```
                           IO funkce v Prelude
                           -------------------
type  FilePath = String
data IOError    -- The internals of this type are system dependent
instance  Show IOError  where ...
instance  Eq IOError  where ...

ioError     :: IOError -> IO a
userError   :: String -> IOError
catch       :: IO a -> (IOError -> IO a) -> IO a
putChar     :: Char -> IO ()
putStr      :: String -> IO ()
putStrLn    :: String -> IO ()
print       :: Show a => a -> IO ()
getChar     :: IO Char
getLine     :: IO String
getContents:: IO String
interact    :: (String -> String) -> IO ()
readFile    :: FilePath -> IO String
writeFile   :: FilePath -> String -> IO ()
appendFile  :: FilePath -> String -> IO ()
readIO      :: Read a => String -> IO a          --raises an exception instead of an error
readIO s    =  case [x | (x,t) <- reads s, ("","") <- lex t] of
                 [x] -> return x
                 []  -> ioError (userError "Prelude.readIO: no parse")
                 _   -> ioError (userError "Prelude.readIO: ambiguous parse")
readLn      :: Read a => IO a

                                 Modul IO
                                 --------
data Handle = ...                        -- implementation-dependent
instance Eq Handle where;     instance Show Handle where;
data HandlePosn = ...                    -- implementation-dependent
instance Eq HandlePosn where; instance Show HandlePosn where;
data IOMode       =  ReadMode | WriteMode | AppendMode | ReadWriteMode
                     deriving (Eq, Ord, Ix, Bounded, Enum, Read, Show)
data BufferMode   =  NoBuffering | LineBuffering |  BlockBuffering (Maybe Int)
                     deriving (Eq, Ord, Read, Show)
data SeekMode     =  AbsoluteSeek | RelativeSeek | SeekFromEnd
                     deriving (Eq, Ord, Ix, Bounded, Enum, Read, Show)
stdin, stdout, stderr :: Handle

openFile              :: FilePath -> IOMode -> IO Handle
hClose                :: Handle -> IO ()

hFileSize             :: Handle -> IO Integer
hIsEOF                :: Handle -> IO Bool
isEOF                 :: IO Bool                {-isEOF = hIsEOF stdin-}

hSetBuffering         :: Handle  -> BufferMode -> IO ()
hGetBuffering         :: Handle  -> IO BufferMode
hFlush                :: Handle -> IO ()
hGetPosn              :: Handle -> IO HandlePosn
hSetPosn              :: HandlePosn -> IO ()
hSeek                 :: Handle -> SeekMode -> Integer -> IO ()

hWaitForInput         :: Handle -> Int -> IO Bool
hReady                :: Handle -> IO Bool      {-hReady h = hWaitForInput h 0-}
hGetChar              :: Handle -> IO Char
hGetLine              :: Handle -> IO String
hLookAhead            :: Handle -> IO Char
hGetContents          :: Handle -> IO String
hPutChar              :: Handle -> Char -> IO ()
hPutStr               :: Handle -> String -> IO ()
hPutStrLn             :: Handle -> String -> IO ()
hPrint                :: Show a => Handle -> a -> IO ()

hIsOpen               :: Handle -> IO Bool
hIsClosed             :: Handle -> IO Bool
```

```
hIsReadable            :: Handle -> IO Bool
hIsWritable            :: Handle -> IO Bool
hIsSeekable            :: Handle -> IO Bool

isAlreadyExistsError   :: IOError -> Bool
isDoesNotExistError    :: IOError -> Bool
isAlreadyInUseError    :: IOError -> Bool
isFullError            :: IOError -> Bool
isEOFError             :: IOError -> Bool
isIllegalOperation     :: IOError -> Bool
isPermissionError      :: IOError -> Bool
isUserError            :: IOError -> Bool

ioeGetErrorString      :: IOError -> String
ioeGetHandle           :: IOError -> Maybe Handle
ioeGetFileName         :: IOError -> Maybe FilePath

try                    :: IO a -> IO (Either IOError a)
bracket                :: IO a -> (a -> IO b) -> (a -> IO c) -> IO c
bracket_               :: IO a -> (a -> IO b) -> IO c -> IO c

                                    Modul Directory
                                    ---------------
data Permissions = Permissions {readable,writable,executable,searchable::Bool}
instance Eq   Permissions where...; instance Ord  Permissions where ...
instance Read Permissions where...; instance Show Permissions where ...

createDirectory      :: FilePath -> IO ()
removeDirectory      :: FilePath -> IO ()
removeFile           :: FilePath -> IO ()
renameDirectory      :: FilePath -> FilePath -> IO ()
renameFile           :: FilePath -> FilePath -> IO ()

getDirectoryContents:: FilePath -> IO [FilePath]
getCurrentDirectory :: IO FilePath
setCurrentDirectory :: FilePath -> IO ()

doesFileExist        :: FilePath -> IO Bool
doesDirectoryExist   :: FilePath -> IO Bool

getPermissions       :: FilePath -> IO Permissions
setPermissions       :: FilePath -> Permissions -> IO ()

getModificationTime :: FilePath -> IO ClockTime

                                     Modul System
                                     ------------
data ExitCode = ExitSuccess | ExitFailure Int deriving (Eq, Ord, Read, Show)

getArgs     :: IO [String]
getProgName :: IO String
getEnv      :: String -> IO String
system      :: String -> IO ExitCode
exitWith    :: ExitCode -> IO a
exitFailure :: IO a
                                     Modul CPUTime
                                     -------------
getCPUTime         :: IO Integer        {-v pikosekundach-}
cpuTimePrecision   :: Integer           {-nejmene meritelne-}

                           Monadické funkce z modulu Random
                           --------------------------------
class Random a where
   ...
   randomRIO :: (a,a) -> IO a
   randomIO  :: IO a
instance Random Int     where ...; instance Random Integer where ...
instance Random Float   where ...; instance Random Double  where ...
instance Random Bool    where ...; instance Random Char    where ...
```