While20.pdf

Juanma21_

Teoría de Los Lenguajes de Programación

4º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

While20.hs

```haskell
-- ------------------------------------------------------------------
-- While20.hs
-- ------------------------------------------------------------------

module While20 where

type  Var   =  String

data  Aexp  =  N Integer
            |  V Var
            |  Add Aexp Aexp
            |  Mult Aexp Aexp
            |  Sub Aexp Aexp
            |  Div Aexp Aexp
            deriving (Show, Eq)

data  Bexp  =  TRUE
            |  FALSE
            |  Eq Aexp Aexp
            |  Le Aexp Aexp
            |  Neg Bexp
            |  And Bexp Bexp
            deriving (Show, Eq)

data  Stm   =  Ass Var Aexp
            |  Skip
            |  Comp Stm Stm
            |  If Bexp Stm Stm
            |  Case Aexp LabelledStms
            |  Swap Var Var
            |  For Stm Bexp Stm Stm
            deriving Show

data LabelledStms = LabelledStm [Integer] Stm LabelledStms
                  | Default Stm
                  | EndLabelledStms
                  deriving Show

type  Z      =  Integer
type  T      =  Bool
type  State  =  Var -> Z

aVal :: Aexp -> State -> Z
aVal (N n) _        =  n
aVal (V x) s        =  s x
aVal (Add a1 a2) s  =  aVal a1 s + aVal a2 s
aVal (Sub a1 a2) s  =  aVal a1 s - aVal a2 s
aVal (Mult a1 a2) s =  aVal a1 s * aVal a2 s
aVal (Div a1 a2) s  =  aVal a1 s `div` aVal a2 s

bVal :: Bexp -> State -> T
```

```
52 bVal TRUE _         =  True
53 bVal FALSE _        =  False
54 bVal (Eq a1 a2) s   =  aVal a1 s == aVal a2 s
55 bVal (Le a1 a2) s   =  aVal a1 s <= aVal a2 s
56 bVal (Neg b) s      =  not(bVal b s)
57 bVal (And b1 b2) s =  bVal b1 s && bVal b2 s
58
```

1/6

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

ING BANK NV se encuentra adherido al Sistema de Garantía de Depósitos Holandés con una garantía de hasta 100.000 euros por depositante. Consulta más información en ing.es

Que te den **10 € para gastar** es una fantasía. ING lo hace realidad.

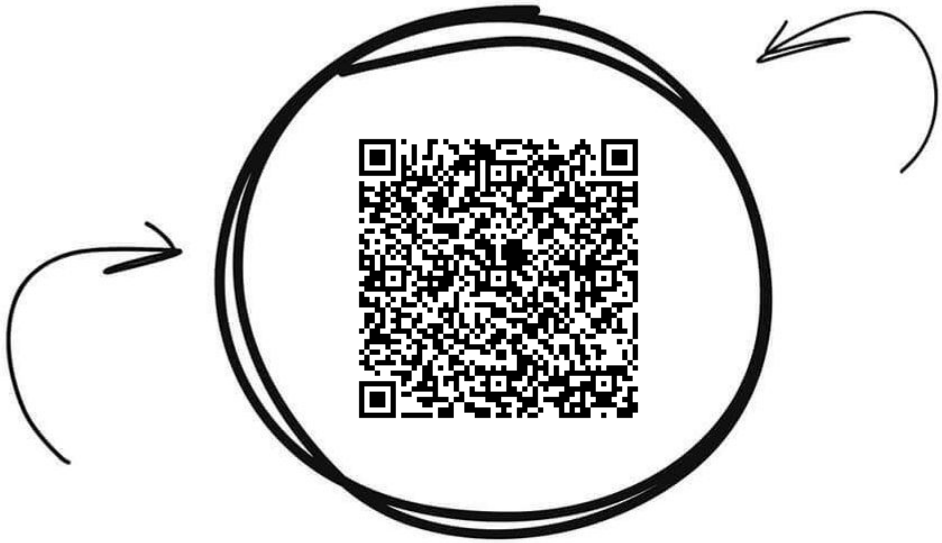
Abre la **Cuenta NoCuenta** con el código **WUOLAH10**, haz tu primer pago y llévate 10 €.

Quiero el cash

Consulta condiciones **aquí**

do your thing

Teoría de Los Lenguajes de P...



Banco de apuntes de la

WUOLAH

Comparte estos flyers en tu clase y consigue más dinero y recompensas

1. Imprime esta hoja
2. Recorta por la mitad
3. Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes
4. Llévate dinero por cada descarga de los documentos descargados a través de tu QR

While20.hs

```haskell
-- ------------------------------------------------------------------
-- While20.hs
-- ------------------------------------------------------------------

module While20 where

type  Var   =  String

data  Aexp  =  N Integer
            |  V Var
            |  Add Aexp Aexp
            |  Mult Aexp Aexp
            |  Sub Aexp Aexp
            |  Div Aexp Aexp
            deriving (Show, Eq)

data  Bexp  =  TRUE
            |  FALSE
            |  Eq Aexp Aexp
            |  Le Aexp Aexp
            |  Neg Bexp
            |  And Bexp Bexp
            deriving (Show, Eq)

data  Stm   =  Ass Var Aexp
            |  Skip
            |  Comp Stm Stm
            |  If Bexp Stm Stm
            |  Case Aexp LabelledStms
            |  Swap Var Var
            |  For Stm Bexp Stm Stm
            deriving Show

data LabelledStms = LabelledStm [Integer] Stm LabelledStms
                  | Default Stm
                  | EndLabelledStms
                  deriving Show

type  Z      =  Integer
type  T      =  Bool
type  State  =  Var -> Z

aVal :: Aexp -> State -> Z
aVal (N n) _        =  n
aVal (V x) s        =  s x
aVal (Add a1 a2) s  =  aVal a1 s + aVal a2 s
aVal (Sub a1 a2) s  =  aVal a1 s - aVal a2 s
aVal (Mult a1 a2) s =  aVal a1 s * aVal a2 s
aVal (Div a1 a2) s  =  aVal a1 s `div` aVal a2 s

bVal :: Bexp -> State -> T
```

```
52 bVal TRUE _         =  True
53 bVal FALSE _        =  False
54 bVal (Eq a1 a2) s   =  aVal a1 s == aVal a2 s
55 bVal (Le a1 a2) s   =  aVal a1 s <= aVal a2 s
56 bVal (Neg b) s      =  not(bVal b s)
57 bVal (And b1 b2) s =  bVal b1 s && bVal b2 s
58
```

While20.hs

```haskell
-- ------------------------------------------------------------------
-- While20.hs
-- ------------------------------------------------------------------

module While20 where

type  Var   =  String

data  Aexp  =  N Integer
            |  V Var
            |  Add Aexp Aexp
            |  Mult Aexp Aexp
            |  Sub Aexp Aexp
            |  Div Aexp Aexp
            deriving (Show, Eq)

data  Bexp  =  TRUE
            |  FALSE
            |  Eq Aexp Aexp
            |  Le Aexp Aexp
            |  Neg Bexp
            |  And Bexp Bexp
            deriving (Show, Eq)

data  Stm   =  Ass Var Aexp
            |  Skip
            |  Comp Stm Stm
            |  If Bexp Stm Stm
            |  Case Aexp LabelledStms
            |  Swap Var Var
            |  For Stm Bexp Stm Stm
            deriving Show

data LabelledStms = LabelledStm [Integer] Stm LabelledStms
                  | Default Stm
                  | EndLabelledStms
                  deriving Show

type  Z      =  Integer
type  T      =  Bool
type  State  =  Var -> Z

aVal :: Aexp -> State -> Z
aVal (N n) _        =  n
aVal (V x) s        =  s x
aVal (Add a1 a2) s  =  aVal a1 s + aVal a2 s
aVal (Sub a1 a2) s  =  aVal a1 s - aVal a2 s
aVal (Mult a1 a2) s =  aVal a1 s * aVal a2 s
aVal (Div a1 a2) s  =  aVal a1 s `div` aVal a2 s

bVal :: Bexp -> State -> T
```

```
52 bVal TRUE _         =  True
53 bVal FALSE _        =  False
54 bVal (Eq a1 a2) s   =  aVal a1 s == aVal a2 s
55 bVal (Le a1 a2) s   =  aVal a1 s <= aVal a2 s
56 bVal (Neg b) s      =  not(bVal b s)
57 bVal (And b1 b2) s =  bVal b1 s && bVal b2 s
58
```

While20.hs

```haskell
-- ------------------------------------------------------------------
-- While20.hs
-- ------------------------------------------------------------------

module While20 where

type  Var   =  String

data  Aexp  =  N Integer
            |  V Var
            |  Add Aexp Aexp
            |  Mult Aexp Aexp
            |  Sub Aexp Aexp
            |  Div Aexp Aexp
            deriving (Show, Eq)

data  Bexp  =  TRUE
            |  FALSE
            |  Eq Aexp Aexp
            |  Le Aexp Aexp
            |  Neg Bexp
            |  And Bexp Bexp
            deriving (Show, Eq)

data  Stm   =  Ass Var Aexp
            |  Skip
            |  Comp Stm Stm
            |  If Bexp Stm Stm
            |  Case Aexp LabelledStms
            |  Swap Var Var
            |  For Stm Bexp Stm Stm
            deriving Show

data LabelledStms = LabelledStm [Integer] Stm LabelledStms
                  | Default Stm
                  | EndLabelledStms
                  deriving Show

type  Z      =  Integer
type  T      =  Bool
type  State  =  Var -> Z

aVal :: Aexp -> State -> Z
aVal (N n) _        =  n
aVal (V x) s        =  s x
aVal (Add a1 a2) s  =  aVal a1 s + aVal a2 s
aVal (Sub a1 a2) s  =  aVal a1 s - aVal a2 s
aVal (Mult a1 a2) s =  aVal a1 s * aVal a2 s
aVal (Div a1 a2) s  =  aVal a1 s `div` aVal a2 s

bVal :: Bexp -> State -> T
```

```
52 bVal TRUE _         =  True
53 bVal FALSE _        =  False
54 bVal (Eq a1 a2) s   =  aVal a1 s == aVal a2 s
55 bVal (Le a1 a2) s   =  aVal a1 s <= aVal a2 s
56 bVal (Neg b) s      =  not(bVal b s)
57 bVal (And b1 b2) s =  bVal b1 s && bVal b2 s
58
```

While20.hs

```haskell
-- ------------------------------------------------------------------
-- While20.hs
-- ------------------------------------------------------------------

module While20 where

type  Var   =  String

data  Aexp  =  N Integer
            |  V Var
            |  Add Aexp Aexp
            |  Mult Aexp Aexp
            |  Sub Aexp Aexp
            |  Div Aexp Aexp
            deriving (Show, Eq)

data  Bexp  =  TRUE
            |  FALSE
            |  Eq Aexp Aexp
            |  Le Aexp Aexp
            |  Neg Bexp
            |  And Bexp Bexp
            deriving (Show, Eq)

data  Stm   =  Ass Var Aexp
            |  Skip
            |  Comp Stm Stm
            |  If Bexp Stm Stm
            |  Case Aexp LabelledStms
            |  Swap Var Var
            |  For Stm Bexp Stm Stm
            deriving Show

data LabelledStms = LabelledStm [Integer] Stm LabelledStms
                  | Default Stm
                  | EndLabelledStms
                  deriving Show

type  Z      =  Integer
type  T      =  Bool
type  State  =  Var -> Z

aVal :: Aexp -> State -> Z
aVal (N n) _        =  n
aVal (V x) s        =  s x
aVal (Add a1 a2) s  =  aVal a1 s + aVal a2 s
aVal (Sub a1 a2) s  =  aVal a1 s - aVal a2 s
aVal (Mult a1 a2) s =  aVal a1 s * aVal a2 s
aVal (Div a1 a2) s  =  aVal a1 s `div` aVal a2 s

bVal :: Bexp -> State -> T
```

```
52 bVal TRUE _         =  True
53 bVal FALSE _        =  False
54 bVal (Eq a1 a2) s   =  aVal a1 s == aVal a2 s
55 bVal (Le a1 a2) s   =  aVal a1 s <= aVal a2 s
56 bVal (Neg b) s      =  not(bVal b s)
57 bVal (And b1 b2) s =  bVal b1 s && bVal b2 s
58
```

While20.hs

```
1  -- ------------------------------------------------------------------
2  -- While20.hs
3  -- ------------------------------------------------------------------
4
5  module While20 where
6
7  type  Var   =  String
8
9  data  Aexp  =  N Integer
10             |  V Var
11             |  Add Aexp Aexp
12             |  Mult Aexp Aexp
13             |  Sub Aexp Aexp
14             |  Div Aexp Aexp
15            deriving (Show, Eq)
16
17 data  Bexp  =  TRUE
18             |  FALSE
19             |  Eq Aexp Aexp
20             |  Le Aexp Aexp
21             |  Neg Bexp
22             |  And Bexp Bexp
23            deriving (Show, Eq)
24
25 data  Stm   =  Ass Var Aexp
26             |  Skip
27             |  Comp Stm Stm
28             |  If Bexp Stm Stm
29             |  Case Aexp LabelledStms
30             |  Swap Var Var
31             |  For Stm Bexp Stm Stm
32            deriving Show
33
34 data LabelledStms = LabelledStm [Integer] Stm LabelledStms
35                  | Default Stm
36                  | EndLabelledStms
37                 deriving Show
38
39 type  Z      =  Integer
40 type  T      =  Bool
41 type  State  =  Var -> Z
42
43 aVal :: Aexp -> State -> Z
44 aVal (N n) _        =  n
45 aVal (V x) s        =  s x
46 aVal (Add a1 a2) s  =  aVal a1 s + aVal a2 s
47 aVal (Sub a1 a2) s  =  aVal a1 s - aVal a2 s
48 aVal (Mult a1 a2) s =  aVal a1 s * aVal a2 s
49 aVal (Div a1 a2) s  =  aVal a1 s `div` aVal a2 s
50
51 bVal :: Bexp -> State -> T
```

```
52 bVal TRUE _         =  True
53 bVal FALSE _        =  False
54 bVal (Eq a1 a2) s   =  aVal a1 s == aVal a2 s
55 bVal (Le a1 a2) s   =  aVal a1 s <= aVal a2 s
56 bVal (Neg b) s      =  not(bVal b s)
57 bVal (And b1 b2) s =  bVal b1 s && bVal b2 s
58
```

While20.hs

```haskell
-- ------------------------------------------------------------------
-- While20.hs
-- ------------------------------------------------------------------

module While20 where

type  Var   =  String

data  Aexp  =  N Integer
            |  V Var
            |  Add Aexp Aexp
            |  Mult Aexp Aexp
            |  Sub Aexp Aexp
            |  Div Aexp Aexp
            deriving (Show, Eq)

data  Bexp  =  TRUE
            |  FALSE
            |  Eq Aexp Aexp
            |  Le Aexp Aexp
            |  Neg Bexp
            |  And Bexp Bexp
            deriving (Show, Eq)

data  Stm   =  Ass Var Aexp
            |  Skip
            |  Comp Stm Stm
            |  If Bexp Stm Stm
            |  Case Aexp LabelledStms
            |  Swap Var Var
            |  For Stm Bexp Stm Stm
            deriving Show

data LabelledStms = LabelledStm [Integer] Stm LabelledStms
                  | Default Stm
                  | EndLabelledStms
                  deriving Show

type  Z      =  Integer
type  T      =  Bool
type  State  =  Var -> Z

aVal :: Aexp -> State -> Z
aVal (N n) _        =  n
aVal (V x) s        =  s x
aVal (Add a1 a2) s  =  aVal a1 s + aVal a2 s
aVal (Sub a1 a2) s  =  aVal a1 s - aVal a2 s
aVal (Mult a1 a2) s =  aVal a1 s * aVal a2 s
aVal (Div a1 a2) s  =  aVal a1 s `div` aVal a2 s

bVal :: Bexp -> State -> T
```

```
52 bVal TRUE _         =  True
53 bVal FALSE _        =  False
54 bVal (Eq a1 a2) s   =  aVal a1 s == aVal a2 s
55 bVal (Le a1 a2) s   =  aVal a1 s <= aVal a2 s
56 bVal (Neg b) s      =  not(bVal b s)
57 bVal (And b1 b2) s =  bVal b1 s && bVal b2 s
58
```

While20.hs

```haskell
-- ------------------------------------------------------------------
-- While20.hs
-- ------------------------------------------------------------------

module While20 where

type  Var   =  String

data  Aexp  =  N Integer
            |  V Var
            |  Add Aexp Aexp
            |  Mult Aexp Aexp
            |  Sub Aexp Aexp
            |  Div Aexp Aexp
            deriving (Show, Eq)

data  Bexp  =  TRUE
            |  FALSE
            |  Eq Aexp Aexp
            |  Le Aexp Aexp
            |  Neg Bexp
            |  And Bexp Bexp
            deriving (Show, Eq)

data  Stm   =  Ass Var Aexp
            |  Skip
            |  Comp Stm Stm
            |  If Bexp Stm Stm
            |  Case Aexp LabelledStms
            |  Swap Var Var
            |  For Stm Bexp Stm Stm
            deriving Show

data LabelledStms = LabelledStm [Integer] Stm LabelledStms
                  | Default Stm
                  | EndLabelledStms
                  deriving Show

type  Z      =  Integer
type  T      =  Bool
type  State  =  Var -> Z

aVal :: Aexp -> State -> Z
aVal (N n) _        =  n
aVal (V x) s        =  s x
aVal (Add a1 a2) s  =  aVal a1 s + aVal a2 s
aVal (Sub a1 a2) s  =  aVal a1 s - aVal a2 s
aVal (Mult a1 a2) s =  aVal a1 s * aVal a2 s
aVal (Div a1 a2) s  =  aVal a1 s `div` aVal a2 s

bVal :: Bexp -> State -> T
```

```
52 bVal TRUE _         =  True
53 bVal FALSE _        =  False
54 bVal (Eq a1 a2) s   =  aVal a1 s == aVal a2 s
55 bVal (Le a1 a2) s   =  aVal a1 s <= aVal a2 s
56 bVal (Neg b) s      =  not(bVal b s)
57 bVal (And b1 b2) s =  bVal b1 s && bVal b2 s
58
```

While20.hs

```haskell
-- ------------------------------------------------------------------
-- While20.hs
-- ------------------------------------------------------------------

module While20 where

type  Var   =  String

data  Aexp  =  N Integer
            |  V Var
            |  Add Aexp Aexp
            |  Mult Aexp Aexp
            |  Sub Aexp Aexp
            |  Div Aexp Aexp
            deriving (Show, Eq)

data  Bexp  =  TRUE
            |  FALSE
            |  Eq Aexp Aexp
            |  Le Aexp Aexp
            |  Neg Bexp
            |  And Bexp Bexp
            deriving (Show, Eq)

data  Stm   =  Ass Var Aexp
            |  Skip
            |  Comp Stm Stm
            |  If Bexp Stm Stm
            |  Case Aexp LabelledStms
            |  Swap Var Var
            |  For Stm Bexp Stm Stm
            deriving Show

data LabelledStms = LabelledStm [Integer] Stm LabelledStms
                  | Default Stm
                  | EndLabelledStms
                  deriving Show

type  Z      =  Integer
type  T      =  Bool
type  State  =  Var -> Z

aVal :: Aexp -> State -> Z
aVal (N n) _        =  n
aVal (V x) s        =  s x
aVal (Add a1 a2) s  =  aVal a1 s + aVal a2 s
aVal (Sub a1 a2) s  =  aVal a1 s - aVal a2 s
aVal (Mult a1 a2) s =  aVal a1 s * aVal a2 s
aVal (Div a1 a2) s  =  aVal a1 s `div` aVal a2 s

bVal :: Bexp -> State -> T
```

```
52 bVal TRUE _         =  True
53 bVal FALSE _        =  False
54 bVal (Eq a1 a2) s   =  aVal a1 s == aVal a2 s
55 bVal (Le a1 a2) s   =  aVal a1 s <= aVal a2 s
56 bVal (Neg b) s      =  not(bVal b s)
57 bVal (And b1 b2) s =  bVal b1 s && bVal b2 s
58
```

While20.hs

```haskell
-- ------------------------------------------------------------------
-- While20.hs
-- ------------------------------------------------------------------

module While20 where

type  Var   =  String

data  Aexp  =  N Integer
            |  V Var
            |  Add Aexp Aexp
            |  Mult Aexp Aexp
            |  Sub Aexp Aexp
            |  Div Aexp Aexp
            deriving (Show, Eq)

data  Bexp  =  TRUE
            |  FALSE
            |  Eq Aexp Aexp
            |  Le Aexp Aexp
            |  Neg Bexp
            |  And Bexp Bexp
            deriving (Show, Eq)

data  Stm   =  Ass Var Aexp
            |  Skip
            |  Comp Stm Stm
            |  If Bexp Stm Stm
            |  Case Aexp LabelledStms
            |  Swap Var Var
            |  For Stm Bexp Stm Stm
            deriving Show

data LabelledStms = LabelledStm [Integer] Stm LabelledStms
                  | Default Stm
                  | EndLabelledStms
                  deriving Show

type  Z      =  Integer
type  T      =  Bool
type  State  =  Var -> Z

aVal :: Aexp -> State -> Z
aVal (N n) _        =  n
aVal (V x) s        =  s x
aVal (Add a1 a2) s  =  aVal a1 s + aVal a2 s
aVal (Sub a1 a2) s  =  aVal a1 s - aVal a2 s
aVal (Mult a1 a2) s =  aVal a1 s * aVal a2 s
aVal (Div a1 a2) s  =  aVal a1 s `div` aVal a2 s

bVal :: Bexp -> State -> T
```

```
52 bVal TRUE _         =  True
53 bVal FALSE _        =  False
54 bVal (Eq a1 a2) s   =  aVal a1 s == aVal a2 s
55 bVal (Le a1 a2) s   =  aVal a1 s <= aVal a2 s
56 bVal (Neg b) s      =  not(bVal b s)
57 bVal (And b1 b2) s =  bVal b1 s && bVal b2 s
58
```

While20.hs

```haskell
-- ------------------------------------------------------------------
-- While20.hs
-- ------------------------------------------------------------------

module While20 where

type  Var   =  String

data  Aexp  =  N Integer
            |  V Var
            |  Add Aexp Aexp
            |  Mult Aexp Aexp
            |  Sub Aexp Aexp
            |  Div Aexp Aexp
            deriving (Show, Eq)

data  Bexp  =  TRUE
            |  FALSE
            |  Eq Aexp Aexp
            |  Le Aexp Aexp
            |  Neg Bexp
            |  And Bexp Bexp
            deriving (Show, Eq)

data  Stm   =  Ass Var Aexp
            |  Skip
            |  Comp Stm Stm
            |  If Bexp Stm Stm
            |  Case Aexp LabelledStms
            |  Swap Var Var
            |  For Stm Bexp Stm Stm
            deriving Show

data LabelledStms = LabelledStm [Integer] Stm LabelledStms
                  | Default Stm
                  | EndLabelledStms
                  deriving Show

type  Z      =  Integer
type  T      =  Bool
type  State  =  Var -> Z

aVal :: Aexp -> State -> Z
aVal (N n) _        =  n
aVal (V x) s        =  s x
aVal (Add a1 a2) s  =  aVal a1 s + aVal a2 s
aVal (Sub a1 a2) s  =  aVal a1 s - aVal a2 s
aVal (Mult a1 a2) s =  aVal a1 s * aVal a2 s
aVal (Div a1 a2) s  =  aVal a1 s `div` aVal a2 s

bVal :: Bexp -> State -> T
```

```
52 bVal TRUE _         =  True
53 bVal FALSE _        =  False
54 bVal (Eq a1 a2) s   =  aVal a1 s == aVal a2 s
55 bVal (Le a1 a2) s   =  aVal a1 s <= aVal a2 s
56 bVal (Neg b) s      =  not(bVal b s)
57 bVal (And b1 b2) s =  bVal b1 s && bVal b2 s
58
```

While20.hs

```haskell
-- ------------------------------------------------------------------
-- While20.hs
-- ------------------------------------------------------------------

module While20 where

type  Var   =  String

data  Aexp  =  N Integer
            |  V Var
            |  Add Aexp Aexp
            |  Mult Aexp Aexp
            |  Sub Aexp Aexp
            |  Div Aexp Aexp
            deriving (Show, Eq)

data  Bexp  =  TRUE
            |  FALSE
            |  Eq Aexp Aexp
            |  Le Aexp Aexp
            |  Neg Bexp
            |  And Bexp Bexp
            deriving (Show, Eq)

data  Stm   =  Ass Var Aexp
            |  Skip
            |  Comp Stm Stm
            |  If Bexp Stm Stm
            |  Case Aexp LabelledStms
            |  Swap Var Var
            |  For Stm Bexp Stm Stm
            deriving Show

data LabelledStms = LabelledStm [Integer] Stm LabelledStms
                  | Default Stm
                  | EndLabelledStms
                  deriving Show

type  Z      =  Integer
type  T      =  Bool
type  State  =  Var -> Z

aVal :: Aexp -> State -> Z
aVal (N n) _        =  n
aVal (V x) s        =  s x
aVal (Add a1 a2) s  =  aVal a1 s + aVal a2 s
aVal (Sub a1 a2) s  =  aVal a1 s - aVal a2 s
aVal (Mult a1 a2) s =  aVal a1 s * aVal a2 s
aVal (Div a1 a2) s  =  aVal a1 s `div` aVal a2 s

bVal :: Bexp -> State -> T
```

```
52 bVal TRUE _         =  True
53 bVal FALSE _        =  False
54 bVal (Eq a1 a2) s   =  aVal a1 s == aVal a2 s
55 bVal (Le a1 a2) s   =  aVal a1 s <= aVal a2 s
56 bVal (Neg b) s      =  not(bVal b s)
57 bVal (And b1 b2) s =  bVal b1 s && bVal b2 s
58
```

While20.hs

```haskell
-- ------------------------------------------------------------------
-- While20.hs
-- ------------------------------------------------------------------

module While20 where

type  Var   =  String

data  Aexp  =  N Integer
            |  V Var
            |  Add Aexp Aexp
            |  Mult Aexp Aexp
            |  Sub Aexp Aexp
            |  Div Aexp Aexp
            deriving (Show, Eq)

data  Bexp  =  TRUE
            |  FALSE
            |  Eq Aexp Aexp
            |  Le Aexp Aexp
            |  Neg Bexp
            |  And Bexp Bexp
            deriving (Show, Eq)

data  Stm   =  Ass Var Aexp
            |  Skip
            |  Comp Stm Stm
            |  If Bexp Stm Stm
            |  Case Aexp LabelledStms
            |  Swap Var Var
            |  For Stm Bexp Stm Stm
            deriving Show

data LabelledStms = LabelledStm [Integer] Stm LabelledStms
                  | Default Stm
                  | EndLabelledStms
                  deriving Show

type  Z      =  Integer
type  T      =  Bool
type  State  =  Var -> Z

aVal :: Aexp -> State -> Z
aVal (N n) _        =  n
aVal (V x) s        =  s x
aVal (Add a1 a2) s  =  aVal a1 s + aVal a2 s
aVal (Sub a1 a2) s  =  aVal a1 s - aVal a2 s
aVal (Mult a1 a2) s =  aVal a1 s * aVal a2 s
aVal (Div a1 a2) s  =  aVal a1 s `div` aVal a2 s

bVal :: Bexp -> State -> T
```

```
52 bVal TRUE _         =  True
53 bVal FALSE _        =  False
54 bVal (Eq a1 a2) s   =  aVal a1 s == aVal a2 s
55 bVal (Le a1 a2) s   =  aVal a1 s <= aVal a2 s
56 bVal (Neg b) s      =  not(bVal b s)
57 bVal (And b1 b2) s =  bVal b1 s && bVal b2 s
58
```

While20.hs

```haskell
-- ------------------------------------------------------------------
-- While20.hs
-- ------------------------------------------------------------------

module While20 where

type  Var   =  String

data  Aexp  =  N Integer
            |  V Var
            |  Add Aexp Aexp
            |  Mult Aexp Aexp
            |  Sub Aexp Aexp
            |  Div Aexp Aexp
            deriving (Show, Eq)

data  Bexp  =  TRUE
            |  FALSE
            |  Eq Aexp Aexp
            |  Le Aexp Aexp
            |  Neg Bexp
            |  And Bexp Bexp
            deriving (Show, Eq)

data  Stm   =  Ass Var Aexp
            |  Skip
            |  Comp Stm Stm
            |  If Bexp Stm Stm
            |  Case Aexp LabelledStms
            |  Swap Var Var
            |  For Stm Bexp Stm Stm
            deriving Show

data LabelledStms = LabelledStm [Integer] Stm LabelledStms
                  | Default Stm
                  | EndLabelledStms
                  deriving Show

type  Z      =  Integer
type  T      =  Bool
type  State  =  Var -> Z

aVal :: Aexp -> State -> Z
aVal (N n) _        =  n
aVal (V x) s        =  s x
aVal (Add a1 a2) s  =  aVal a1 s + aVal a2 s
aVal (Sub a1 a2) s  =  aVal a1 s - aVal a2 s
aVal (Mult a1 a2) s =  aVal a1 s * aVal a2 s
aVal (Div a1 a2) s  =  aVal a1 s `div` aVal a2 s

bVal :: Bexp -> State -> T
```

```
52 bVal TRUE _         =  True
53 bVal FALSE _        =  False
54 bVal (Eq a1 a2) s   =  aVal a1 s == aVal a2 s
55 bVal (Le a1 a2) s   =  aVal a1 s <= aVal a2 s
56 bVal (Neg b) s      =  not(bVal b s)
57 bVal (And b1 b2) s =  bVal b1 s && bVal b2 s
58
```

While20.hs

```haskell
-- ------------------------------------------------------------------
-- While20.hs
-- ------------------------------------------------------------------

module While20 where

type  Var   =  String

data  Aexp  =  N Integer
            |  V Var
            |  Add Aexp Aexp
            |  Mult Aexp Aexp
            |  Sub Aexp Aexp
            |  Div Aexp Aexp
            deriving (Show, Eq)

data  Bexp  =  TRUE
            |  FALSE
            |  Eq Aexp Aexp
            |  Le Aexp Aexp
            |  Neg Bexp
            |  And Bexp Bexp
            deriving (Show, Eq)

data  Stm   =  Ass Var Aexp
            |  Skip
            |  Comp Stm Stm
            |  If Bexp Stm Stm
            |  Case Aexp LabelledStms
            |  Swap Var Var
            |  For Stm Bexp Stm Stm
            deriving Show

data LabelledStms = LabelledStm [Integer] Stm LabelledStms
                  | Default Stm
                  | EndLabelledStms
                  deriving Show

type  Z      =  Integer
type  T      =  Bool
type  State  =  Var -> Z

aVal :: Aexp -> State -> Z
aVal (N n) _        =  n
aVal (V x) s        =  s x
aVal (Add a1 a2) s  =  aVal a1 s + aVal a2 s
aVal (Sub a1 a2) s  =  aVal a1 s - aVal a2 s
aVal (Mult a1 a2) s =  aVal a1 s * aVal a2 s
aVal (Div a1 a2) s  =  aVal a1 s `div` aVal a2 s

bVal :: Bexp -> State -> T
```

```
52 bVal TRUE _         =  True
53 bVal FALSE _        =  False
54 bVal (Eq a1 a2) s   =  aVal a1 s == aVal a2 s
55 bVal (Le a1 a2) s   =  aVal a1 s <= aVal a2 s
56 bVal (Neg b) s      =  not(bVal b s)
57 bVal (And b1 b2) s =  bVal b1 s && bVal b2 s
58
```