input : Lambda expression representing a program
output : Expressions annotated with address space information

```
inferAddressSpaceProg(lambda)
1  foreach param in lambda.params do
2    if param.type is ScalarType then param.as = PrivateMemory;
3    else param.as = GlobalMemory;
4  inferASExpr(lambda.body, null)

inferASExpr(expr, writeTo)
5  switch expr.type do
6    case Literal expr.as = PrivateMemory;
7    case Param assert (expr.as != null);
8    case FunCall
9      foreach arg in expr.args do
10       inferASExpr(arg, writeTo)
11     switch expr.f.type do
12       case is UserFun
13         if writeTo != null then expr.as = writeTo;
14         else expr.as = inferASFromArgs(expr.args);
15       case is Lambda inferASFunCall(expr.f, expr.args, writeTo);
16       case is toPrivate
17        inferASFunCall(expr.f.lambda, expr.args, PrivateMemory);
18       case is toLocal
19        inferASFunCall(expr.f.lambda, expr.args, LocalMemory);
20       case is toGlobal
21        inferASFunCall(expr.f.lambda, expr.args, GlobalMemory);
22       case is Reduce
23        inferASFunCall(expr.f.f, expr.args, expr.f.init.as);
24       case is Iterate or Map
25        inferASFunCall(expr.f.f, expr.args, writeTo);
26       otherwise do  expr.as = expr.args.as;

inferASFunCall(lambda, args, writeTo)
27 foreach p in lambda.params and a in args do p.as = a.as;
28 inferASExpr(lambda.body, writeTo)
```

Algorithm 1: Recursive address space inference algorithm