```
note

    description:

        "Lists implemented with linked cells"

    library:    "Gobo Eiffel Lexical Library"
    copyright:  "Copyright (c) 1999, Eric Bezault and others"
    license:    "MIT License"

class SAMPLE [G]

inherit

    DS_LIST [G]
        redefine
            has, occurrences, swap
        end

    KL_IMPORTED_ANY_ROUTINES

create

    make, make_equal, make_from_linear

feature {NONE} -- Initialization

    make
            -- Create an empty list.
            -- Use ‘=’ as comparison criterion.
        do
            internal_cursor := new_cursor
        ensure
            empty: is_empty
            before: before
        end

    make_equal
            -- Create an empty list.
            -- Use ‘equal’ as comparison criterion.
        do
            internal_cursor := new_cursor
            create equality_tester
        ensure
            empty: is_empty
            before: before
        end

    make_from_linear (other: DS_LINEAR [G])
            -- Create a new list and fill it with items of ‘other’.
            -- Use ‘=’ as comparison criterion.
        require
            other_not_void: other /= Void
        local
            new_cell, new_last: like first_cell
            other_cursor: DS_LINEAR_CURSOR [G]
        do
            make
            if not other.is_empty then
                from
                    other_cursor := other.new_cursor
                    other_cursor.start
                    create first_cell.make (other_cursor.item)
                    new_last := first_cell
                    other_cursor.forth
                until
                    other_cursor.after
                loop
                    create new_cell.make (other_cursor.item)
                    new_last.put_right (new_cell)
                    new_last := new_cell
                    other_cursor.forth
                end
                last_cell := new_last
                count := other.count
            end
        ensure
            count_set: count = other.count
            before: before
        end

feature -- Access

    item, at alias "@" (i: INTEGER): G
            -- Item at index ‘i’
            -- (Performance: O(i).)
        local
            a_cell: like first_cell
            j: INTEGER
        do
            a_cell := first_cell
            from j := 1 until j = i loop
                a_cell := a_cell.right
                j := j + 1
            end
            Result := a_cell.item
        end

    first: G
            -- First item in list
            -- (Performance: O(1).)
        do
            Result := first_cell.item
        end

    last: G
            -- Last item in list
            -- (Performance: O(1).)
        do
            Result := last_cell.item
        end

    new_cursor: DS_LINKED_LIST_CURSOR [G]
            -- New external cursor for traversal
        do
            create Result.make (Current)
        end

feature -- Measurement

    count: INTEGER
```

```
            -- Number of items in list
            -- (Performance: O(1).)

    occurrences (v: G): INTEGER
            -- Number of times ‘v’ appears in list
            -- (Use ‘equality_tester’’s comparison criterion
            -- if not void, use ‘=’ criterion otherwise.)
        local
            a_cell: like first_cell
            a_tester: like equality_tester
        do
            a_cell := first_cell
            a_tester := equality_tester
            if a_tester /= Void then
                from until (a_cell = Void) loop
                    if a_tester.test (a_cell.item, v) then
                        Result := Result + 1
                    end
                    a_cell := a_cell.right
                end
            else
                    -- Use ‘=’ as comparison criterion.
                from until (a_cell = Void) loop
                    if a_cell.item = v then
                        Result := Result + 1
                    end
                    a_cell := a_cell.right
                end
            end
        end

feature -- Status report

    has (v: G): BOOLEAN
            -- Does list include ‘v’?
            -- (Use ‘equality_tester’’s comparison criterion
            -- if not void, use ‘=’ criterion otherwise.)
        local
            a_cell: like first_cell
            a_tester: like equality_tester
        do
            a_cell := first_cell
            a_tester := equality_tester
            if a_tester /= Void then
                from until (a_cell = Void) loop
                    if a_tester.test (a_cell.item, v) then
                        Result := True
                        a_cell := Void  -- Jump out of the loop.
                    else
                        a_cell := a_cell.right
                    end
                end
            else
                    -- Use ‘=’ as comparison criterion.
                from until (a_cell = Void) loop
                    if a_cell.item = v then
                        Result := True
                        a_cell := Void  -- Jump out of the loop.
                    else
                        a_cell := a_cell.right
                    end
                end
            end
        end

    extendible (n: INTEGER): BOOLEAN
            -- May list be extended with ‘n’ items?
        do
            Result := True
        ensure then
            definition: Result = True
        end

feature -- Duplication

    copy (other: like Current)
            -- Copy ‘other’ to current list.
            -- Move all cursors ‘off’ (unless ‘other = Current’).
            -- (Performance: O(other.count).)
        local
            a_cell, new_cell, new_last: like first_cell
            old_cursor: like new_cursor
        do
            if other /= Current then
                old_cursor := internal_cursor
                move_all_cursors_after
                standard_copy (other)
                if old_cursor = Void then
                        -- This may happen when ‘copy’ is called from ‘clone’
                        -- and the target has not been properly initialized.
                    internal_cursor := new_cursor
                else
                    internal_cursor := old_cursor
                end
                if not other.is_empty then
                    from
                        a_cell := other.first_cell
                        create first_cell.make (a_cell.item)
                        new_last := first_cell
                        a_cell := a_cell.right
                    until
                        a_cell = Void
                    loop
                        create new_cell.make (a_cell.item)
                        new_last.put_right (new_cell)
                        new_last := new_cell
                        a_cell := a_cell.right
                    end
                    last_cell := new_last
                end
            end
        end

feature -- Comparison

    is_equal (other: like Current): BOOLEAN
            -- Is list equal to ‘other’?
            -- Do not take cursor positions nor
            -- ‘equality_tester’ into account.
            -- (Performance: O(count).)
```

```
        local
            a_cell, other_cell: like first_cell
        do
            if Current = other then
                Result := True
            elseif ANY_.same_types (Current, other) and other.count = count then
                from
                    a_cell := first_cell
                    other_cell := other.first_cell
                    Result := True
                until
                    a_cell = Void
                loop
                    if a_cell.item /= other_cell.item then
                        Result := False
                        a_cell := Void  -- Jump out of the loop.
                    else
                        a_cell := a_cell.right
                        other_cell := other_cell.right
                    end
                end
            end
        end

feature -- Element change

    put_first, force_first (v: G)
            -- Add ‘v’ to beginning of list.
            -- Do not move cursors.
            -- (Performance: O(1).)
        local
            old_cell: like first_cell
        do
            if is_empty then
                create first_cell.make (v)
                last_cell := first_cell
                count := 1
            else
                old_cell := first_cell
                create first_cell.make (v)
                first_cell.put_right (old_cell)
                count := count + 1
            end
        end

    put_last, force_last (v: G)
            -- Add ‘v’ to end of list.
            -- Do not move cursors.
            -- (Performance: O(1).)
        local
            old_cell: like first_cell
        do
            if is_empty then
                create first_cell.make (v)
                last_cell := first_cell
                count := 1
            else
                old_cell := last_cell
                create last_cell.make (v)
                old_cell.put_right (last_cell)
                count := count + 1
            end
        end

    replace (v: G; i: INTEGER)
            -- Replace item at index ‘i’ by ‘v’.
            -- Do not move cursors.
            -- (Performance: O(i).)
        local
            a_cell: like first_cell
            j: INTEGER
        do
            a_cell := first_cell
            from j := 1 until j = i loop
                a_cell := a_cell.right
                j := j + 1
            end
            a_cell.put (v)
        end

    put, force (v: G; i: INTEGER)
            -- Add ‘v’ at ‘i’-th position.
            -- Do not move cursors.
            -- (Performance: O(i).)
        local
            a_cell, new_cell: like first_cell
            j: INTEGER
        do
            if i = 1 then
                put_first (v)
            elseif i = count + 1 then
                put_last (v)
            else
                    -- Go to cell at index ‘i-1’.
                a_cell := first_cell
                from j := 2 until j = i loop
                    a_cell := a_cell.right
                    j := j + 1
                end
                    -- The cell at index ‘i-1’ exists (because i /= 1)
                    -- and has a right neighbor (because i /= count + 1).
                    -- Insert a new cell in-between.
                create new_cell.make (v)
                new_cell.put_right (a_cell.right)
                a_cell.put_right (new_cell)
                count := count + 1
            end
        end

    put_left_cursor, force_left_cursor (v: G; a_cursor: like new_cursor)
            -- Add ‘v’ to left of ‘a_cursor’ position.
            -- Do not move cursors.
            -- (Synonym of ‘a_cursor.put_left (v)’.)
            -- (Performance: O(1).)
        local
            a_cell, new_cell, new_right: like first_cell
        do
            if a_cursor.after then
                put_last (v)
            elseif a_cursor.is_first then
```

```
            put_first (v)
        else
                -- Add a cell to the right of the current cell
                -- and swap the two items. Do not forget to update
                -- the cursor positions due the swap of items.
            a_cell := a_cursor.current_cell
            create new_cell.make (a_cell.item)
            a_cell.put (v)
            new_right := a_cell.right
            a_cell.put_right (new_cell)
            if new_right = Void then
                    -- The cursor was on the last cell. The new cell
                    -- becomes the new last cell of the list.
                last_cell := new_cell
            else
                new_cell.put_right (new_right)
            end
            count := count + 1
                -- Update the cursor positions due the swap of items.
            move_all_cursors (a_cell, new_cell)
        end
    end

put_right_cursor, force_right_cursor (v: G; a_cursor: like new_cursor)
        -- Add 'v' to right of 'a_cursor' position.
        -- Do not move cursors.
        -- (Synonym of 'a_cursor.put_right (v)'.)
        -- (Performance: O(1).)
    local
        new_cell, old_cell: like first_cell
    do
        if a_cursor.before then
            put_first (v)
        elseif a_cursor.is_last then
            put_last (v)
        else
            old_cell := a_cursor.current_cell
            create new_cell.make (v)
            new_cell.put_right (old_cell.right)
            old_cell.put_right (new_cell)
            count := count + 1
        end
    end

swap (i, j: INTEGER)
        -- Exchange items at indexes i and j.
        -- Do not move cursors.
        -- (Performance: O(max(i,j)).)
    local
        ii, jj, k: INTEGER
        i_cell, j_cell: like first_cell
        v: G
    do
        if i /= j then
            if i < j then
                ii := i
                jj := j
            else
                ii := j
                jj := i
            end
            i_cell := first_cell
            from k := 1 until k = ii loop
                i_cell := i_cell.right
                k := k + 1
            end
            j_cell := i_cell
            from until k = jj loop
                j_cell := j_cell.right
                k := k + 1
            end
            v := i_cell.item
            i_cell.put (j_cell.item)
            j_cell.put (v)
        end
    end

extend_first, append_first (other: DS_LINEAR [G])
        -- Add items of 'other' to beginning of list.
        -- Keep items of 'other' in the same order.
        -- Do not move cursors.
        -- (Performance: O(other.count).)
    local
        linked_other: like Current
        fcell, lcell: like first_cell
    do
        if not other.is_empty then
            create linked_other.make_from_linear (other)
            fcell := linked_other.first_cell
            lcell := linked_other.last_cell
            if is_empty then
                first_cell := fcell
                last_cell := lcell
            else
                lcell.put_right (first_cell)
                first_cell := fcell
            end
            count := count + other.count
        end
    end

extend_last, append_last (other: DS_LINEAR [G])
        -- Add items of 'other' to end of list.
        -- Keep items of 'other' in the same order.
        -- Do not move cursors.
        -- (Performance: O(other.count).)
    local
        linked_other: like Current
        fcell, lcell: like first_cell
    do
        if not other.is_empty then
            create linked_other.make_from_linear (other)
            fcell := linked_other.first_cell
            lcell := linked_other.last_cell
            if is_empty then
                first_cell := fcell
                last_cell := lcell
            else
                last_cell.put_right (fcell)
                last_cell := lcell
```

```
                end
                count := count + other.count
            end
        end

    extend, append (other: DS_LINEAR [G]; i: INTEGER)
            -- Add items of ‘other’ at ‘i’-th position.
            -- Keep items of ‘other’ in the same order.
            -- Do not move cursors.
            -- (Performance: O(i+other.count).)
        local
            linked_other: like Current
            a_cell: like first_cell
            fcell, lcell: like first_cell
            j: INTEGER
        do
            if i = 1 then
                extend_first (other)
            elseif i = count + 1 then
                extend_last (other)
            elseif not other.is_empty then
                create linked_other.make_from_linear (other)
                fcell := linked_other.first_cell
                lcell := linked_other.last_cell
                    -- Go to cell at index ‘i-1’.
                a_cell := first_cell
                from j := 2 until j = i loop
                    a_cell := a_cell.right
                    j := j + 1
                end
                    -- The cell at index ‘i-1’ exists (because i /= 1)
                    -- and has a right neighbor (because i /= count + 1).
                    -- Insert the new cells in-between.
                lcell.put_right (a_cell.right)
                a_cell.put_right (fcell)
                count := count + other.count
            end
        end

    extend_left_cursor, append_left_cursor (other: DS_LINEAR [G]; a_cursor: like
new_cursor)
            -- Add items of ‘other’ to left of ‘a_cursor’ position.
            -- Keep items of ‘other’ in the same order.
            -- Do not move cursors.
            -- (Synonym of ‘a_cursor.extend_left (other)’.)
            -- (Performance: O(other.count).)
        local
            linked_other: like Current
            a_cell, new_cell, new_right: like first_cell
            fcell, lcell: like first_cell
        do
            if a_cursor.after then
                extend_last (other)
            elseif a_cursor.is_first then
                extend_first (other)
            elseif not other.is_empty then
                    -- Add ‘other.count’ cells to the right of the current
                    -- cell, swap item at current cell with the last cell
                    -- added, put the first item of ‘other’ in the current
                    -- cell and its remaining items in the subsequent cells.
                    -- Do not forget to update the cursor positions due the
                    -- swap of items at current cell.
                a_cell := a_cursor.current_cell
                create new_cell.make (a_cell.item)
                if other.count = 1 then
                    a_cell.put (other.first)
                    new_right := a_cell.right
                    a_cell.put_right (new_cell)
                    if new_right = Void then
                            -- The cursor was on the last cell. The new cell
                            -- becomes the new last cell of the list.
                        last_cell := new_cell
                    else
                        new_cell.put_right (new_right)
                    end
                else
                    create linked_other.make_from_linear (other)
                    fcell := linked_other.first_cell
                    lcell := linked_other.last_cell
                    a_cell.put (fcell.item)
                    new_right := a_cell.right
                    a_cell.put_right (fcell.right)
                    lcell.put_right (new_cell)
                    if new_right = Void then
                            -- The cursor was on the last cell. The new cell
                            -- becomes the new last cell of the list.
                        last_cell := new_cell
                    else
                        new_cell.put_right (new_right)
                    end
                end
                count := count + other.count
                    -- Update the cursor positions due the swap of items.
                move_all_cursors (a_cell, new_cell)
            end
        end

    extend_right_cursor, append_right_cursor (other: DS_LINEAR [G]; a_cursor: like
new_cursor)
            -- Add items of ‘other’ to right of ‘a_cursor’ position.
            -- Keep items of ‘other’ in the same order.
            -- Do not move cursors.
            -- (Synonym of ‘a_cursor.extend_right (other)’.)
            -- (Performance: O(other.count).)
        local
            linked_other: like Current
            old_cell, fcell, lcell: like first_cell
        do
            if not other.is_empty then
                create linked_other.make_from_linear (other)
                fcell := linked_other.first_cell
                lcell := linked_other.last_cell
                if is_empty then
                    first_cell := fcell
                    last_cell := lcell
                elseif a_cursor.before then
                    lcell.put_right (first_cell)
                    first_cell := fcell
                elseif a_cursor.is_last then
                    last_cell.put_right (fcell)
```

```
                    last_cell := lcell
                else
                        -- The current cell exists (because the cursor is
                        -- not 'before' (see above) and not 'after' (see
                        -- precondition)) and it has a right neighbor
                        -- (because the cursor is not 'is_last' (see above)).
                        -- Insert new cells in-between.
                    old_cell := a_cursor.current_cell
                    lcell.put_right (old_cell.right)
                    old_cell.put_right (fcell)
                end
                count := count + other.count
            end
        end

feature -- Removal

    remove_first
            -- Remove item at beginning of list.
            -- Move any cursors at this position 'forth'.
            -- (Performance: O(1).)
        local
            a_cell: like first_cell
        do
            if count = 1 then
                wipe_out
            else
                a_cell := first_cell.right
                move_all_cursors (first_cell, a_cell)
                set_first_cell (a_cell)
                count := count - 1
            end
        end

    remove_last
            -- Remove item at end of list.
            -- Move any cursors at this position 'forth'.
            -- (Performance: O(count).)
        local
            a_cell: like first_cell
            i: INTEGER
        do
            if count = 1 then
                wipe_out
            else
                move_last_cursors_after
                    -- Go to second to last cell.
                a_cell := first_cell
                from i := count until i = 2 loop
                    a_cell := a_cell.right
                    i := i - 1
                end
                set_last_cell (a_cell)
                count := count - 1
            end
        end

    remove (i: INTEGER)
            -- Remove item at 'i'-th position.
            -- Move any cursors at this position 'forth'.
            -- (Performance: O(i).)
        local
            a_cell, old_right, new_right: like first_cell
            j: INTEGER
        do
            if i = 1 then
                remove_first
            elseif i = count then
                remove_last
            else
                    -- Go to the cell at index 'i-1'.
                a_cell := first_cell
                from j := 2 until j = i loop
                    a_cell := a_cell.right
                    j := j + 1
                end
                    -- The cell at index 'i-1' exists (because i /= 1)
                    -- and has a right neighbor (because i /= count + 1,
                    -- see precondition).  Remove the new cell in-between.
                old_right := a_cell.right
                new_right := old_right.right
                move_all_cursors (old_right, new_right)
                a_cell.put_right (new_right)
                count := count - 1
            end
        end

    remove_at_cursor (a_cursor: like new_cursor)
            -- Remove item at 'a_cursor' position.
            -- Move any cursors at this position 'forth'.
            -- (Synonym of 'a_cursor.remove'.)
            -- (Performance: O(1).)
        local
            a_cell, old_cell: like first_cell
        do
            if a_cursor.is_first then
                remove_first
            elseif a_cursor.is_last then
                remove_last
            else
                    -- Remove the cell to the right of the current cell
                    -- and swap the two items. Do not forget to update
                    -- the cursor positions due the swap of items.
                a_cell := a_cursor.current_cell
                old_cell := a_cell.right
                if old_cell = last_cell then
                        -- The cursor was on the second to last cell.
                        -- The cell at cursor position becomes the
                        -- new last cell of the list.
                    set_last_cell (a_cell)
                else
                    a_cell.put_right (old_cell.right)
                end
                a_cell.put (old_cell.item)
                count := count - 1
                    -- Update the cursor positions due the swap of items.
                move_all_cursors (old_cell, a_cell)
            end
        end
```

```
remove_left_cursor (a_cursor: like new_cursor)
        -- Remove item to left of ‘a_cusor’ position.
        -- Move any cursors at this position ‘forth’.
        -- (Synonym of ‘a_cursor.remove_left’.)
        -- (Performance: O(a_cursor.index).)
    local
        new_left, old_left, current_cell: like first_cell
    do
        if a_cursor.after then
            remove_last
        else
            current_cell := a_cursor.current_cell
            new_left := first_cell
            old_left := new_left.right
            if old_left = current_cell then
                    -- The cell to be removed is the first cell
                    -- in the list.
                move_all_cursors (new_left, current_cell)
                set_first_cell (current_cell)
            else
                    -- Go to cell to the left of the cell
                    -- to be removed.
                from
                until
                    old_left.right = current_cell
                loop
                    new_left := old_left
                    old_left := new_left.right
                end
                move_all_cursors (old_left, current_cell)
                new_left.put_right (current_cell)
            end
            count := count - 1
        end
    end

remove_right_cursor (a_cursor: like new_cursor)
        -- Remove item to right of ‘a_cursor’ position.
        -- Move any cursors at this position ‘forth’.
        -- (Synonym of ‘a_cursor.remove_right’.)
        -- (Performance: O(1).)
    local
        current_cell, new_right, old_right: like first_cell
    do
        if a_cursor.before then
            remove_first
        else
            current_cell := a_cursor.current_cell
            old_right := current_cell.right
            new_right := old_right.right
            if new_right = Void then
                    -- The cell to be removed was the last in the
                    -- list. The cell at cursor position becomes
                    -- the new last cell.
                move_last_cursors_after
                set_last_cell (current_cell)
            else
                move_all_cursors (old_right, new_right)
                current_cell.put_right (new_right)
            end
            count := count - 1
        end
    end

prune_first (n: INTEGER)
        -- Remove ‘n’ first items from list.
        -- Move all cursors ‘off’.
        -- (Performance: O(n).)
    local
        i: INTEGER
        new_first: like first_cell
    do
        if count = n then
            wipe_out
        elseif n /= 0 then
            move_all_cursors_after
            new_first := first_cell
            from i := 1 until i > n loop
                new_first := new_first.right
                i := i + 1
            end
            set_first_cell (new_first)
            count := count - n
        else
            move_all_cursors_after
        end
    end

prune_last (n: INTEGER)
        -- Remove ‘n’ last items from list.
        -- Move all cursors ‘off’.
        -- (Performance: O(count-n).)
    do
        keep_first (count - n)
    end

prune (n: INTEGER; i: INTEGER)
        -- Remove ‘n’ items at and after ‘i’-th position.
        -- Move all cursors ‘off’.
        -- (Performance: O(i+n).)
    local
        a_cell, new_right: like first_cell
        j: INTEGER
    do
        if i = 1 then
            prune_first (n)
        elseif n /= 0 then
            move_all_cursors_after
                -- Go to cell at index ‘i-1’.
            a_cell := first_cell
            from j := 2 until j = i loop
                a_cell := a_cell.right
                j := j + 1
            end
                -- Go to the last cell to be removed.
            new_right := a_cell.right
            from j := 1 until j = n loop
                new_right := new_right.right
                j := j + 1
            end
```

```
            new_right := new_right.right
            if new_right = Void then
                    -- The last cell to be removed is actually
                    -- the last cell of the list. The cell at
                    -- index ‘i-1’ becomes the new last cell.
                set_last_cell (a_cell)
            else
                a_cell.put_right (new_right)
            end
            count := count - n
        else
            move_all_cursors_after
        end
    end

prune_left_cursor (n: INTEGER; a_cursor: like new_cursor)
        -- Remove ‘n’ items to left of ‘a_cursor’ position.
        -- Move all cursors ‘off’.
        -- (Synonym of ‘a_cursor.prune_left (n)’.)
        -- (Performance: O(2*a_cursor.index-n).)
    local
        i, j: INTEGER
        a_cell, current_cell: like first_cell
    do
        if a_cursor.after then
            prune_last (n)
        elseif n /= 0 then
                -- Go to the cell to the left of the first
                -- cell to be removed.
            current_cell := a_cursor.current_cell
            j := a_cursor.index - n - 1
            if j = 0 then
                    -- There is no such cell. The cell at cursor
                    -- position becomes the new first cell of the list.
                set_first_cell (current_cell)
            else
                a_cell := first_cell
                from i := 1 until i = j loop
                    a_cell := a_cell.right
                    i := i + 1
                end
                a_cell.put_right (current_cell)
            end
            move_all_cursors_after
            count := count - n
        else
            move_all_cursors_after
        end
    end

prune_right_cursor (n: INTEGER; a_cursor: like new_cursor)
        -- Remove ‘n’ items to right of ‘a_cursor’ position.
        -- Move all cursors ‘off’.
        -- (Synonym of ‘a_cursor.prune_right (n)’.)
        -- (Performance: O(n).)
    local
        i: INTEGER
        new_right, current_cell: like first_cell
    do
        if a_cursor.before then
            prune_first (n)
        elseif n /= 0 then
            current_cell := a_cursor.current_cell
            move_all_cursors_after
                -- Go to the cell to the right of the last
                -- cell to be removed.
            new_right := current_cell.right
            from i := 1 until i > n loop
                new_right := new_right.right
                i := i + 1
            end
            if new_right = Void then
                    -- There is no such cell. The cell at cursor
                    -- position becomes the new last cell of the list.
                set_last_cell (current_cell)
            else
                current_cell.put_right (new_right)
            end
            count := count - n
        else
            move_all_cursors_after
        end
    end

keep_first (n: INTEGER)
        -- Keep ‘n’ first items in list.
        -- Move all cursors ‘off’.
        -- (Performance: O(n).)
    local
        i: INTEGER
        new_last: like first_cell
    do
        if n = 0 then
            wipe_out
        elseif n = count then
            move_all_cursors_after
        else
            move_all_cursors_after
            new_last := first_cell
            from i := 1 until i = n loop
                new_last := new_last.right
                i := i + 1
            end
            set_last_cell (new_last)
            count := n
        end
    end

keep_last (n: INTEGER)
        -- Keep ‘n’ last items in list.
        -- Move all cursors ‘off’.
        -- (Performance: O(count-n).)
    do
        prune_first (count - n)
    end

delete (v: G)
        -- Remove all occurrences of ‘v’.
        -- (Use ‘equality_tester’’s comparison criterion
        -- if not void, use ‘=’ criterion otherwise.)
```

```
    -- Move all cursors ‘off’.
    -- (Performance: O(count).)
local
    n: INTEGER
    a_cell, right_cell: like first_cell
    a_tester: like equality_tester
do
    if is_empty then
        move_all_cursors_after
    else
        a_tester := equality_tester
        if a_tester /= Void then
                -- Remove all occurrences of ‘v’ at the beginning
                -- of the list.
            from
                a_cell := first_cell
            until
                (a_cell = Void) or else
                not a_tester.test (a_cell.item, v)
            loop
                a_cell := a_cell.right
                n := n + 1
            end
            if a_cell = Void then
                    -- All items were occurrences of ‘v’.
                    -- The list becomes empty.
                first_cell := Void
                last_cell := Void
                count := 0
            else
                if n > 0 then
                        -- There was some occurrences of ‘v’ at the
                        -- beginning of the list. Set the new first
                        -- cell of the list accordingly.
                    set_first_cell (a_cell)
                end
                    -- Remove subsequent occurrences of ‘v’.
                right_cell := a_cell.right
                from until (right_cell = Void) loop
                    if a_tester.test (right_cell.item, v) then
                        right_cell := right_cell.right
                        if right_cell /= Void then
                            a_cell.put_right (right_cell)
                        end
                        n := n + 1
                    else
                        a_cell := right_cell
                        right_cell := a_cell.right
                    end
                end
                if a_cell /= last_cell then
                        -- The last items in the list were occurrences
                        -- of ‘v’. Set the new last cell of the list
                        -- accordingly.
                    set_last_cell (a_cell)
                end
                count := count - n
                move_all_cursors_after
            end
        else
                -- Use ‘=’ as comparison criterion.

                -- Remove all occurrences of ‘v’ at the beginning
                -- of the list.
            from
                a_cell := first_cell
            until
                (a_cell = Void) or else (a_cell.item /= v)
            loop
                a_cell := a_cell.right
                n := n + 1
            end
            if a_cell = Void then
                    -- All items were occurrences of ‘v’.
                    -- The list becomes empty.
                first_cell := Void
                last_cell := Void
                count := 0
            else
                if n > 0 then
                        -- There was some occurrences of ‘v’ at the
                        -- beginning of the list. Set the new first
                        -- cell of the list accordingly.
                    set_first_cell (a_cell)
                end
                    -- Remove subsequent occurrences of ‘v’.
                right_cell := a_cell.right
                from until (right_cell = Void) loop
                    if right_cell.item = v then
                        right_cell := right_cell.right
                        if right_cell /= Void then
                            a_cell.put_right (right_cell)
                        end
                        n := n + 1
                    else
                        a_cell := right_cell
                        right_cell := a_cell.right
                    end
                end
                if a_cell /= last_cell then
                        -- The last items in the list were occurrences
                        -- of ‘v’. Set the new last cell of the list
                        -- accordingly.
                    set_last_cell (a_cell)
                end
                count := count - n
                move_all_cursors_after
            end
        end
    end
end

wipe_out
        -- Remove all items from list.
        -- Move all cursors ‘off’.
        -- (Performance: O(1).)
    do
        move_all_cursors_after
        first_cell := Void
        last_cell := Void
```

```
            count := 0
        end

feature {DS_LINKED_LIST, DS_LINKED_LIST_CURSOR} -- Implementation

    first_cell: DS_LINKABLE [G]
            -- First cell in list

    last_cell: like first_cell
            -- Last cell in list

feature {NONE} -- Implementation

    set_first_cell (a_cell: like first_cell)
            -- Set ‘first_cell’ to ‘a_cell’.
            -- This routine has to be called (instead of
            -- making a direct assignment to ‘first_cell’)
            -- when ‘a_cell’ may have dangling left neighbors
            -- that need to be gotten rid of. There is no need
            -- to call this routine when ‘a_cell’ has just
            -- been created.
        require
            a_cell_not_void: a_cell /= Void
        do
            first_cell := a_cell
        ensure
            first_cell_set: first_cell = a_cell
        end

    set_last_cell (a_cell: like first_cell)
            -- Set ‘last_cell’ to ‘a_cell’.
            -- This routine has to be called (instead of
            -- making a direct assignment to ‘last_cell’)
            -- when ‘a_cell’ may have dangling right neighbors
            -- that need to be gotten rid of. There is no need
            -- to call this routine when ‘a_cell’ has just
            -- been created.
        require
            a_cell_not_void: a_cell /= Void
        do
            a_cell.forget_right
            last_cell := a_cell
        ensure
            last_cell_set: last_cell = a_cell
        end

feature {NONE} -- Cursor movement

    move_last_cursors_after
            -- Move ‘after’ all cursors at last position.
        local
            a_cursor, previous_cursor, next_cursor: like new_cursor
            old_cell: like first_cell
        do
            a_cursor := internal_cursor
            old_cell := last_cell
            if a_cursor.current_cell = old_cell then
                a_cursor.set_after
            end
            previous_cursor := a_cursor
            a_cursor := a_cursor.next_cursor
            from until (a_cursor = Void) loop
                if a_cursor.current_cell = old_cell then
                    a_cursor.set_after
                    next_cursor := a_cursor.next_cursor
                    previous_cursor.set_next_cursor (next_cursor)
                    a_cursor.set_next_cursor (Void)
                    a_cursor := next_cursor
                else
                    previous_cursor := a_cursor
                    a_cursor := a_cursor.next_cursor
                end
            end
        end

    move_all_cursors (old_cell, new_cell: like first_cell)
            -- Move all cursors at position ‘old_cell’ to ‘new_cell’.
        require
            old_cell_not_void: old_cell /= Void
            new_cell_not_void: new_cell /= Void
        local
            a_cursor: like new_cursor
        do
            from a_cursor := internal_cursor until (a_cursor = Void) loop
                if a_cursor.current_cell = old_cell then
                    a_cursor.set_current_cell (new_cell)
                end
                a_cursor := a_cursor.next_cursor
            end
        end

    move_all_cursors_after
            -- Move ‘after’ all cursors.
        local
            a_cursor, next_cursor: like new_cursor
        do
            from a_cursor := internal_cursor until (a_cursor = Void) loop
                a_cursor.set_after
                next_cursor := a_cursor.next_cursor
                a_cursor.set_next_cursor (Void)
                a_cursor := next_cursor
            end
        end

invariant

    first_cell: is_empty = (first_cell = Void)
    last_cell: is_empty = (last_cell = Void)
    last_constraint: last_cell /= Void implies last_cell.right = Void

end
```