Aufgabe 1 – Verkettete Liste

**Lösungsidee:**

Die Liste verwendet eine einfach verkettete Liste, in der die Elemente in der Reihenfolge ihres Einfügens hinzugefügt werden.

Die sortierte Liste verwendet eine einfach verkettete Liste, in der die Elemente in aufsteigend sortierter Reihenfolge eingefügt werden. Beim Hinzufügen eines neuen Elements wird es an der richtigen Position eingefügt, um die Sortierung beizubehalten.

**Laufzeitkomplexität:**

In der List wird eine einfache lineare Suche durchgeführt, indem jedes Element der Liste nacheinander überprüft wird. Die Laufzeitkomplexität ist daher linear (O(n)), da alle Elemente durchlaufen werden müssen, um das gesuchte Element zu finden oder festzustellen, dass es nicht vorhanden ist.

In der SortedList wird hingegen eine optimierte Suche verwendet, um das Element effizient zu finden. Der Code nutzt die Sortierung der Liste aus, um die Suche vorzeitig abzubrechen, wenn der aktuelle Wert größer als value ist. Dadurch wird die Anzahl der durchlaufenden Elemente reduziert und die Laufzeitkomplexität auf O(log n) reduziert.

**Zeitaufwand: ~**1h

**Code:**

unit ListUnit;

interface

type

  ListPtr = ^ListNode;

  ListNode = object

    value: Integer;

    next: ListPtr;

  end;

  List = ^ListObj;

  ListObj = object

  public

    constructor Init;

    destructor Done; virtual;

    procedure Add(val: Integer);

    function Contains(val: Integer): Boolean;

    function Size: Integer;

    procedure Remove(val: Integer);

    procedure Clear;

  protected

    head: ListPtr;

    function NewNode(val: integer): ListPtr;

  end;

function NewList: List;

implementation

function NewList: List;

var

  l: List;

begin

  New(l, Init);

  NewList := l;

end;

constructor ListObj.Init;

begin

  head := nil;

end;

destructor ListObj.Done;

begin

  Clear;

end;

procedure ListObj.Add(val: Integer);

var

  node, lastNode: ListPtr;

begin

  node := NewNode(val);

  if head = nil then

    head := node

  else

  begin

    lastNode := head;

    while lastNode^.next <> nil do

      lastNode := lastNode^.next;

    lastNode^.next := node;

  end;

end;

function ListObj.Contains(val: Integer): Boolean;

var

  currentNode: ListPtr;

begin

  currentNode := head;

  while currentNode <> nil do

  begin

    if currentNode^.value = val then

    begin

      Contains := true;

      Exit;

    end;

    currentNode := currentNode^.next;

  end;

  Contains := false;

end;

function ListObj.Size: Integer;

var

  currentNode: ListPtr;

  i: integer;

begin

  currentNode := head;

  i := 0;

  while currentNode <> nil do

  begin

    Inc(i);

    currentNode := currentNode^.next;

  end;

  Size := i;

end;

procedure ListObj.Remove(val: Integer);

var

  currentNode, prevNode, tempNode: ListPtr;

begin

  prevNode := nil;

  currentNode := head;

  while currentNode <> nil do

    if currentNode^.value = val then

    begin

      if prevNode = nil then

      begin

        // The node to remove is the head node

        tempNode := head;

        head := currentNode^.next;

        Dispose(tempNode);

        currentNode := head;

      end

      else

      begin

        // The node to remove is not the head node

        tempNode := currentNode;

        prevNode^.next := currentNode^.next;

        currentNode := currentNode^.next;

        Dispose(tempNode);

      end;

    end

    else

    begin

      prevNode := currentNode;

      currentNode := currentNode^.next;

    end;

end;

procedure ListObj.Clear;

var

  currentNode, tempNode: ListPtr;

begin

  currentNode := head;

  while currentNode <> nil do

  begin

    tempNode := currentNode;

    currentNode := currentNode^.next;

    Dispose(tempNode);

  end;

  head := nil;

end;

function ListObj.NewNode(val: integer): ListPtr;

var

  node: ListPtr;

begin

  New(node);

  node^.value := val;

  node^.next := nil;

  NewNode := node;

end;

end.

unit SortedListUnit;

interface

uses

  ListUnit;

type

  SortedList = ^SortedListObj;

  SortedListObj = object(ListObj)

    constructor Init;

    destructor Done; virtual;

    procedure Add(val: Integer); virtual;

    function Contains(val: Integer): Boolean; virtual;

  end;

function NewSortedList: SortedList;

implementation

function NewSortedList: SortedList;

var

  sl: SortedList;

begin

  New(sl, Init);

  NewSortedList := sl;

end;

constructor SortedListObj.Init;

begin

  inherited Init;

end;

destructor SortedListObj.Done;

begin

  inherited Done;

end;

procedure SortedListObj.Add(val: Integer);

var

  node, currentNode, prevNode: ListPtr;

begin

  node := NewNode(val);

  if head = nil then

    head := node

  else if val < head^.value then

  begin

    node^.next := head;

    head := node;

  end

  else

  begin

    prevNode := head;

    currentNode := head^.next;

    while (currentNode <> nil) and (currentNode^.value < val) do

    begin

      prevNode := currentNode;

      currentNode := currentNode^.next;

    end;

    node^.next := currentNode;

    prevNode^.next := node;

  end;

end;

function SortedListObj.Contains(val: Integer): Boolean;

var

  currentNode: ListPtr;

begin

  currentNode := head;

  while (currentNode <> nil) and (currentNode^.value < val) do

    currentNode := currentNode^.next;

  Contains := (currentNode <> nil) and (currentNode^.value = val);

end;

end.

**Test:**

program TestList;

uses

  ListUnit, SortedListUnit;

procedure ExecuteListTests(l: List);

begin

  // Add some values to the list

  l^.Add(5);

  l^.Add(10);

  l^.Add(15);

  // Print the size of the list

  Writeln('Size of the list: ', l^.Size);

  // Check if the list contains a value

  Writeln('List contains 10: ', l^.Contains(10));

  Writeln('List contains 20: ', l^.Contains(20));

  // Remove a value from the list

  l^.Remove(10);

  // Print the updated size of the list

  Writeln('Size of the list after removal: ', l^.Size);

  // Clear the list

  l^.Clear;

  Writeln('Size of the list after clearing: ', l^.Size);

end;

var

  l: List;

  sl: SortedList;

begin

  l := NewList;

  sl := NewSortedList;

  Writeln('Testing List:');

  ExecuteListTests(l);

  Dispose(l, Done);

  writeln;writeln;

  Writeln('Testing SortedList:');

  ExecuteListTests(sl);

  Dispose(sl, Done);

  writeln;writeln;

end.

A screenshot of a computer program

Description automatically generated with medium confidence