# binembed

Claude Heiland-Allen

2010–2015

# Contents



## 1 binembed.cabal

```
Name:                binembed
Version:             0.1.0.3
Synopsis:            Embed data into object files.
Description:         Given a list of files and directories to include,
                     binembed generates assembly source to include the data
                     into an object file that can be linked to a library or
                     executable, along with interface modules for higher
                     level access from languages such as C, Haskell, ...
                     .
                     See the package 'binembed-example' for a concrete
                     example.
Homepage:            http://code.mathr.co.uk/binembed
License:             BSD3
License-file:        LICENSE
Author:              Claude Heiland-Allen
Maintainer:          claude@mathr.co.uk
Category:            Distribution
Build-type:          Simple
Cabal-version:       >=1.6
Extra-source-files:  include/binembed.h

Library
  Build-depends:     base >= 4 && < 4.9,
                     bytestring >= 0.9 && < 0.11,
```

```
                      Cabal >= 1.8 && < 1.23
  Exposed-modules:    Data.BinEmbed
                      Distribution.Simple.BinEmbed
  include-dirs:       include
  install-includes:   binembed.h
  GHC-options:        -Wall

Executable binembed
  Build-depends:      base >= 4 && < 4.9,
                      containers >= 0 && < 0.6,
                      directory >= 1 && < 1.3,
                      dlist >= 0.4 && < 0.8,
                      filepath >= 1 && < 1.5
  Main-is:            src/binembed.hs
  GHC-options:        -Wall

Source-repository head
  type:     git
  location: http://code.mathr.co.uk/binembed.git

Source-repository this
  type:     git
  location: http://code.mathr.co.uk/binembed.git
  tag:      v0.1.0.3
```

# 2 Data/BinEmbed.hs

```
-----------------------------------------------------------------------------
-- |
-- Module      :  Data.BinEmbed
-- Copyright   :  Claude Heiland-Allen 2010
-- Maintainer  :  claude@mathr.co.uk
--
-- Support code used by the output of @binembed --output-hs=@.
--
-- For example, given @MyData.binembed@ listing some files, you might
-- get at the contents embedded into your executable using:
--
-- > import MyData   -- which re-exports this module
-- > main = do
-- >   myData' <- unBinEmbed myData
-- >   ...
--
-- See the 'binembed-example' package for a more detailed example.

module Data.BinEmbed
  ( Node(File, Dir)
  , unBinEmbed
  , unBinEmbedFile
  ) where

import Prelude hiding (sequence)

import Foreign.Ptr (Ptr, castPtr, minusPtr)
import Data.ByteString (ByteString)
import Data.ByteString.Unsafe (unsafePackCStringLen)
import Data.Foldable (Foldable, foldMap)
```

```
import Data.Traversable (Traversable, traverse, sequence)
import Data.Map (Map)

-- | A directory tree
data Node a
  = File a                      -- ^ A file has contents.
  | Dir (Map String (Node a))   -- ^ A directory has named @Node@s.
  deriving (Show, Read, Eq, Ord)

instance Functor Node where
  fmap f (File a) = File (           f  a)
  fmap f (Dir  a) = Dir  (fmap (fmap f) a)

instance Foldable Node where
  foldMap f (File a) =                  f  a
  foldMap f (Dir  a) = foldMap (foldMap f) a

instance Traversable Node where
  traverse f (File a) = File `fmap`                     f  a
  traverse f (Dir  a) = Dir  `fmap` traverse (traverse f) a

-- | Unpack embedded data.
unBinEmbed :: Node (IO ByteString) -> IO (Node ByteString)
unBinEmbed = sequence

-- | Repack the contents between two pointers.  Your code probably
--   doesn't need to call this, but it's needed in generated code.
{-  The usage in the code output by @binembed --output-hs=@ is safe,
--  because the embedded file data is in a .rodata section (ie, it is
--  immutable).
-}
unBinEmbedFile :: Ptr () -> Ptr () -> IO ByteString
unBinEmbedFile s e = unsafePackCStringLen (castPtr s, e `minusPtr` s)
```

# 3 Distribution/Simple/BinEmbed.hs

```
-----------------------------------------------------------------------------
-- |
-- Module      :  Distribution.Simple.BinEmbed
-- Copyright   :  Claude Heiland-Allen 2010
-- Maintainer  :  claude@mathr.co.uk
--
-- Support code to use @binembed@ as a pre-processor in Cabal.  For
-- example, your @Setup.hs@ might look like:
--
-- > import Distribution.Simple
-- > import Distribution.Simple.BinEmbed
-- > main = defaultMainWithHooks (withBinEmbed simpleUserHooks)
--
-- See the 'binembed-example' package for a more detailed example.

module Distribution.Simple.BinEmbed (withBinEmbed) where

import Distribution.Simple
import Distribution.Simple.LocalBuildInfo
import Distribution.Simple.PreProcess
import Distribution.Simple.Program
```

```
import Distribution.Simple.Setup
import Distribution.PackageDescription
import Data.Maybe (maybeToList)
import System.FilePath ((</>), dropExtension)

-- | Add hooks to use @binembed@ as a pre-processor, with input files
--   having the file name extension @.binembed@.  These hooks also
--   handle building the assembly output of @binembed@, as well as
--   cleaning it up afterwards.
withBinEmbed :: UserHooks -> UserHooks
withBinEmbed hooks = hooks
  { hookedPreProcessors = ("binembed", binembedPreProcessor) :
                          hookedPreProcessors hooks
  , hookedPrograms = binembedProgram : hookedPrograms hooks
  , buildHook = binembedBuild (buildHook hooks)
  , cleanHook = binembedClean (cleanHook hooks)
  }

--   @binembed@ executable.
binembedProgram :: Program
binembedProgram = simpleProgram "binembed"

--   The pre-processor invokes @binembed@ with sensible options, in
--   particular the output assembler source needs to be next to the data
--   files that it references.
binembedPreProcessor :: BuildInfo -> LocalBuildInfo -> PreProcessor
binembedPreProcessor _bi lbi = PreProcessor
  { platformIndependent = False
  , runPreProcessor = \(inBaseDir, inRelativeFile)
                       (outBaseDir, outRelativeFile)
                       verbosity -> do
      runDbProgram verbosity binembedProgram (withPrograms lbi) $
        [ "--output-hs=" ++ outBaseDir </> outRelativeFile
        , "--output-s=" ++ inBaseDir </> sfile (dropExtension outRelativeFile)
        , inBaseDir </> inRelativeFile
        ]
  }

--   Build by adding the output assembler to the C sources (this assumes
--   that the compiler used for C can also handle assembler sources; this
--   is true of GHC and GCC).
binembedBuild :: (PackageDescription -> LocalBuildInfo -> UserHooks -> ↙
    ↳ BuildFlags -> IO ())
              ->  PackageDescription -> LocalBuildInfo -> UserHooks -> ↙
                  ↳ BuildFlags -> IO ()
binembedBuild buildHook0 pd lbi hooks flags = do
  let pd' = pd{ executables = map (\e -> e{ buildInfo = f $ buildInfo e })
                              $ executables pd
              , library = fmap (\l -> l{ libBuildInfo = f $ libBuildInfo l })
                        $ library pd
              }
  buildHook0 pd' lbi hooks flags
  where
    f bi = case lookup binembedX $ customFieldsBI bi of
             Nothing -> bi
             Just be -> bi{ cSources = sfiles be ++ cSources bi }
```

```
--  Clean up the intermediary assembler files.
binembedClean :: (PackageDescription -> () -> UserHooks -> CleanFlags -> IO ())
              ->  PackageDescription -> () -> UserHooks -> CleanFlags -> IO ()
binembedClean cleanHook0 pd x hooks flags = do
  let pd' = pd{ extraTmpFiles = concat . concat $
                        [ map (f . buildInfo) (executables pd)
                        , map (f . libBuildInfo) (maybeToList $ library pd)
                        , [extraTmpFiles pd]
                        ]
              }
  cleanHook0 pd' x hooks flags
  where
    f bi = case lookup binembedX $ customFieldsBI bi of
             Nothing -> []
             Just be -> sfiles be

--  The 'build' and 'clean' hooks need to know which modules are really
--  generated by @binembed@ - use the 'x-binembed: ModuleName' field in
--  the @pkgname.cabal@ file to accomplish this.
binembedX :: String
binembedX = "x-binembed"

--  Munge a module name to assembler source file name; don't just add
--  an extension or the generated @.o@ file will clash with the @.o@
--  file generated from the Haskell output.
sfile :: String -> String
sfile = (++ "_be.s")

--  The same, but for more than one module.
sfiles :: String -> [String]
sfiles be = map sfile (words be)
```

# 4 example/binembed-example.cabal

```
Name:                binembed-example
Version:             0.1.0.3
Synopsis:            Example project using binembed to embed data in object ↙
    ↳ files.
Description:         binembed-example prints out its source code, embedded into ↙
    ↳ it
                     at compile time using the "binembed" package.

Homepage:            http://code.mathr.co.uk/binembed
License:             BSD3
License-file:        LICENSE
Author:              Claude Heiland-Allen
Maintainer:          claude@mathr.co.uk
Category:            Distribution
Build-type:          Custom
Cabal-version:       >=1.6

Extra-source-files:  main_c.c
                     Makefile
                     README

Executable binembed-example
  Main-is:             Main_haskell.hs
```

```
    Other-modules:       Example
    -- The next line is needed for the @withBinEmbed@ hooks to work:
    x-binembed:          Example
    Build-depends:       base >= 4 && < 4.9,
                         binembed >= 0.1 && < 0.2,
                         bytestring >= 0.9 && < 0.11,
                         containers >= 0.2 && < 0.6,
                         filepath >= 1 && < 1.5
    GHC-options:         -Wall

Source-repository head
  type:     git
  location: http://code.mathr.co.uk/binembed.git
  subdir:   example

Source-repository this
  type:     git
  location: http://code.mathr.co.uk/binembed.git
  subdir:   example
  tag:      v0.1.0.3
```

# 5 example/Example.binembed

```
Example.binembed
LICENSE
Main_haskell.hs
Makefile
README
Setup.hs
binembed-example.cabal
main_c.c
```

# 6 example/LICENSE

```
Copyright (c)2010, Claude Heiland-Allen

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.

    * Redistributions in binary form must reproduce the above
      copyright notice, this list of conditions and the following
      disclaimer in the documentation and/or other materials provided
      with the distribution.

    * Neither the name of Claude Heiland-Allen nor the names of other
      contributors may be used to endorse or promote products derived
      from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
```

# 7 example/main_c.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Example.h"

#define SEP "/"

void printEmbedded(char const *path, struct binembed_node const *node) {
  int pathLength = 0;
  char *pathBuffer = 0;
  for (; node->type != binembed_none; ++node) {
    switch (node->type) {
      case binembed_file:
        pathLength = strlen(path) + strlen(SEP) + strlen(node->name);
        pathBuffer = malloc(pathLength + 1);
        if (! pathBuffer) {
          exit(1);
        }
        sprintf(pathBuffer, "%s%s%s", path, SEP, node->name);
        printf("\n\n%s\n", pathBuffer);
        for (int i = 0; i < pathLength; ++i) {
          printf("-");
        }
        printf("\n\n{{{\n");
        struct binembed_file const *f = node->content.file;
        fflush(stdout);
        if (1 != fwrite(f->data, f->size, 1, stdout)) {
          exit(1);
        }
        fflush(stdout);
        printf("}}}\n");
        free(pathBuffer);
        pathBuffer = 0;
        break;
      case binembed_dir:
        pathLength = strlen(path) + strlen(SEP) + strlen(node->name);
        pathBuffer = malloc(pathLength + 1);
        if (! pathBuffer) {
          exit(1);
        }
        sprintf(pathBuffer, "%s%s%s", path, SEP, node->name);
        printEmbedded(pathBuffer, node->content.dir);
        free(pathBuffer);
        pathBuffer = 0;
        break;
      default:
```

```
            break;
        }
      }
    }
}

int main(int argc, char **argv) {
  printf("binembed-example\n");
  printf("=================\n");
  printEmbedded("binembed-example", Example);
  return 0;
}
```

# 8 example/Main_haskell.hs

```
import Prelude hiding  (putStr)
import Data.ByteString (putStr, ByteString)
import Data.Map (toAscList)
import System.FilePath ((</>))
import Example

main :: IO ()
main = do
  putStrLn "binembed-example\n\
           \================="
  printEmbedded "binembed-example" =<< unBinEmbed example

printEmbedded :: String -> Node ByteString -> IO ()
printEmbedded path (File f) = do
  putStrLn "\n"
  putStrLn path
  putStrLn (replicate (length path) '-')
  putStrLn ""
  putStrLn "{{{"
  putStr f
  putStrLn "}}}"

printEmbedded path (Dir d) = do
  mapM_ (\(p, n) -> printEmbedded (path </> p) n) (toAscList d)
```

# 9 example/Makefile

```
all: binembed-example

clean:
        rm -f binembed-example \
        Example.s Example.h Example.c

binembed-example: main_c.c Example.c Example.s
        gcc -std=c99 -Wall -pedantic -O2 -o $@ $^

%.s %.h %.c: %.binembed
        binembed --output-s=$*.s --output-h=$*.h --output-c=$*.c $*.binembed
```

# 10 example/README

```
binembed is designed to be as simple as possible to use.  This example
package contains two programs, each of which print out the sources of
the package.

Usage from Haskell:
  Example.binembed
  Main_haskell.hs
  Setup.hs
  binembed-example.cabal
  -- 'cabal sdist' is currently broken; use 'runhaskell Setup.hs sdist'

Usage from C:
  Example.binembed
  main_c.c
  Makefile
  -- 'make install' is currently unimplemented
```

## 11 example/Setup.hs

```
import Distribution.Simple
import Distribution.Simple.BinEmbed

main :: IO ()
main = defaultMainWithHooks (withBinEmbed simpleUserHooks)
```

## 12 .gitignore

```
.cabal-sandbox
cabal.sandbox.config
dist
example/Example_embed.hi
example/Example_embed.hs
example/Example_embed.o
example/Example_be.s
example/Main_haskell
example/Main_haskell.hi
example/Main_haskell.o
example/Example.c
example/Example.h
example/Example.s
example/binembed-example
```

## 13 include/binembed.h

```
#ifndef BINEMBED_H
#define BINEMBED_H 1
#include <stdint.h>
struct binembed_file {
  char const * const data;
  intptr_t const size;
};
struct binembed_node {
  char const * const name;
  enum { binembed_none = 0, binembed_file, binembed_dir } const type;
  union {
    void const * const none;
```

```
    struct binembed_file const * const file;
    struct binembed_node const * const dir;
  } const content;
};
#endif
```

# 14 LICENSE

```
Copyright (c)2010, Claude Heiland-Allen

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.

    * Redistributions in binary form must reproduce the above
      copyright notice, this list of conditions and the following
      disclaimer in the documentation and/or other materials provided
      with the distribution.

    * Neither the name of Claude Heiland-Allen nor the names of other
      contributors may be used to endorse or promote products derived
      from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

# 15 Setup.hs

```
import Distribution.Simple
main = defaultMain
```

# 16 src/binembed.hs

```
import Control.Monad (ap, liftM)
import Data.Char (ord, toUpper, toLower)
import Data.DList (DList, fromList, toList)
import Data.List (intercalate, isSuffixOf)
import Data.Map (Map)
import qualified Data.Map as M
import Data.Maybe (listToMaybe)
import Data.Monoid (Monoid, mempty, mappend, mconcat)
import System.Directory (doesDirectoryExist, doesFileExist, getDirectoryContents↙
    ↳ )
```

```
10  import System.Environment (getArgs)
    import System.FilePath ((</>), (<.>), joinPath, splitDirectories, splitExtension↙
        ↳ )

    import Paths_binembed (getLibDir)

15  {- file path -}

    type Name = String
    type Dir  = Map Name Node
    data Node = File | Dir Dir
20
    listRec :: FilePath -> FilePath -> IO (Name, Node)
    listRec t f = do
      let tf = t </> f
      isDir  <- doesDirectoryExist tf
25    isFile <- doesFileExist tf
      case (isDir, isFile) of
        (True, False) -> do
          cs <- mapM (listRec tf) =<< filter (not . hidden) `fmap` ↙
              ↳ getDirectoryContents tf
          return $ (f, Dir (M.fromList cs))
30      (False, True) -> do
          return $ (f, File)
      where
        hidden ('.':_) = True
        hidden _ = False
35
    depthFirstPostOrder :: Monad m => ([Name] -> m a) -> ([Name] -> [a] -> m a) -> [↙
        ↳ Name] -> Node -> m a
    depthFirstPostOrder fact _dact path File = fact path
    depthFirstPostOrder fact dact path (Dir dir) =
      mapM (\(name, node) -> depthFirstPostOrder fact dact (path ++ [name]) node) (M↙
          ↳ .toAscList dir) >>= dact path
40
    textMode :: [String] -> String -> Bool
    textMode exts file = any (`isSuffixOf` file) exts

    {- code output monoid -}
45
    type Code = DList Char
    data Output = Output{ oAS, oCH, oCC, oHS1, oHS2 :: Code }

    instance Monoid Output where
50    mempty = Output mempty mempty mempty mempty mempty
      Output as ch cc hs1 hs2 `mappend` Output as' ch' cc' hs1' hs2' =
        Output (as `mappend` as') (ch `mappend` ch') (cc `mappend` cc') (hs1 `↙
            ↳ mappend` hs1') (hs2 `mappend` hs2')

    {- code output monad -}
55
    data Env = Env{ eCHName, eCCName, eCInclude, eHSName :: String, eAlignment :: ↙
        ↳ Int, eTextExts :: [String] }
    data CodeGen a = CodeGen{ runCodeGen :: Env -> (a, Output) }

    instance Functor CodeGen where
60    fmap = liftM
```

```
instance Applicative CodeGen where
  pure = return
  (<*>) = ap

instance Monad CodeGen where
  return a = CodeGen $ \_e -> (a, mempty)
  CodeGen a >>= b = CodeGen $ \e -> let (aa, ao) = a e
                                        (bb, bo) = runCodeGen (b aa) e
                                    in  (bb, ao `mappend` bo)

execCodeGen :: Env -> CodeGen () -> Output
execCodeGen e (CodeGen f) = snd (f e)

tell :: Output -> CodeGen ()
tell out = CodeGen $ \_e -> ((), out)

asks :: (Env -> a) -> CodeGen a
asks x = CodeGen $ \e -> (x e, mempty)

{- symbol munging -}

symbols :: String -> String
symbols (s:ss) = symbol0 s ++ concatMap symbol ss

isSymbol0, isSymbol :: Char -> Bool
isSymbol0 c = ('a' <= c && c <= 'z') || ('A' <= c && c <= 'Z')
isSymbol c = isSymbol0 c || ('0' <= c && c <= '9')

escape, symbol0, symbol :: Char -> String
escape c = "_u" ++ show (ord c) ++ "_"
symbol0 c | isSymbol0 c = [c]
          | otherwise = escape c
symbol  c | isSymbol  c = [c]
          | otherwise = escape c

{- code generation -}

initialEnv :: [Char] -> Int -> [String] -> String -> Env
initialEnv name@(n:ame) alignment textExts include =
  Env
    { eCHName = map toUpper name
    , eCCName = name
    , eCInclude = include
    , eHSName = toUpper n : ame
    , eAlignment = alignment
    , eTextExts = textExts
    }

initialCode :: CodeGen ()
initialCode = do
  chName <- asks eCHName
  ccName <- asks eCCName
  cInc   <- asks eCInclude
  hsName <- asks eHSName
  tell Output
    { oAS = fromList $  "/*" ++ warning ++ "*/\n.section .rodata\n"
```

```
      , oCH = fromList $  "/*" ++ warning ++ "*/\n\
                          \#ifndef " ++ chName ++ "_H\n\
                          \#define " ++ chName ++ "_H 1\n\
                          \#include <" ++ cInc ++ ">\n"
      , oCC = fromList $  "/*" ++ warning ++ "*/\n\
                          \#include \"" ++ ccName <.> "h" ++ "\"\n"
      , oHS1 = fromList $ "{-# LANGUAGE ForeignFunctionInterface #-}\n\
                          \--" ++ warning ++ "\n\
                          \module " ++ hsName ++ "(\n  module Data.BinEmbed\n"
      , oHS2 = fromList $ ") where\n\
                          \import Foreign.Ptr (Ptr)\n\
                          \import Data.ByteString (ByteString)\n\
                          \import Data.Map (fromList)\n\
                          \import Data.BinEmbed\n"
      }
    where
      warning = " autogenerated file; do not edit "

finalCode :: CodeGen ()
finalCode = do
  tell Output
    { oAS = fromList $  ""
    , oCH = fromList $  "#endif\n"
    , oCC = fromList $  ""
    , oHS1 = fromList $ ""
    , oHS2 = fromList $ ""
    }

fileCode :: [String] -> CodeGen (Either (String, String) (String, String))
fileCode path = do
  align <- asks eAlignment
  exts <- asks eTextExts
  let f = joinPath . tail . splitDirectories . joinPath $ path
      s@(s0:ss)    = symbols . intercalate "/" . splitDirectories . joinPath $ ↙
        ↳ path
      e@(e0:es)    = s ++ "_end"
      z            = s ++ "_size"
      fs@(fs0:fss) = s ++ "_file"
      hs  = toLower s0  : ss
      he  = toLower e0  : es
      hfs = toLower fs0 : fss
      text = textMode exts (last path)
  tell Output
    { oAS  = fromList . unlines . map concat $
                          [ [".align ", show align]
                          , [".global ", s], [s, ":"]
                          , [".incbin ", show f]
                          , if text then [".byte 0"] else []
                          , [".global ", e], [e, ":"]
                          , [".global ", z], [".set ", z, ", ", e, " - ", s]
                          ]
    , oCH  = fromList $   "extern char const " ++ s ++ "[];\n\
                          \extern void const " ++ z ++ ";\n"
    , oCC  = fromList $   "static struct binembed_file const " ++ fs ++ " = {\n\
                          \  " ++ s ++ ", (intptr_t) &" ++ z ++ "\n};\n"
    , oHS1 = fromList $   "  , " ++ hfs ++ "\n"
    , oHS2 = fromList $   "foreign import stdcall \"&" ++ s ++ "\" " ++ hs ++ " ↙
```

```
        ↳ :: Ptr ()\n\
                          \foreign import stdcall \"&" ++ e ++ "\" " ++ he ++ " ↙
                              ↳ :: Ptr ()\n" ++
175                       hfs ++ " :: IO ByteString\n" ++
                          hfs ++ " = unBinEmbedFile " ++ hs ++ " " ++ he ++ "\n"
    }
  return $ Left (last path, fs)

180 dirCode :: Bool -> [String] -> [Either (String, String) (String, String)] -> ↙
    ↳ CodeGen (Either (String, String) (String, String))
dirCode extern path contents = do
  let dc@(dc0:dcs) = symbols . intercalate "/" . splitDirectories . joinPath $ ↙
      ↳ path
      hdc = toLower dc0 : dcs
  tell Output
185   { oAS  = fromList $ ""
    , oCH  = fromList $  if extern then "extern\            \ struct ↙
        ↳ binembed_node const " ++ dc ++ "[];\n" else ""
    , oCC  = fromList ( (if extern then "" else "static") ++ " struct ↙
        ↳ binembed_node const " ++ dc ++ "[] = {\n" ) `mappend`
          mconcat  ( map (\edf -> fromList $ case edf of
                          Left  (f, fs) -> "  { " ++ show f ++ ", binembed_file,↙
                              ↳  { .file = &" ++ fs ++ " } },\n"
190                       Right (d, ds) -> "  { " ++ show d ++ ", binembed_dir, ↙
                              ↳  { .dir  = " ++ ds ++ " } },\n"
                    ) contents )
        `mappend` fromList "  { 0, binembed_none, { .none = 0 } }\n};\n"
    , oHS1 = fromList $ if extern then "  , " ++ hdc ++ "\n" else ""
    , oHS2 = fromList ( hdc ++ " :: Node (IO ByteString)\n" ++ hdc ++ " = Dir . ↙
        ↳ fromList . tail $\n  [ undefined\n" ) `mappend`
          mconcat  ( map (\edf -> fromList $ case edf of
                          Left  (f, h:fs) -> "  , (" ++ show f ++ ", File " ++ (↙
                              ↳ toLower h : fs) ++ ")\n"
                          Right (d, h:ds) -> "  , (" ++ show d ++ ",      " ++ (↙
                              ↳ toLower h : ds) ++ ")\n"
                    ) contents )
        `mappend` fromList "  ]\n"
    }
  return $ Right (last path, dc)

{- main program -}

main :: IO ()
main = do
  args <- getArgs
  main' args

data Args = Args{ outputS, outputH, outputC, outputHS, infile :: [FilePath] }

instance Monoid Args where
  mempty = Args mempty mempty mempty mempty mempty
  Args a b c d e `mappend` Args x y z u v = Args (a`mappend`x) (b`mappend`y) (c`↙
      ↳ mappend`z) (d`mappend`u) (e`mappend`v)

toArg :: String -> Args
toArg ('-':'-':'o':'u':'t':'p':'u':'t':'-':'s':'=' :xs) = mempty{ outputS  =↙
    ↳  [xs] }
```

```
toArg ('-':'-':'o':'u':'t':'p':'u':'t':'-':'h':'=':xs) = mempty{ outputH  =↙
    ↪ [xs] }
toArg ('-':'-':'o':'u':'t':'p':'u':'t':'-':'c':'=':xs) = mempty{ outputC  =↙
    ↪ [xs] }
toArg ('-':'-':'o':'u':'t':'p':'u':'t':'-':'h':'s':'=':xs) = mempty{ outputHS =↙
    ↪ [xs] }
toArg a@('-':_) = error $ "binembed: bad argument: " ++ a
toArg xs = mempty{ infile = [xs] }

main' :: [String] -> IO ()
main' args = do
  let args' = mconcat . map toArg $ args
      infile' = listToMaybe . reverse . infile $ args'
      outs  = listToMaybe . reverse . outputS  $ args'
      outh  = listToMaybe . reverse . outputH  $ args'
      outc  = listToMaybe . reverse . outputC  $ args'
      ouths = listToMaybe . reverse . outputHS $ args'
      Just infile'' = infile'
      dirs = splitDirectories infile''
      top  = init dirs
      name = last dirs
      (stem, _ext) = splitExtension name
      textExts = []
  libDir <- getLibDir
  let include = libDir </> "include" </> "binembed.h"
  ins <- mapM (listRec (joinPath top)) =<< lines `fmap` readFile infile''
  let out = execCodeGen (initialEnv stem 64 textExts include) $ do
              initialCode
              tops <- mapM (\(name', node) -> depthFirstPostOrder fileCode (↙
                  ↪ dirCode False) [stem, name'] node) ins
              _ <- dirCode True [stem] tops
              finalCode
  case outs of
    Just outs' -> writeFile outs' (toList $ oAS out)
    Nothing    -> return ()
  case ouths of
    Just ouths' -> writeFile ouths' (toList $ oHS1 out `mappend` oHS2 out)
    Nothing     -> return ()
  case outc of
    Just outc' -> writeFile outc' (toList $ oCC out)
    Nothing    -> return ()
  case outh of
    Just outh' -> writeFile outh' (toList $ oCH out)
    Nothing    -> return ()
```