```
% Stack parameterized with maximal size and type
% of elements; for illustration an iterator and
% a restriction have been included.
mystack = datatype[maxn: int, t: type ]
  is new, pop, push, elements
    where t has default: proctype() returns(t)
  % restriction on element type
  rep = record { % representation; not visible outside
                 n: int;
                 a: array[t]
  }
  new := proc() returns(cvt) % return abstract type
    return({n: 0, a: array[t]$fill(maxn, t$default())})
    % record constructor invoking an array constructor
    % restriction on type parameter used here
  end % new
  pop = proc(s: cvt) returns(t) signals(empty)
    % s viewed as the concrete representation
    if s.n = 0 then signal(empty) end % raise exception
    s.n := s.n - 1
    return(s.a[n])
  end % pop
  push = proc(s: cvt, e: t) signals(full)
    if n = maxn then signal(full) end
    s.a[n] := e
    s.n := s.n + 1
  end % push
  elements = iter(s: cvt) yields(t) % an iterator
    for e: t in elements(:s.a)
    % invocation of an iterator; example of use below
        yield(t)
    end
  end % elements
end % stack

% Example of instantiating and using stack abstraction
a_stack = stack[100, int use default = int$maxint]
% Type constant used for type instance
as: a_stack := a_stack$new()
for i: int in [1 .. 10] do
  push(:as, i)
end
sum: int := 0
for i: int in elements(:as) do
  sum := sum + i
end
```