Contract a pattern
Solution

1 Class \texttt{MY\_LIST}

\hspace{1em} \texttt{note}
\hspace{1em} \texttt{description: "Lists without commitment to any particular representation."}
\hspace{1em} \texttt{author: ""}
\hspace{1em} \texttt{date: "$Date\$"}
\hspace{1em} \texttt{revision: "$Revision\$"}

\hspace{1em} \texttt{deferred class}
\hspace{1em} \texttt{MY\_LIST [G]}

\hspace{1em} \texttt{inherit}
\hspace{1em} \texttt{ANY}
\hspace{1em} \texttt{redefine}
\hspace{1em} \texttt{default\_create}
\hspace{1em} \texttt{end}

\hspace{1em} \texttt{feature -- Initialization}
\hspace{1em} \texttt{default\_create is}
\hspace{1em} \texttt{--- Create an empty list}
\hspace{1em} \texttt{do}
\hspace{1em} \texttt{ensure then}
\hspace{1em} \texttt{is\_empty: is\_empty}
\hspace{1em} \texttt{end}

\hspace{1em} \texttt{feature -- Status report}
\hspace{1em} \texttt{is\_empty: BOOLEAN is}
\hspace{1em} \texttt{--- Is the list empty?}
\hspace{1em} \texttt{deferred}
\hspace{1em} \texttt{end}

\hspace{1em} \texttt{has\_active\_iterators: BOOLEAN is}
\hspace{1em} \texttt{--- Are there any active iterators attached to Current?}
\hspace{1em} \texttt{do}
\hspace{1em} \texttt{Result := active\_iterator\_count > 0}
\hspace{1em} \texttt{end}
feature -- Element change
extend (x: G) is
   -- Add 'x' at the front
   require
      no_iterators: not has_active_iterators
defered
   ensure
      not_empty: not is_empty
end

remove is
   -- Remove an element from the front
   require
      not_empty: not is_empty
      no_iterators: not has_active_iterators
defered
end

feature -- Iteration
new_iterator: MY_ITERATOR |G| is
   -- New iterator attached to the list
defered
end

feature \{MY_ITERATOR\} -- Iteration
active_iterator_count : INTEGER
   -- Number of active iterators
activate_iterator is
   -- Record a new active iterator
   do
      active_iterator_count := active_iterator_count + 1
   ensure
      one_more_iterator: active_iterator_count = old active_iterator_count + 1
   end
deactivate_iterator is
   -- Record that an iterator became inactive
   require
      iterators_exist : active_iterator_count > 0
   do
      active_iterator_count := active_iterator_count - 1
   ensure
      one_less_iterator: active_iterator_count = old active_iterator_count - 1
   end
invariant
   iterator_count_non_negative: active_iterator_count >= 0
2 Class **MY_ITERATOR**

NOTE

description: "Iterators through lists."
author: ""
date: "$Date$"
revision: "$Revision$"

Deferred class

**MY_ITERATOR** \[G\]

Inherit

ANY

redefine
copy
end

**feature** \{NONE\} -- Initialization

make \((l: MY\_LIST[\mathbb{G}])\) is

-- Create an iterator attached to 'l'

require

\(l\_exists : l \neq \text{Void}\)
do

list := l
ensure

is_off : off
end

**feature** -- Access

item: \(G\) is

-- Current item

require

\(\text{not}\_off : \text{not}\ off\)
deferred
end

list : \(MY\_LIST[\mathbb{G}]\)

-- The list to which iterator is attached

**feature** -- Status report

off: \(\text{BOOLEAN}\) is

-- Is the iterator not pointing to any valid location in the list?
deferred
end

**feature** -- Basic operations

frozen start is

-- Start iteration
require
   is_off: off
do
   if not list.is_empty then
      list.activate_iterator
      internal_start
   end
ensure
   not_off_in_nonempty: not list.is_empty implies not off
   one_more_iterator_in_nonempty: not list.is_empty implies list.
      active_iterator_count = old list.active_iterator_count + 1
end

frozen forth is
   -- Go one step forward
require
   not_off: not off
do
   internal_forth
   if off then
      list.deactivate_iterator
   end
ensure
   one_less_iterator_if_off : off implies list.active_iterator_count =
      old list.active_iterator_count - 1
end

frozen go_off is
   -- Deactivate iteration
require
   not_off: not off
do
   internal_go_off
   list.deactivate_iterator
ensure
   is_off: off
   one_less_iterator : list.active_iterator_count = old list.
      active_iterator_count - 1
end

feature -- Copy
   copy (other: like Current) is
      -- Copy 'other'
   do
      if other /= Current then
         standard_copy (other)
         if not off then
            list.activate_iterator
         end
   end
ensure then
  same_list: list = other_list
  same_off: off = other_off
  same_item_if_not_off: not off implies item = other.item
  one_more_iterator_if_not_off: not off implies list.
  active_iterator_count = old_list.active_iterator_count + 1
end

feature {NONE} -- Implementation
  internal_start is
    -- Move cursor to the first element of 'list'
    require
      not_empty: not list.is_empty
    deferred
    ensure
      not_off: not off
    end

  internal_forth is
    -- Move cursor one step forward
    require
      not_off: not off
    deferred
    end

  internal_go_off is
    -- Make cursor point to no valid element of 'list'
    deferred
    ensure
      is_off: off
    end
end

3 Classes MY_LINKABLE and MY_LINKED_LIST

note
  description: "Linkable cells."
  author: ""
  date: "$Date$"
  revision: "$Revision$"

class
  MY_LINKABLE [G]

create
  put

feature -- Access
  item: G
    -- Content of cell
right: like Current
   -- Right neighbor

feature -- Element change
put (v: like item) is
   -- Make ‘v’ the cell’s ‘item’
do
   item := v
ensure
   item_inserted: item = v
end

put_right (other: like Current) is
   -- Put ‘other’ to the right of current cell
do
   right := other
ensure
   chained: right = other
end

note
   description: "Lists implemented as singly–linked lists."
   author: ""
   date: "$Date$"
   revision: "$Revision$"

class
   MY_LINKED_LIST [G]

inherit
   MY_LIST [G]

feature -- Status report
is_empty: BOOLEAN is
   -- Is the list empty?
do
   Result := (first = Void)
end

feature -- Element change
extend (x: G) is
   -- Add ‘x’ at the front
local
   new: MY_LINKABLE [G]
do
   create new.put (x)
   new.put_right (first)
   first := new
remove is
    -- Remove an element from the front
    do
      first := first . right
    end

feature -- Iteration
new_iterator: MY_LINKED_ITERATOR [G] is
    -- New iterator attached to current
    do
      create Result.make (Current)
    end

feature {MY_LINKED_ITERATOR} -- Implementation
first: MY_LINKABLE [G]
    -- First cell of the list
end

4 Class MY_LINKED_ITERATOR

note
  description: "Iterators through linked lists."
  author: ""
  date: "$Date$"
  revision: "$Revision$"

class
  MY_LINKED_ITERATOR [G]

inherit
  MY_ITERATOR [G]
  redefine
    list
  end

create {MY_LINKED_LIST}
make

feature -- Access
item: G is
    -- Current item
    do
      Result := active.item
    end

list: MY_LINKED_LIST [G]
    -- The list to which iterator is attached
feature -- Status report
  off : BOOLEAN is
    -- Is the iterator not pointing to any valid location in the list?
    do
      Result := (active = Void)
    end

feature {NONE} -- Implementation
  active: MY_LINKABLE [G]
    -- Current list cell

  internal_start is
    -- Move cursor to the first element of 'list'
    do
      active := list.first
    end

  internal_forth is
    -- Move cursor one step forward
    do
      active := active.right
    end

  internal_go_off is
    -- Make cursor point to no valid element of 'list'
    do
      active := Void
    end
end

5 Class MY_ARRAYED_LIST

note
description: "Lists that store their elements in arrays."
author: ""
date: "$Date$"
revision: "$Revision$"

class
  MY_ARRAYED_LIST [G]

inherit
  MY_LIST [G]
  redefine
    default_create
  end

feature -- Initialization
  default_create is
    -- Create an empty list
do
    create array.make (1, default_capacity)
end

feature -- Status report
is_empty: BOOLEAN is
    -- Is the list empty?
do
    Result := (first_index = 0)
end

feature -- Element change
extend (x: G) is
    -- Add 'x' at the front
do
    if first_index = array.count then
        array.conservative_resize (1, array.count + default_capacity)
    end
    first_index := first_index + 1;
    array.put (x, first_index);
end

remove is
    -- Remove an element from the front
do
    first_index := first_index - 1;
end

feature -- Iteration
new_iterator: MY_ARRAYED_ITERATOR [G] is
    -- New iterator attached to current
do
    create Result.make (Current)
end

feature {MY_ARRAYED_ITERATOR} -- Implementation
array: ARRAY [G]
    -- Array to store list elements
    first_index: INTEGER
        -- Index of the first list element in the array
default_capacity: INTEGER is 10
end

6 Class **MY_ARRAYED_ITERATOR**

**note**
description: "Iterators through arrayed lists."
author: ""
class
    MY_ARRAYED_ITERATOR [G]

inherit
    MY_ITERATOR [G]
    redefine
        list
    end

create {MY_ARRAYED_LIST}
make

feature -- Access
    item: G is
        -- Current item
        do
            Result := list.array [index]
        end

    list : MY_ARRAYED_LIST [G]
        -- The list to which iterator is attached

feature -- Status report
    off : BOOLEAN is
        -- Is the iterator not pointing to any valid location in the list?
        do
            Result := (index = 0)
        end

feature {NONE} -- Implementation
    index: INTEGER
        -- Index of current element

    internal_start is
        -- Move cursor to the first element of 'list'
        do
            index := list.first.index
        end

    internal_forth is
        -- Move cursor one step forward
        do
            index := index - 1
        end

    internal_go_off is
        -- Make cursor point to no valid element of 'list'

do
  index := 0
end

invariant
index_not_too_small: index >= 0
index_not_too_large: index <= list.first_index
end