = convert mexpr2 in
  CApp (expr1, convert_mty mty1, expr2, convert_mty mty2, convert_mty out_mty)
| M.Box (mexpr, mty) -> Box (convert mexpr, convert_mty mty)
| M.Unbox (mexpr, mty) -> Unbox (convert mexpr, convert_mty mty)
| M.Int i -> CInt i
| M.Bool i -> CBool i
| M.Char i -> CChar i
| M.Construction (cname, _mty) -> (

  let cons_tys = List.map convert_mty
    @@ List.assoc cname @@ List.concat (List.map snd prog.datadefs) in
  let dty = CDataTy (fst @@ List.find (fun (_, d) -> List.mem cname (List.map
  let funs' = List.map (fun (name,tyl,expr,ty) -> name, (tyl, expr, ty)) !funs

  (match List.assoc_opt cname funs' with
  | Some _ -> CClos (cname, [], [])
  | None ->
    let mk_globals =
      let rec mk i arg_tys = function
      | [] ->
        let args = List.rev @@ List.mapi (fun i ty -> CArg (i, ty)) arg_tys in
        CConstruction (cname, args, dty), dty
      | (ty::tys) ->
        let uname = unique_name cname in
        let expr, out_ty = mk (i+1) (ty::arg_tys) tys in
        let len = List.length (ty::tys) in
        funs := (uname, len, expr, out_ty)::!funs;
        CClos (uname, List.mapi (fun i ty -> CArg (i, ty)) arg_tys, arg_tys),
      in
      fst <<< mk 0 []
    in
    mk_globals cons_tys
))
| M.Binop (binop, _ty) ->
  CClos ("__prim__" ^ string_of_binop binop , [], [])
| M.Unop (unop, _ty) ->
  CClos ("__prim__" ^ string_of_unop unop, [], [])
| M.If (mexpr1, mexpr2, mexpr3, out_mty) ->
```

```
  let expr1 = convert mexpr1 in
  let expr2 = convert mexpr2 in
  let expr3 = convert mexpr3 in
  CIf (expr1, expr2, expr3, convert_mty out_mty)
| M.Let (mexpr1, mty1, mexpr2, mty2) ->
  (* let x = e in y ~> (\x. y) e *)
  (* let f x = e in y ~> (\f. y) (\f. \x. e) *)
  let _ = unique_name name in
  let name1 = unique_name name in
  let name2 = unique_name name in
  let expr2 = convert mexpr2 in
  let expr1 = (match mty1 with
  | M.Arr (_, _) ->
    let mexpr1 = beta_reduce mexpr1 (M.Global (name1, mty1)) in
    let fvs = free_vars mexpr1 in
    let expr1 = convert mexpr1 in
    let args = List.map (fun (i,t) -> CArg (i, t)) @@ fvs in
    let fv_tys = List.map snd fvs in
    let fn = name1, List.length fvs + 1, expr1, CClosT in
    funs := fn::!funs;
    CApp (CClos (name1, args, fv_tys), CClosT, CInt 0, CIntT, CClosT)
  | _ -> convert mexpr1) in
  let fvs = free_vars @@ M.Lam (mexpr2, mty1, mty2) in
  let args = List.map (fun (i,t) -> CArg (i, t)) @@ fvs in
  let fv_tys = List.map snd fvs in
  let ty2 = convert_mty mty2 in
  let fn = name2, List.length fvs + 1, expr2, ty2 in
  funs := fn::!funs;
  CApp (CClos (name2, args, fv_tys), CClosT, expr1, convert_mty mty1, ty2)
and convert_calt = function
| M.Destructor (name, num_abstr, mexpr, mty) ->
  let expr = convert mexpr in
  CDestructor (name, num_abstr, expr, convert_mty mty)
| M.Wildcard (mexpr, mty) ->
  let expr = convert mexpr in
  CWildcard (expr, convert_mty mty)
in convert expr
```

```
let gen_ops () =
  let decls = ref [] in
  let binops = [
    Tc.Plus
  ; Tc.Minus
  ; Tc.Times
  ; Tc.Divide
  ; Tc.Geq
  ; Tc.Gt
  ; Tc.Eq
  ; Tc.Leq
  ; Tc.Lt
  ; Tc.Neq
  ; Tc.And
  ; Tc.Or ]
  in
  let unops = [ Tc.Neg; Tc.Not ] in
  let unop_tys = [ CIntT, CIntT; CBoolT, CBoolT ] in
  let binop_tys = [
    CIntT, CIntT
  ; CIntT, CIntT
  ; CIntT, CIntT
  ; CIntT, CIntT
  ; CIntT, CBoolT
  ; CIntT, CBoolT
  ; CIntT, CBoolT
  ; CIntT, CBoolT
  ; CIntT, CBoolT
  ; CIntT, CBoolT
  ; CBoolT, CBoolT
  ; CBoolT, CBoolT
  ] in
  let gen_binop binop (in_ty, out_ty) =
    let name = string_of_binop binop in
    let clos1 = "__prim__" ^ name, 1, CClos ("__prim__" ^ name ^ "1", [CArg (0,
    let clos2 = "__prim__" ^ name ^ "1", 2, CCall ("__prim__binop__" ^ name, [CA
    decls := ("__prim__binop__" ^ name, ([in_ty; in_ty], out_ty))::!decls;
```

```
    funs := clos1::clos2::!funs;
  in
  let gen_unop unop (in_ty, out_ty) =
    let name = string_of_unop unop in
    let clos = "__prim__" ^ name, 1, CCall ("__prim__unop__" ^ name, [CArg (0, i
    decls := ("__prim__unop__" ^ name, ([in_ty], out_ty))::!decls;
    funs := clos::!funs;
  in
  List.iter2 gen_binop binops binop_tys;
  List.iter2 gen_unop unops unop_tys;
  !decls


let rec convert_decl = function
| M.Arr (in_mty, out_mty) ->
  let tys, out_ty = convert_decl out_mty in
  (convert_mty in_mty :: tys), out_ty
| mty -> [], convert_mty mty

let decl_globals decls =
  let mk_globals (name, (ty_list, out_ty)) =
    let pname = "__" ^ name ^ "__" in
    let rec mk i arg_tys = function
    | [] ->
      CCall (name, List.rev @@ List.mapi (fun i ty -> CArg (i, ty), ty) arg_tys)
    | (ty::tys) ->
      let uname = unique_name pname in
      let expr, out_ty = mk (i+1) (ty::arg_tys) tys in
      let len = List.length (ty::tys) in
      funs := (uname, len, expr, out_ty)::!funs;
      CClos (uname, List.mapi (fun i ty -> CArg (i, ty)) arg_tys, arg_tys), CClo
    in
    ignore @@ mk 0 [] ty_list
  in
  List.iter mk_globals decls

let convert_prog ({ main; letdefs; decls; datadefs } as prog: M.program) =
  let expr = convert_mexpr "__main__" prog main in
```

```
  let globals = List.map (fun (name, mty, mexpr) -> name, (convert_mexpr name pr
  List.iter (fun (name, (expr, ty)) -> funs := (name, 1, expr, ty)::!funs) globa
  let decls = List.map (fun (name, ty) -> name, convert_decl ty) decls in
  decl_globals decls;
  let datatys = List.map (
    fun (dname, cs) -> dname, List.map (
      fun (cname, tys) -> cname, List.map convert_mty tys) cs) datadefs
  in
  let decls = (gen_ops ()) @ decls in
  let ret = { funs = !funs; main = expr; datatys = datatys; decls = decls } in
  funs := [];
  vars := SM.empty;
  ret
```

8.2.9 lib/closure/cast.ml

```
(* Authors: Ben, Jay *)
type arglen = int
type cindex = int
type cname = string

type cty
  = CIntT
  | CBoolT
  | CharT
  | CDataTy of cname
  | CClosT
  | BoxT

type cexpr
  = CInt of int
  | CChar of char
  | CBool of bool
  | CApp of cexpr * cty * cexpr * cty * cty
  | CClos of cname * cexpr list * cty list
  | CCall of cname * (cexpr * cty) list
  | Box of cexpr * cty
  | Unbox of cexpr * cty
```

```
  | CArg of cindex * cty
  | CIf  of cexpr * cexpr * cexpr * cty
  | CConstruction of cname * cexpr list * cty
  | CCase of cexpr * cty * ccase_alt list * cty
and ccase_alt
  = CDestructor of cname * arglen * cexpr * cty
  | CWildcard of cexpr * cty

type ccons = cname * cty list
type fundef = cname * arglen * cexpr * cty
type cdataty = cname * ccons list
type cclosuredef = cname * cty list

type program = {
  funs : fundef list;
  datatys : cdataty list;
  decls : (cname * (cty list * cty)) list;
  main : cexpr;
}

(*
let prog : program = {
  datatys = [
    "List_Int", ["Cons", [CIntT, CDataTy "List_Int"]; "Nothing", [] ]
  ],
  closures = [
    "f", [IntT]; // points at f (we didnt write it below)
    // get arbitrary name F_Clos code ptr to f and args IntT
    "foldl1", [IntT];
  ]
  globals = [
    "foldl",
      [CClosT "f"], // Apply can take in more than a closure
      CClos ("foldl1", [CArg 0]), // return this as a closure (struct Clos_foldl
      CClosT "foldl1";
      struct Clos_Foldl1 foldl(Clos_F) {
        return (struct Clos_foldl1) {
          code_ptr = foldl1,
```

```
          arg0 = C arg0
        }
      }
    "foldl1",
      [CClosT "f"; CInt],
      CClos ("foldl2", [CArg 0, CArg 1]), struct { code_ptr = foldl2, arg0 = CAr
      CClosT "foldl2"; // return type
    "foldl2", [CClosT "f"; CInt; CDataTy "List_Int"],
      CCase (CArg 2, [
      CDestructor ("Nil", [], CArg 1);
      CDestructor ("Cons", [IntT; CDataTy "List_Int"],
        CApp(
          CApp(
            CApp(
              CClos("foldl", []), CArg 0
            ),
          CApp(
            CApp(CArg 0, CArg 1)
          , CArg 3)
          ), CArg 4))])
    ]
  main =
  CApp(CApp(CApp(CClos "foldl", CClos "f"), CInt 0),
  CConstruction ("Cons", [CInt 0, CConstruction ("Cons", [CInt 1, CConstruction
}

@a @b (a -> a) -> (b -> b) -> int
(BoxT -> BoxT) -> (BoxT -> BoxT) -> Int

foo : (@a a -> a) -> (Int, Char)
foo = \f. (f @Int 0, f @Char 'a')


BoxT -> BoxT -> Tuple
foo : (BoxT -> BoxT) -> Tuple
foo = \f. (f (Box 0), f (Box 'a'))

foo : (BoxT -> BoxT) -> (BoxT, BoxT)
```

```
foo = \f. (f (MkBox 0), f (MkBox 'a'))

main :
Unbox (case (foo id) of
  Tuple a b -> a (* Box 0 *), Int)


*)
(*

foo a b = a + b
main = (foo 10) 20


{
  datatys = []
  globals = [
    // Fn Name, Arg Types, returned closure (which we need to build), return typ
    ("foo", [CIntT], CClos ("foo1", [CArg (0, CIntT)]), CClosT "foo1");
    ("foo1", [CIntT; CIntT], CApp (CClos ("prim_add" Add CArg (0, CIntT), CArg (
  ]
  closures = [
    "foo", [];        // points at foo with 0 arguments
    "foo1", [CIntT]   // points at foo1 with 1 argument of type CIntT
  ]
  main =
    CApp(
      CApp(
        CClos ("foo", [], CClosT "foo1"),  // lhs (closure)
        CClosT "foo",// type of lhs (type of closure)
        CInt 10,     // rhs (argument to apply)
        CIntT,       // type of rhs (type of argument to Apply)
        ClosT "foo1" // type of result of apply
      ), // lhs
      CClosT "foo1", // type of lhs
      CInt 20,       // rhs
      CIntT,         // type of rhs
      CIntT          // type of result of apply
```

```
    );
}
*)
```

8.2.10 lib/codegen/codegen.ml

```
(* Authors: Ben, Jay, Sophia *)
module L = Llvm
module C = Closure.Cast
exception CodegenError
exception NotImplemented
let (<<<) f g x = f (g x)
let translate (prog : C.program) =
  (* Create LLVM Context *)
  let context = L.global_context () in

  (* Create LLVM compilation module into which
     we will generate code *)
  let _module = L.create_module context "lingo" in

  (* Build LLVM types *)
  let i32_t       = L.i32_type context in
  let i64_t       = L.i64_type context in
  let char_t      = L.i8_type context in
  let void_t      = L.void_type context in
  let i8_ptr_t    = L.pointer_type (L.i8_type context) in
  let bool_t      = L.i1_type context in
  let decl_struct_t name = L.named_struct_type context name in
  let def_struct_t name fields =
    let llstruct_t = decl_struct_t name in
    L.struct_set_body llstruct_t (Array.of_list fields) false;
    llstruct_t
  in
  let adt_t = def_struct_t "adt" [i64_t; i8_ptr_t] in
  let box_t = def_struct_t "boxt" [i8_ptr_t; i8_ptr_t] in
  let clos_t = def_struct_t "clos" [i8_ptr_t; L.pointer_type i8_ptr_t] in
  let die_function = L.declare_function "__die__" (L.function_type void_t [||])
  let add_terminal builder instr =
```

```
    (match L.block_terminator (L.insertion_block builder) with
      Some _ -> ()
    | None -> ignore (instr builder))
in
ignore (adt_t);
let ltype_of_type = function
| C.CIntT         -> i64_t
| C.CBoolT        -> bool_t
| C.CharT         -> char_t
| C.CDataTy _     -> adt_t
| C.CClosT        -> clos_t
| C.BoxT          -> box_t
in
let create_function_t (cname, _, _, cty) =
  let fun_t = L.function_type (L.pointer_type @@ ltype_of_type cty) ([| i8_ptr
  L.define_function cname fun_t _module
in
let cons_ts =
  let cons_t i (cname, cty_list) =
    let ts = List.map ltype_of_type cty_list in
    cname, (i, def_struct_t cname ts) in
  List.concat @@ List.map (List.mapi cons_t <<< snd) prog.datatys
in
let function_vals = List.map (fun ((name,_,_,_) as gl) -> name, create_functio
let create_decl (name, (arg_ctys, out_cty)) =
  let fn_t = L.function_type (ltype_of_type out_cty) (Array.map ltype_of_type
  L.declare_function name fn_t _module
in
let _ = List.map create_decl prog.decls in
let translate_cexpr fn builder =
  let rec translate_cexpr value_to_set bb extra_args = function
  | C.CInt i ->
    L.build_store (L.const_int i64_t i) value_to_set builder
  | C.CChar c ->
    L.build_store (L.const_int char_t (int_of_char c)) value_to_set builder
  | C.CBool b ->
    L.build_store (L.const_int bool_t (if b then 1 else 0)) value_to_set build
  | C.CIf (predicate, then_expr, else_expr, _ty) ->
```

```
    let brend = L.append_block context "end" fn in
    L.position_at_end brend builder;

    let brtrue = L.append_block context "brtrue" fn in
    L.position_at_end brtrue builder;
    ignore (translate_cexpr value_to_set brtrue extra_args then_expr);
    add_terminal builder (L.build_br brend);

    let brfalse = L.append_block context "brfalse" fn in
    L.position_at_end brfalse builder;
    ignore (translate_cexpr value_to_set brfalse extra_args else_expr);
    add_terminal builder (L.build_br brend);

    L.move_block_after brfalse brend;
    L.position_at_end bb builder;
    let branch =
      let cond_ptr = L.build_alloca bool_t "ifcond" builder in
      ignore (translate_cexpr cond_ptr bb extra_args predicate);
      let cond_val = L.build_load cond_ptr "cond_val" builder in
      L.build_cond_br cond_val brtrue brfalse
    in
    add_terminal builder branch;
    L.position_at_end brend builder;
    value_to_set
| C.Box (cexpr, cty) ->
  let unbox = L.build_alloca (ltype_of_type cty) "unbox" builder in
  ignore (translate_cexpr unbox bb extra_args cexpr);
  let box_ptr = L.build_bitcast unbox (L.pointer_type box_t) "box_ptr" build
  let box_val = L.build_load box_ptr "box_val" builder in
  L.build_store box_val value_to_set builder
| C.Unbox (cexpr, cty) ->
  let boxed = L.build_alloca box_t "boxed" builder in
  ignore (translate_cexpr boxed bb extra_args cexpr);
  let unbox_ptr = L.build_bitcast boxed (L.pointer_type @@ ltype_of_type cty
  let unbox_val = L.build_load unbox_ptr "unbox_val" builder in
  L.build_store unbox_val value_to_set builder
| C.CClos (cname, env, env_tys) ->
  let clos = value_to_set in
```

```
  let clos_fn_ptr = L.build_struct_gep clos 0 "clos_fn_ptr" builder in
  let raw_fn_ptr = L.build_bitcast (List.assoc cname function_vals) i8_ptr_t
  let _ = L.build_store raw_fn_ptr clos_fn_ptr builder in
  let env_ptr = L.build_malloc (L.array_type i8_ptr_t (List.length env)) "en
  let translate_env i (cexpr, cexpr_cty) =
    let arg_raw_ptr_ptr = L.build_in_bounds_gep env_ptr (Array.map (L.const_
    let arg_ptr = L.build_malloc (ltype_of_type cexpr_cty) "closarg_ptr" bui
    ignore (translate_cexpr arg_ptr bb extra_args cexpr);
    let arg_raw_ptr = L.build_bitcast arg_ptr i8_ptr_t "raw_arg_ptr" builder
    ignore (L.build_store arg_raw_ptr arg_raw_ptr_ptr builder);
  in
  ignore (List.mapi translate_env (List.combine env env_tys));
  ignore (L.build_store (L.build_bitcast env_ptr (L.pointer_type i8_ptr_t) "
  value_to_set
| C.CApp (cexpr1, _, cexpr2, cty2, out_ty) ->
  let clos = L.build_alloca clos_t "app_lhs" builder in
  let _ = translate_cexpr clos bb extra_args cexpr1 in
  let to_apply = L.build_alloca (ltype_of_type cty2) "app_rhs" builder in
  let _ = translate_cexpr to_apply bb extra_args cexpr2 in
  let to_apply = L.build_bitcast to_apply i8_ptr_t "raw_app_rhs" builder in
  let fn_t = L.pointer_type @@ L.function_type (L.pointer_type @@ ltype_of_t
  let fn_ptr = L.build_bitcast (L.build_load (L.build_struct_gep clos 0 "raw
  let args = L.build_load (L.build_struct_gep clos 1 "args_ptr" builder) "ar
  let app_res_ptr = L.build_call fn_ptr [| to_apply; args |] "app_res_ptr" b
  let app_res = L.build_load app_res_ptr "app_res" builder in
  L.build_store app_res value_to_set builder
| C.CArg (idx, cty) ->
  let len = List.length extra_args in
  if idx < len then (
    L.build_store (List.nth extra_args idx) value_to_set builder
  ) else if idx - len == 0 then
    let ptr = L.build_bitcast (L.param fn 0) (L.pointer_type @@ ltype_of_typ
    let arg = L.build_load ptr "arg" builder in
    L.build_store arg value_to_set builder
  else (
    let gep_ptr = L.build_bitcast (L.param fn 1) (L.pointer_type @@ L.array_
    let raw_ptr_ptr = L.build_gep gep_ptr [|L.const_int i32_t 0; L.const_int
    let raw_ptr = L.build_load raw_ptr_ptr "raw_arg_ptr" builder in
```

```
    let ptr = L.build_bitcast raw_ptr (L.pointer_type @@ ltype_of_type cty)
    let arg = L.build_load ptr "arg" builder in
    L.build_store arg value_to_set builder)
| C.CConstruction (name, cargs, _) ->
  let tag, cons_t = List.assoc name cons_ts in
  let args = List.map (fun t -> L.build_malloc t "carg_aloc" builder) (Array
  List.iter (fun (arg, carg) -> ignore (translate_cexpr arg bb extra_args ca
  let cons  = L.build_malloc cons_t "cons" builder in
  let build_cons (arg, i) =
    let arg_val = L.build_load arg "carg_aloc_val" builder in
    L.build_store arg_val (L.build_struct_gep cons i "carg" builder) builder
  in
  List.iter (ignore <<< build_cons) (List.combine args (List.init (List.leng
  let cons_ptr = L.build_bitcast cons (i8_ptr_t) "cons_vptr" builder in
  let tag_ptr = L.build_struct_gep value_to_set 0 "tag" builder in
  let data_ptr = L.build_struct_gep value_to_set 1 "data_ptr" builder in
  ignore (L.build_store (L.const_int i64_t tag) tag_ptr builder);
  ignore (L.build_store cons_ptr data_ptr builder);
  value_to_set
| C.CCase (cscrut, _, calts, _out_cty) ->
  let scrut_var = L.build_alloca adt_t "scrut" builder in
  ignore (translate_cexpr scrut_var bb extra_args cscrut);
  let scrut_tag_ptr = L.build_struct_gep scrut_var 0 "switch_tag_ptr" builde
  let scrut_data = L.build_struct_gep scrut_var 1 "scrut_data" builder in
  let scrut_tag = L.build_load scrut_tag_ptr "switch_tag" builder in
  let brend = L.append_block context "case_continue" fn in
  let first_wc = List.find_map (fun x ->
    match x with
    | C.CWildcard   (cexpr, _) -> Some cexpr
    | C.CDestructor _ -> None
  ) calts in
  let destructors = List.filter_map (fun x ->
    match x with
    | C.CWildcard _ -> None
    | C.CDestructor (cname, num_abstr, cexpr, _) -> Some (cname, num_abstr,
  in
  let default_bb = L.append_block context "default" fn in
  L.position_at_end default_bb builder;
```

```
  (match first_wc with
    | Some default_expr ->
    ignore (translate_cexpr value_to_set default_bb extra_args default_expr)
    | _ ->
    ignore (L.build_call (die_function) [||] "" builder);
    );
  ignore (L.build_br brend builder);
  L.position_at_end bb builder;
  let switch = L.build_switch scrut_tag default_bb (List.length calts) build
  let translate_destructor value_to_set case_bb (cname, num_abstr, cexpr) =
    L.position_at_end case_bb builder;
    let cons_tag, cons_t = List.assoc cname cons_ts in
    let scrut_data_deref = L.build_load scrut_data "scrut_data_deref" builde
    let cons_ptr = L.build_bitcast scrut_data_deref (L.pointer_type cons_t)
    let extra_args' = List.init (num_abstr) (fun i ->
      L.build_load (L.build_struct_gep cons_ptr i "cons_destruct_ptr" builde
    L.add_case switch (L.const_int i64_t cons_tag) case_bb;
    let expr = translate_cexpr value_to_set case_bb ((List.rev extra_args')
    ignore (L.build_br brend builder);
    L.position_at_end bb builder;
    expr
  in
  let case_bbs = List.init (List.length destructors) (fun _ -> L.append_bloc
  let cases = List.map2 (translate_destructor value_to_set) case_bbs destruc
  List.iter ignore cases;
  L.move_block_after (List.nth case_bbs (List.length case_bbs - 1)) brend;
  L.position_at_end brend builder;
  switch
| C.CCall (cname, cexpr_list) ->
  let alloc_args = List.map (fun (_, cty) -> L.build_alloca (ltype_of_type c
  ignore (List.map2 (fun aloc (expr, _) -> translate_cexpr aloc bb extra_arg
  let args = List.map (fun aloc -> L.build_load aloc "call_arg" builder) all
  let arg_ctys, out_cty  = List.assoc cname prog.decls in
  let arg_ts = List.map ltype_of_type arg_ctys in
  let out_t = ltype_of_type out_cty in
  let fun_t = L.function_type out_t (Array.of_list arg_ts) in
  let fun_decl = L.declare_function cname fun_t _module in
  let ret = L.build_call fun_decl (Array.of_list args) "call_ret" builder in
```

```
      L.build_store ret value_to_set builder
    in translate_cexpr
  in
  let build_fun (name, _, cexpr, cty) =
    let fn_def = List.assoc name function_vals in
    let builder = L.builder_at_end context (L.entry_block fn_def) in
    let rval_ptr = L.build_malloc (ltype_of_type cty) "rval_ptr" builder in
    ignore (translate_cexpr fn_def builder rval_ptr (L.entry_block fn_def) [] ce
    add_terminal builder (L.build_ret rval_ptr);
  in
  List.iter build_fun prog.funs;
  let main_t = L.function_type i64_t [||] in
  let main_fn = L.define_function "main" main_t _module in
  let builder = L.builder_at_end context (L.entry_block main_fn) in
  let rval = L.build_alloca (i64_t) "ret" builder in
  ignore (translate_cexpr main_fn builder rval (L.entry_block main_fn) [] prog.m
  add_terminal builder (L.build_ret (L.build_load rval "retval" builder));
  _module
```

8.2.11 lib/lib.c

```
// Authors: Ben, Jay, Sophia
#include <stdio.h>
#include <stdlib.h>

typedef struct adt
{
    long long tag;
    char *data;
} ADT;

typedef struct box
{
    char *a;
    char *b;
} Box;

typedef struct tuple
```

```
{
    Box a;
    Box b;
} Tuple;
ADT unit = (ADT){.tag = 0, .data = NULL};
long long print_int(long long t)
{
    printf("%lld\n", t);
    return 0;
}

char print_char(char c)
{
    printf("%c\n", c);
    return c;
}

int print_bool(int b)
{
    if (b == 0)
    {
        printf("false\n");
    }
    else if (b == 1)
    {
        printf("true\n");
    }
    return b;
}

ADT print_string(ADT adt)
{
    printf("%s\n", adt.data);
    return adt;
}
ADT prim_alloc(long long size)
{
    char *p = malloc((size_t)size);
```

```
    return (ADT){.tag = 0, .data = p};
}
ADT prim_drop(ADT d)
{
    free(d.data);
    return unit;
}
ADT set_bit(ADT adt, long long i, char c)
{
    adt.data[i] = c;
    return adt;
}
char *believe_me(char *x)
{
    return x;
}
ADT open_file(ADT filename, ADT filemode)
{
    FILE *fp = fopen(filename.data, filemode.data);
    char *data = (char *)fp;
    return (ADT){.tag = 0, .data = data};
}

ADT read_file(ADT file)
{
    FILE *fp = (FILE *)file.data;
    fseek(fp, 0L, SEEK_END);
    long size = ftell(fp);
    fseek(fp, 0L, SEEK_SET);
    char *data = malloc(size);
    fread(data, 1, size, fp);
    Tuple *tup = malloc(sizeof(Tuple));
    ADT mem;
    mem.tag = 0;
    mem.data = data;
    ADT file2;
    file2.tag = 0;
    file2.data = (char *)fp;
```

```
    tup->a = *((Box *)&mem);
    tup->b = *((Box *)&file2);
    return (ADT){.tag = 0, .data = (char *)tup};
}

ADT close_file(ADT file)
{
    FILE *fp = (FILE *)file.data;
    fclose(fp);
    return unit;
}

int __prim__binop__and(int a, int b) { return a && b; }
int __prim__binop__or(int a, int b) { return a || b; }
long long __prim__binop__plus(int a, int b) { return a + b; }
long long __prim__binop__minus(int a, int b) { return a - b; }
long long __prim__binop__times(int a, int b) { return a * b; }
long long __prim__binop__divide(int a, int b) { return a / b; }
int __prim__binop__geq(long long a, long long b) { return a >= b; }
int __prim__binop__gt(long long a, long long b) { return a > b; }
int __prim__binop__leq(long long a, long long b) { return a <= b; }
int __prim__binop__lt(long long a, long long b) { return a < b; }
int __prim__binop__eq(long long a, long long b) { return a == b; }
int __prim__binop__neq(long long a, long long b) { return a != b; }
int __prim__unop__not(int a) { return !a; }
long long __prim__unop__neg(long long a) { return -a; }
void __die__()
{
    fprintf(stderr, "Unhandled case, exiting.\n");
    exit(1);
}
```

8.2.12 src/lingo.ml

```
(* Authors: Sophia *)
let src_file = Sys.argv.(1)

let string_of_file file =
```

```
  let ic = open_in file in
  let str = really_input_string ic (in_channel_length ic) in
  close_in ic;
  str

let llmodule_string src_file =
  let src = string_of_file src_file in
  let prog = let lexbuf = Lexing.from_string src in Parse.Parser.program Parse.S
  in
  let sast = let core_ast = Core.Conversion.convert prog in Core.Typecheck.check
  in
  let mast = Mono.Conversion.convert_prog sast in
  let cast = Closure.Conversion.convert_prog mast in
  Llvm.string_of_llmodule @@ Codegen.translate cast

let _ = print_endline @@ llmodule_string src_file
```

8.2.13 src/lingoc

```
#!/bin/bash
# Authors: Jay
dune build
# Get Base Name
FILE="$1"
FILE_NO_EXT=${FILE%.*}
BASE=$(basename $FILE_NO_EXT)

# Compile LLVM
LLVM=$(dune exec src/lingo.exe $FILE)

# Compile ASM
ASM=$(echo "$LLVM" | llc)
echo "$ASM" | gcc -c -x assembler -o temp.o -

# Compile to Binary and link
gcc -c -o lib.o lib/lib.c
gcc -no-pie -o "$BASE" lib.o temp.o
```

```
# Clean Up
rm temp.o lib.o
dune clean
```

8.2.14 reg-tests/RunTests.py

```
# Authors: Ben, Jay, Sophia
from os import listdir, getcwd, mkdir, path, remove
import argparse
import subprocess
from shutil import rmtree

FAIL = False

cwd = f'{getcwd()}/reg-tests'
parser = argparse.ArgumentParser(
    description="Runs regression tests for lingo files")
parser.add_argument("-c", "--clean", action='store_true')
parser.add_argument("-s", "--src-file", default=None)
parser.add_argument("-d", "--src-dir", default=f'{cwd}/src')
parser.add_argument("-o", "--diff-dir", default=f'{cwd}/diff')
parser.add_argument("-ll", "--llvm-dir", default=f'{cwd}/llvm')
parser.add_argument("-asm", "--asm-dir", default=f'{cwd}/asm')
parser.add_argument("-ex", "--exec-dir", default=f'{cwd}/exec')
parser.add_argument("-out", "--out-dir", default=f'{cwd}/out')

parser.add_argument("-lib", default=f'{getcwd()}/lib/lib.c')
args = parser.parse_args()

diff_dir = args.diff_dir
in_src_file = args.src_file
src_dir = args.src_dir
llvm_dir = args.llvm_dir
asm_dir = args.asm_dir
exec_dir = args.exec_dir
out_dir = args.out_dir
lib = args.lib
```

```
class RunException(Exception):
    def __init__(self, args, returncode, stdout, stderr):
        self.args = args
        self.returncode = returncode
        self.stdout = stdout
        self.stderr = stderr

    def tuple(self):
        return self.args, self.returncode, self.stdout, self.stderr


class bcolors:
    HEADER = '\033[95m'
    OKBLUE = '\033[94m'
    OKCYAN = '\033[96m'
    OKGREEN = '\033[92m'
    WARNING = '\033[93m'
    FAIL = '\033[91m'
    ENDC = '\033[0m'
    BOLD = '\033[1m'
    UNDERLINE = '\033[4m'


def run(args):
    proc = subprocess.run(args, capture_output=True)
    args, returncode, stdout, stderr = \
        args, proc.returncode, proc.stdout, proc.stderr
    if returncode != 0:
        raise RunException(args, returncode, stdout, stderr)
    return stdout, stderr


def log(msg, should_print=True):
    if should_print:
        print(msg)
    with open(f'{cwd}/log.txt', 'a+') as file:
        file.write(msg + '\n')
```

```
def get_llvm(src, llvm_file):
    stdout, _ = run(
        ["dune", "exec", f'{"./src/lingo.exe"}', f'{src_dir}/{src}'])
    with open(llvm_file, 'wb') as file:
        file.write(stdout)


def build_asm(llvm_file, asm_file):
    run(["llc", llvm_file, "-o", asm_file])


def build_exec(asm_file, exec_file):
    run(["gcc", "-no-pie", asm_file, lib, "-o", exec_file])


def run_exec(execu, out):
    stdout, _ = run([execu])
    with open(out, 'w') as file:
        file.write(stdout.decode('utf-8'))


def diff_output(lingo_file, expected, actual, out):
    try:
        run(["diff", expected, actual])
        log(f'{bcolors.OKGREEN}...{lingo_file} PASSED {bcolors.ENDC}\n')
    except RunException as err:
        args, returncode, stdout, stderr = err.tuple()

        global FAIL
        FAIL = True

        log(f'{" ".join(args)} returned {returncode}')
        log(f'STDOUT: \n {stdout.decode("utf-8")}')
        log(f'STDERR: \n {stderr.decode("utf-8")}')
        log(f'{bcolors.FAIL}Diff between {expected} '
            f'and {actual}.{bcolors.ENDC}')
```

```
        log(f'{bcolors.FAIL}...{lingo_file} FAILED {bcolors.ENDC}\n')

        with open(out, 'wb') as file:
            file.write(stdout)


def run_test(src_file):
    lingo_file = f'{src_file}.lingo'
    llvm_file = f'{llvm_dir}/{src_file}.llvm'
    asm_file = f'{asm_dir}/{src_file}.s'
    execu_file = f'{exec_dir}/{src_file}.exe'
    expected_out_file = f'{out_dir}/{src_file}.out'
    out_file = f'{out_dir}/{src_file}.actual.out'
    diff_file = f'{diff_dir}/{src_file}.diff'

    log(f'{bcolors.WARNING}---------- TESTING '
        f'{lingo_file}... ----------{bcolors.ENDC}')
    try:
        get_llvm(lingo_file, llvm_file)
        build_asm(llvm_file, asm_file)
        build_exec(asm_file, execu_file)
        run_exec(execu_file, out_file)
    except RunException as err:
        _, _, _, stderr = err.tuple()
        with open(out_file, 'w') as file:
            file.write(stderr.decode('utf-8'))

    diff_output(lingo_file, expected_out_file, out_file, diff_file)


def clean():
    print(f'{bcolors.OKCYAN}CLEANING...{bcolors.ENDC}')
    # Get Rid of llvm, asm, exec, diff, .actual.out

    to_delete = [llvm_dir, asm_dir, exec_dir, diff_dir]
    for p in to_delete:
        if path.exists(p):
            rmtree(p)
```

```
    for f_name in listdir(out_dir):
        if 'actual.out' in f_name:
            remove(f'{out_dir}/{f_name}')

    log = f'{cwd}/log.txt'
    if path.exists(log):
        remove(log)


def make_dirs():
    to_create = [llvm_dir, asm_dir, exec_dir, diff_dir]
    for p in to_create:
        if not path.exists(p):
            mkdir(p)


if args.clean:
    clean()
    run(["dune", "clean"])
else:
    make_dirs()
    run(["dune", "build"])
    if in_src_file:
        run_test(in_src_file)

    else:
        for in_src_file in listdir(src_dir):
            in_src_f, *rest = in_src_file.split('.lingo')
            if (f'{".ignore"}') in rest:
                continue
            run_test(in_src_f)

    run(["dune", "clean"])

if FAIL:
    exit(1)
```

8.2.15 docker-compose.yml

```
services:
    lingo_testbed:
        image: lingo:version2
        volumes:
            - .:/home/lingo
```

8.2.16 Dockerfile

```
# Based on 20.04 LTS
# Authors: Jay
FROM ubuntu:focal

ENV DEBIAN_FRONTEND=noninteractive

RUN apt-get -yq update && \
    apt-get -y upgrade && \
    apt-get -yq --no-install-suggests --no-install-recommends install \
    ocaml \
    menhir \
    llvm \
    llvm-dev \
    m4 \
    git \
    aspcud \
    ca-certificates \
    python2.7 \
    python3.9 \
    pkg-config \
    cmake \
    opam \
    vim


# RUN ln -s /usr/bin/lli-10 /usr/bin/lli
# RUN ln -s /usr/bin/llc-10 /usr/bin/llc
```

```
RUN opam init -yq --disable-sandboxing --reinit
RUN opam switch create 4.12.0
RUN opam install -yq \
    llvm.10.0.0 \
    ocaml \
    dune \
    utop



WORKDIR /home/lingo

COPY . .

ENTRYPOINT ["opam", "config", "exec", "--"]

CMD ["bash"]
```

8.2.17 test.sh

```
#!/bin/bash
# Authors: Jay
ARGS="$@"
# Run Python Tests
docker-compose run lingo_testbed bash -c "python3 ./reg-tests/RunTests.py $ARGS"
FAIL=$?
# Clean Up Stopped Docker Container
docker rm $(docker ps -a -q) > /dev/null 2>&1
if [[ $FAIL -eq 1 ]]
then
    exit 1
fi
```

8.2.18 demo/file.lingo

```
(* Authors: Ben *)
data File where
data Mem where
```

```
data Unit where
  Unit : Unit;
data Tuple a b #p #q where
    Tuple : a -p> b -q> Tuple a b;

print_int : Int -* Int;
print_string : Mem -* Mem;
prim_alloc : Int -* Mem;
prim_drop : Mem -* ();
set_bit : Mem -* Int -> Char -> Mem;
open_file : Mem -> Mem -> File;
read_file : File -* Tuple Mem File #Unr #One;
close_file : File -* ();

data String where
    E : String;
    C : Char -> String -> String;

len : String -> Int
    = \s. case s of
        C c s -> 1 + len s;
        _ -> 0;
    ;
printString : Mem -* Mem = print_string;

string_to_mem : String -> Mem
            = \s.
            let m : Mem = prim_alloc (1 + len s) in
            let string_to_mem' : String -> Mem -* Int -> Mem
               = \s. \m. \i. case s of
                     C c s -> string_to_mem' s (set_bit m i c) (i + 1);
                     _ -> set_bit m i '\000';
            in
            string_to_mem' s m 0;

openFile : String -> String -> (File -* ()) -> ()
         = \filename. \mode. \k. k (open_file (string_to_mem filename) (string_t
```

```
readFile : File -* Tuple Mem File #Unr #One = read_file;

closeFile : File -* () = close_file;

const : @a @b a -> b -> a
      = \a. \b. \x. \y. x;

printFile : String -> String -> ()
          =
          \filename. \mode.
            let f : File -* () = \file.
              case (readFile file) of
                Tuple str file ->
                  let x : Mem = printString str in
                  let y : () = closeFile file in
                  closeFile file;
            in
            openFile filename mode f;

main : Int = const @Int @() 0 (printFile "./reg-tests/file.txt" "r");
```

8.2.19 demo/malloc.lingo

```
(* Authors: Ben, Jay *)
data Mem where

data Unit where
    Unit : Unit;

prim_alloc : Int -* Mem;
prim_drop : Mem -* ();
print_string : Mem -* Mem;
print_int : Int -> Int;
set_bit : Mem -* Int -> Char -> Mem;

data String where
    E : String;
```

```
    C : Char -> String -> String;

len : String -> Int
    = \s. case s of
        C c s -> 1 + len s;
        _ -> 0;
    ;

alloc : Int -> (Mem -* ()) -> ()
      = \m. \f. f (prim_alloc m);

drop : Mem -* ()
     = prim_drop;

stringToMem' : String -> Mem -* Int -> Mem
             = \s. \m. \i. case s of
                    C c s -> stringToMem' s (set_bit m i c) (i + 1);
                    _ -> set_bit m i '\000';
            ;

stringToMem : String -> (Mem -* ()) -> ()
            = \s. \f. alloc (1 + len s) (\m. f (stringToMem' s m 0));

printAndDrop : Mem -* ()
             = \m. drop (print_string m);

const : @a @b a -> b -> a
      = \a. \b. \x. \y. x;

main : Int = const @Int @Unit 0 (stringToMem "hi" printAndDrop);
```

8.2.20 demo/stephen.lingo

```
(* Authors: Ben, Jay, Sophia *)
print_int : Int -> Int;
succ x : Int -> Int = x + 1;

compose p q x y z stephen a edwards : #p #q @x @y @z (y -p> z) -* (x -q> y) -p>
```

```
main : Int = (compose #Unr #Unr @Int @Int @Int print_int succ) 41;
```

8.3 Test listing

All of the following test source files and their expected output files can be found in `reg-tests/src` and `reg-tests/out` respectively.

8.3.1 airth1.lingo

```
print_int : Int -> Int;
main : Int = print_int (39 + 3);
```

8.3.2 airth1.out

```
42
```

8.3.3 arith2.lingo

```
print_int : Int -> Int;
main : Int = print_int (1 + 2 * 3 + 4);
```

8.3.4 arith2.out

```
11
```

8.3.5 arith3.lingo

```
print_int : Int -> Int;
main : Int =
    let a : Int = 42 in
    let a : Int = a + 5 in
    print_int a;
```

8.3.6 arith3.out

```
47
```

8.3.7 assign_fail1.lingo

```
main : Bool =
    let a : Int = 10 in
    let b : Bool = 10 in
    b;
```

8.3.8 assign_fail1.out

```
Type Mismatch: Bool /= Int
Fatal error: exception Core.Typecheck.TypeMismatch(_, _)
```

8.3.9 assign_fail2.lingo

```
foo x : Int -> Int = x;

main : Int = foo;
```

8.3.10 assign_fail2.out

```
Type Mismatch: Int /= (Int -Unr> Int)
Fatal error: exception Core.Typecheck.TypeMismatch(_, _)
```

8.3.11 compose.lingo

```
print_int : Int -> Int;
succ x : Int -> Int = x + 1;

compose p q a b c f g x : #p #q @a @b @c (b -p> c) -> (a -q> b) -> a -p*q> c = f
main : Int = (compose #Unr #Unr @Int @Int @Int print_int succ) 100;
```

8.3.12 compose.out

```
101
```

8.3.13 compose_fail.lingo

```
print_int : Int -> Int;
succ : Int -> Int = \x. x + 1;
```

```
compose p q a b c f g x : #p #q @a @b @c (b -p> c) -> (a -q> b) -> a -* c = f (g
main : Int = (compose #Unr #Unr @Int @Int @Int print_int succ) 100;
```

8.3.14 compose_fail.out

```
Multiplicity Mismatch: #4*#3 > One
Fatal error: exception Core.Typecheck.MultMismatch(_, 0)
```

8.3.15 compose_fail2.lingo

```
 print_int : Int -> Int;
succ x : Int -* Int = x + 1;

compose p q a b c f g x : #p #q @a @b @c (b -p> c) -> (a -q> b) -> a -p*q> c = f
main : Int =
    let f : Int -* Int = \x. (compose #Unr #One @Int @Int @Int print_int succ) x
    in f 100;
```

8.3.16 compose_fail2.out

```
Multiplicity Mismatch: Unr > One
Fatal error: exception Core.Typecheck.MultMismatch(1, 0)
```

8.3.17 die.lingo

```
data Animal where
    Cow : Int -> Animal;
    Horse : Int -> Animal;

main : Int =
    case (Cow 0) of
        Horse i -> i;
    ;
```

8.3.18 die.out

```
Unhandled case, exiting.
```

8.3.19 extern1.lingo

```
print_char : Char -> Char;
print_bool : Bool -> Bool;
print_int : Int -> Int;

main : Int =
    let x : Char = print_char 'a' in
    let x : Char = print_char '0' in
    let x : Bool = print_bool true in
    let x : Bool = print_bool false in
    print_int 42;
```

8.3.20 extern1.out

```
a
0
true
false
42
```

8.3.21 fac.lingo

```
print_int : Int -> Int;

f : Int -> Int = \x. if x == 0 then 1 else x * f (x - 1);
main : Int = print_int (f 5);
```

8.3.22 fac.out

```
120
```

8.3.23 fac_fail.lingo

```
print_int : Int -> Int;
f : Int = \x. if x == 0 then 1 else x * f (x - 1);
main : Int = f 5;
```

8.3.24 fac_fail.out

```
Fatal error: exception Core.Typecheck.ExpectedAbs(_)
```

8.3.25 fib.lingo

```
print_int : Int -> Int;

fib x : Int -> Int =
    if (x < 2) then 1 else (fib (x - 1)) + fib (x - 2);

main : Int = print_int (fib 10);
```

8.3.26 fib.out

```
89
```

8.3.27 file.lingo

```
data File where
data Mem where

data Tuple a b #p #q where
    Tuple : a -p> b -q> Tuple a b;

print_int : Int -* Int;
print_string : Mem -* Mem;
prim_alloc : Int -* Mem;
prim_drop : Mem -* ();
set_bit : Mem -* Int -> Char -> Mem;
open_file : Mem -> Mem -> File;
read_file : File -* Tuple Mem File #Unr #One;
close_file : File -* ();

data String where
    E : String;
    C : Char -> String -> String;

len : String -> Int
```

```
    = \s. case s of
        C c s -> 1 + len s;
        _ -> 0;
    ;
printString : Mem -* Mem = print_string;

string_to_mem : String -> Mem
            = \s.
            let m : Mem = prim_alloc (1 + len s) in
            let string_to_mem' : String -> Mem -* Int -> Mem
               = \s. \m. \i. case s of
                     C c s -> string_to_mem' s (set_bit m i c) (i + 1);
                     _ -> set_bit m i '\000';
            in
            string_to_mem' s m 0;

openFile : String -> String -> (File -* ()) -> ()
         = \filename. \mode. \k. k (open_file (string_to_mem filename) (string_t

readFile : File -* Tuple Mem File #Unr #One = read_file;

closeFile : File -* () = close_file;

const : @a @b a -> b -> a
      = \a. \b. \x. \y. x;

printFile : String -> String -> ()
          =
          \filename. \mode.
            let f : File -* () = \file.
              case (readFile file) of
                Tuple str file ->
                  let x : Mem = printString str in
                  closeFile file;
            in
            openFile filename mode f;

main : Int = const @Int @() 0 (printFile "./reg-tests/file.txt" "r");
```

8.3.28 file.out

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequa
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu
Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est laborum.
```

8.3.29 func1.lingo

```
print_char : Char -> Char;

return_a x : Int -> Char = 'a';
const y : Char -> Int = 0;

main : Int = const (print_char (return_a 10));
```

8.3.30 func1.out

```
a
```

8.3.31 func2.lingo

```
print_int : Int -> Int;

bar a b c : Int -> Int -> Int -> Int = a + b + c;

main : Int = print_int (bar 10 10 10);
```

8.3.32 func2.out

```
30
```

8.3.33 func3.lingo

```
print_int : Int -> Int;
```

```
succ x : Int -> Int = x + 10;

succ_succ y : Int -> Int = succ (succ y);

main : Int = print_int (succ_succ 10);
```

8.3.34 func3.out

```
30
```

8.3.35 func4.lingo

```
print_bool : Bool -> Bool;

bool_int_bool_int_bool b1 i1 b2 i2 : Bool -> Int -> Bool -> Int -> Bool = true;

const x : Bool -> Int = 0;

main : Int = const (print_bool (bool_int_bool_int_bool true 10 false 10));
```

8.3.36 func4.out

```
true
```

8.3.37 func5.lingo

```
f a b x : @a #b a -b> a = x;

const a b x y : @a @b (a -> b -> Int) = 0;

main : Int = (const @Int @Bool (f @Int #One 10) (f @Bool #One true));
```

8.3.38 func5.out

8.3.39 func_fail1.lingo

```
f x y z : Int -> Int -> Int = x;

(* Abs Expected *)
main : Int = (f 1 2 3);
```

8.3.40 func_fail1.out

```
Fatal error: exception Core.Typecheck.ExpectedAbs(_)
```

8.3.41 func_fail2.lingo

```
f a b x : @a #b a -b> a = x;

const a b : @a @b (a -> b -> Int) = 0;

(* Must specify const polymorphic types *)
main : Int = (const (f @Int #One 10) (f @Bool #One true));
```

8.3.42 func_fail2.out

```
Type Mismatch: (#1 -Unr> (#0 -Unr> Int)) /= Int
Fatal error: exception Core.Typecheck.TypeMismatch(_, _)
```

8.3.43 func_fail3.lingo

```
data Tuple a b #p #q where
    Tuple : a -p> b -q> Tuple a b #p #q;

f a b x : @a #b a -b> (Tuple @a @a #b #b) = Tuple @a @a #b #b x x;

(* const t : Tuple @Int @Int #Unr #Unr -> Int = 0; *)

main : Tuple @Int @Int #Unr #Unr = f @Int #Unr 10;
```

8.3.44 func_fail3.out

```
Multiplicity Mismatch: Unr > #0
```

```
Fatal error: exception Core.Typecheck.MultMismatch(1, _)
```

8.3.45 hello_world.lingo

```
print_int : Int -> Int;
main : Int = print_int 0;
```

8.3.46 hello_world.out

```
0
```

8.3.47 id.lingo

```
id a c x : @a #c a -c> a = x;

main : Int = id @Int #Unr 10;
```

8.3.48 id.out

8.3.49 if1.lingo

```
print_int : Int -> Int;
main : Int = if true then (print_int 42) else (print_int 17);
```

8.3.50 if1.out

```
42
```

8.3.51 if2.lingo

```
print_int : Int -> Int;
main : Int = if false then (print_int 42) else if true then (print_int 42) else
```

8.3.52 if2.out

```
42
```

8.3.53 let1.lingo

```
print_int : Int -> Int;

main : Int =
    let fac x : Int -> Int = if x == 0 then 1 else x * fac (x - 1) in
    let x : Int = 1 in
    let y : Int = 2 in
    print_int (fac (x + y));
```

8.3.54 let1.out

```
6
```

8.3.55 let2.lingo

```
print_int : Int -> Int;

main : Int =
    let x : Int = 10 in
    let y : Int = 20 in
    let z : Int = 12 in
    print_int (x + y + z);
```

8.3.56 let2.out

```
42
```

8.3.57 let3.lingo

```
print_int : Int -> Int;

data Maybe a #p where
  Just    : a -p> Maybe a #p;
  Nothing : Maybe a #p;

main : Int =
    let j : Maybe @Int #Unr = Just @Int #Unr 10 in
    let cs : Maybe Int #Unr -> Int
```

```
        = \x. case x of
            Just i -> print_int i;
            _ -> 0;
    in
    cs j;
```

8.3.58 let3.out

```
10
```

8.3.59 let4.lingo

```
print_int : Int -> Int;

main : Int =
    let f : (Int -> Int) -> (Int -> Int) -> Int -> Int = \g. \h. \x. g (h x) in
    let succ : Int -> Int = \x. x + 1 in
    let dec : Int -> Int = \x. x - 1 in
    print_int (f succ dec 10);
```

8.3.60 let4.out

```
10
```

8.3.61 let5.lingo

```
print_int : Int -> Int;

main : Int =
    let f g h x : (Int -> Int) -> (Int -> Int) -> Int -> Int = g (h x) in
    let succ x : Int -> Int = x + 1 in
    let dec x : Int -> Int = x - 1 in
    print_int (f succ dec 10);
```

8.3.62 let5.out

```
10
```

8.3.63 let6.lingo

```
print_int : Int -> Int;
main : Int =
    let #Unr x : Int = 10 in
    print_int (x + x);
```

8.3.64 let6.out

```
20
```

8.3.65 let_fail1.lingo

```
print_int : Int -> Int;
main : Int =
    let #One x : Int = 10 in
    print_int (x + x);
```

8.3.66 let_fail1.out

```
Multiplicity Mismatch: Unr > One
Fatal error: exception Core.Typecheck.MultMismatch(1, 0)
```

8.3.67 let_fail2.lingo

```
print_int : Int -> Int;

main : Int =
    let #One id x : Int -> Int = x in
    let g x y : Int -> Int -> Int = y in
    print_int (g (id 10) (id 10));
```

8.3.68 let_fail2.out

```
Multiplicity Mismatch: Unr > One
Fatal error: exception Core.Typecheck.MultMismatch(1, 0)
```

8.3.69 malloc.lingo

```
data Mem where

data Unit where
    Unit : Unit;

prim_alloc : Int -* Mem;
prim_drop : Mem -* ();
print_string : Mem -* Mem;
print_int : Int -> Int;
set_bit : Mem -* Int -> Char -> Mem;

data String where
    E : String;
    C : Char -> String -> String;

len : String -> Int
    = \s. case s of
        C c s -> 1 + len s;
        _ -> 0;
    ;

alloc : Int -> (Mem -* ()) -> ()
      = \m. \f. f (prim_alloc m);

drop : Mem -* ()
     = prim_drop;

stringToMem' : String -> Mem -* Int -> Mem
             = \s. \m. \i. case s of
                    C c s -> stringToMem' s (set_bit m i c) (i + 1);
                    _ -> set_bit m i '\000';
            ;

stringToMem : String -> (Mem -* ()) -> ()
            = \s. \f. alloc (1 + len s) (\m. f (stringToMem' s m 0));
```

```
printAndDrop : Mem -* ()
             = \m. drop (print_string m);

const : @a @b a -> b -> a
      = \a. \b. \x. \y. x;

main : Int = const @Int @Unit 0 (stringToMem "hi" printAndDrop);
```

8.3.70 malloc.out

```
hi
```

8.3.71 maybe.lingo

```
data Maybe a #p where
  Just    : a -p> Maybe a #p;
  Nothing : Maybe a #p;

foo : Int -> Maybe Int #One
    = \x. Just @Int #One x;

foo' m
  : Maybe Int #One -> Int
  = case m of
    Just i -> i;
    _ -> 0;
  ;

print_int : Int -> Int;
main : Int = print_int (foo' (Just @Int #One 1));
```

8.3.72 maybe.out

```
1
```

8.3.73 nomain.lingo

```
foo a x y : @a (a -> a) -> a -> a = x y;
bar b z : @b b -> b = z;
```

8.3.74 nomain.out

8.3.75 ops1.lingo

```
print_int : Int -> Int;
print_bool : Bool -> Bool;

main : Int =
    let x : Int = print_int (1 + 2) in
    let x : Int = print_int (1 - 2) in
    let x : Int = print_int (1 * 2) in
    let x : Int = print_int (100 / 2) in
    let x : Int = print_int (99) in
    let x : Bool = print_bool (1 == 2) in
    let x : Bool = print_bool (1 == 1) in
    let x : Int = print_int (99) in
    let x : Bool = print_bool (1 != 2) in
    let x : Bool = print_bool (1 != 1) in
    let x : Int = print_int (99) in
    let x : Bool = print_bool (1 < 2) in
    let x : Bool = print_bool (2 < 1) in
    let x : Int = print_int (99) in
    let x : Bool = print_bool (1 <= 2) in
    let x : Bool = print_bool (1 <= 1) in
    let x : Bool = print_bool (2 <= 1) in
    let x : Int = print_int (99) in
    let x : Bool = print_bool (1 > 2) in
    let x : Bool = print_bool (2 > 1) in
    let x : Bool = print_bool (1 >= 2) in
    let x : Bool = print_bool (1 >= 1) in
    let x : Bool = print_bool (1 >= 1) in
    let x : Bool = print_bool (2 <= 1) in
    0;
```

8.3.76 ops1.out

```
3
```

```
-1
2
50
99
false
true
99
true
false
99
true
false
99
true
true
false
99
false
true
true
true
true
false
```

8.3.77 ops2.lingo

```
print_int : Int -> Int;
print_bool : Bool -> Bool;

main : Int =
    let x : Bool = print_bool (true) in
    let x : Bool = print_bool (false) in
    let x : Bool = print_bool (true && true) in
    let x : Bool = print_bool (false && true) in
    let x : Bool = print_bool (true && false) in
    let x : Bool = print_bool (false && false) in
    let x : Bool = print_bool (true || true) in
    let x : Bool = print_bool (true || false) in
```

```
    let x : Bool = print_bool (false || true) in
    let x : Bool = print_bool (false || false) in
    let x : Bool = print_bool (!false) in
    let x : Bool = print_bool (!true) in
    let x : Int = print_int (-10) in
    0;
```

8.3.78 ops2.out

```
true
false
true
false
false
false
true
true
true
false
true
false
-10
```

8.3.79 syntax1.lingo

```
print_int : Int -> Int;
main : Int = (\x. print_int x : Int -> Int) 0;
```

8.3.80 syntax1.out

```
0
```

8.3.81 syntax_fail1.lingo

```
(* Syntax Error No Colon *)
main Int = 1;
```

8.3.82 syntax_fail1.out

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

8.3.83 syntax_fail2.lingo

```
(* Syntax Error No Equals *)
main : Int 1;
```

8.3.84 syntax_fail2.out

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

8.3.85 syntax_fail3.lingo

```
print_int : Int -> Int;
f : Int -> Int = \x. x;

(* Syntax Error No Ending SemiColon *)
main : Int = print_int (f 10)
```

8.3.86 syntax_fail3.out

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

8.3.87 usage_fail1.lingo

```
id x : Int -* Int = x;

use a f y : @a (a -> a) -> a -> a = f y;

main : Int = use @Int id 10;
```

8.3.88 usage_fail1.out

```
Type Mismatch: (Int -Unr> Int) /= (Int -One> Int)
Fatal error: exception Core.Typecheck.TypeMismatch(_, _)
```

8.3.89 usage_fail2.lingo

```
id x : Int -* Int = x;

use a f y : #a (Int -a> Int) -> Int -> Int = f y;

main : Int = use #Unr id 10;
```

8.3.90 usage_fail2.out

```
Type Mismatch: (Int -Unr> Int) /= (Int -One> Int)
Fatal error: exception Core.Typecheck.TypeMismatch(_, _)
```