```
%
% Abstract Data Types
%

% A Stack ADT using lists (open, declarative, unbundled)

declare NewStack Push Pop IsEmpty in
fun {NewStack} nil end
fun {Push S E} E|S end
fun {Pop S E} case S of  X|S1 then E = X  S1 end end
fun {IsEmpty S} S==nil end

local S1 S2 V in
   S1 = {Push {Push {NewStack} 1} 2}
   {Browse S1}
   S2 = {Pop S1 V}
   {Browse V}
   {Browse S2}
   case S2 of H|_ then {Browse H} end % misuse of ABSTRACT data type;
                                      % will not work with record
representation.
end

% Another Stack ADT Implementation using records
declare NewStack Push Pop IsEmpty in
fun {NewStack} emptyStack end
fun {Push S E} stack(E S) end
fun {Pop S E} case S of stack(X S1) then E = X S1 end end
fun {IsEmpty S} S==emptyStack end

local S1 S2 V in
   S1 = {Push {Push {NewStack} 1} 2}
   {Browse S1}
   S2 = {Pop S1 V}
   {Browse V}
   {Browse S2}
end

% A wrapper/unwrapper implementation using higher-order programming
declare NewWrapper in
proc {NewWrapper ?Wrap ?Unwrap}
        Key={NewName}
in
        fun {Wrap X}
             fun {$ K} if K==Key then X end end
        end
        fun {Unwrap C}
             {C Key}
        end
end

% A secure declarative unbundled Stack
declare NewStack Push Pop IsEmpty in
local Wrap Unwrap in
        {NewWrapper Wrap Unwrap}
        fun {NewStack} {Wrap nil} end
        fun {Push S E} {Wrap E|{Unwrap S}} end
        fun {Pop S E}
                case {Unwrap S} of X|S1 then E=X  {Wrap S1} end
        end
        fun {IsEmpty S} {Unwrap S}==nil end
end

local S1 S2 V in
   S1 = {Push {Push {NewStack} 1} 2}
   {Browse S1}
   S2 = {Pop S1 V}
   {Browse V}
   {Browse S2}
end

% A wrapper/unwrapper implementation using chunks (restricted records)
declare NewWrapper in
proc {NewWrapper ?Wrap ?Unwrap}
   Key={NewName}
in
   fun {Wrap X}
      {NewChunk foo(Key:X)}
   end
   fun {Unwrap C}
      C.Key
   end
end

declare NewStack Push Pop IsEmpty in
local Wrap Unwrap in
        {NewWrapper Wrap Unwrap}
        fun {NewStack} {Wrap nil} end
        fun {Push S E} {Wrap E|{Unwrap S}} end
        fun {Pop S E}
                case {Unwrap S} of X|S1 then E=X  {Wrap S1} end
        end
        fun {IsEmpty S} {Unwrap S}==nil end
end

local S1 S2 V in
   S1 = {Push {Push {NewStack} 1} 2}
   {Browse S1}
   S2 = {Pop S1 V}
   {Browse V}
   {Browse S2}
end

% exercise:  write a secure Stack implementation using chunks directly

% A wrapper/unwrapper implementation using ONLY higher-order programming
% courtesy of Jason Laporte
declare NewWrapper in
proc {NewWrapper ?Wrap ?Unwrap}
        Key=fun {$} 0 end
in
        fun {Wrap X}
            fun {$ K} if K==Key then X end end
        end
        fun {Unwrap C}
            {C Key}
        end
end

declare NewStack Push Pop IsEmpty in
local Wrap Unwrap in
        {NewWrapper Wrap Unwrap}
        fun {NewStack} {Wrap nil} end
        fun {Push S E} {Wrap E|{Unwrap S}} end
        fun {Pop S E}
                case {Unwrap S} of X|S1 then E=X  {Wrap S1} end
        end
        fun {IsEmpty S} {Unwrap S}==nil end
end

local S1 S2 V in
   S1 = {Push {Push {NewStack} 1} 2}
   {Browse S1}
   S2 = {Pop S1 V}
   {Browse V}
   {Browse S2}
   {Browse {S2 fun {$} 0 end}} % failed attempt to break into secure datatype
end
```