Usage

```
Run alg program for REPL.
```

Features

[Product Types](#) (from Class Webpage):

```
t ::= ...
      {t, t, ...}
      t.i             (i≥1)

v ::= ...
      {v, v, ...}

T ::= ...
      {T, T, ...}

E ::= ...
      E.i
      {v, v, ..., E, t, t, ...}

Evaluation Rules

E-ProjTuple
T-Tuple
T-Proj
```

Sum Types (from TAPL):

```
t ::= …
       inl t as T
       inr as T
       case t of inl x=>t| inr x=>t

v ::= …
       inl v as T
       inr v as T

t ::=
       T+T

Evaluation Rules
From Chapter 11.9 Page 132
E-CASE

From Chapter 11.9 Page 135
E-CASEINL
E-CASEINR
E-INL
E-INR
T-INL
T-INR
```

Recursive Types (from TAPL):

From Chapter 20.2 Page 276

```
t ::= …
      fold t
      unfold t

v ::= …
      fold v
      unfold v

T ::= …
      X
      μX.T
```

Evaluation Rules

E-FLD
E-UNFLD
T-FLD
T-UNFLD

I assumed that there will only be one recursive type in each context, so I used fold t instead of fold [T] t and likewise for unfold. Additionally, recursive typing was only implemented for sum and product types.

Example Outputs

cdr

Input	Output
(\l:μX.Bool+Nat*X. case (unfold l) of inl y => y | inr z => z.2) {0, false}	case unfold {0, false} of inl y => y | inr z => z.2 : μX.Bool+Nat*X
(\l:μX.Bool+Nat*X. case (unfold l) of inl y => y | inr z => z.2) false	case unfold false of inl y => y | inr z => z.2 : μX.Bool+Nat*X
(\l:μX.Bool+Nat*X. case (unfold l) of inl y => y | inr z => z.2) 0	error: types do not match
(\l:μX.Nat+Bool*X. case (unfold l) of inl y => y | inr z => z.2) {iszero 0, {iszero (succ 0), {true, {false, {false, (succ 0)}}}}}	case unfold {true, {false, {true, {false, {false, succ 0}}}}} of inl y => y | inr z => z.2 : μX.Nat+Bool*X

car

Input	Output
(\l:μX.Bool+Nat*X. case (unfold l) of inl y => 0 | inr z => z.1) {0, {0, false}}	case unfold {0, {0, false}} of inl y => 0 | inr z => z.1 : Nat
(\l:μX.Bool+Nat*X. case (unfold l) of inl y => 0 | inr z => z.1)	λl. case unfold l of inl y => 0 | inr z => z.1 : (μX.Bool+Nat*X)->Nat
(\l:μX.Bool+Nat*X. case (unfold l) of inl y => 0 | inr z => z.1) 0	error: types do not match

cons

(\x:Nat.\y:μX.Bool+Nat*X. fold {x, y}) 0 {0, {0, false}}	fold {0, {0, {0, false}}} : μX.Bool+Nat*X
(\x:Bool.\y:μX.Bool+Nat*X. fold {x, y}) false {0, {0, false}}	error: {x, y} is not of specified recursive type

Sum

case inr true as Nat+Bool of inl y => if iszero y then true else false | inr z => if z then false else true	false : Bool
case inl (if true then 0 else succ 0) as Nat+Bool of inl y => iszero y | inr z => z	true : Bool
(\x:Nat. inr x as Bool+Nat) (succ 0)	inr (succ 0) : Bool+Nat

Product

{true, false}	{true, false} : Bool*Bool
{true, false}.1	true : Bool
let x = {true, false} in if x.1 then x.2 else x.1	false : Bool