Aufgabe 1 – Ein neuer Behälter für die MiniLib

**Lösungsidee:**

Neues Objekt MLList anlegen und Methoden von MLColl ableiten, in Obj die KopfNode speichern und alle Standardmethoden implementieren.

Für Prepend die neue Node am Anfang der Liste einfügen, bei Add am Ende der Liste.

Für den Iterator die KopfNode beim Init speichern und bei jedem Methodenaufruf von Next den aktuell gespeicherten Wert zurückgeben und Next auf die nächste Node setzten.

**Zeitaufwand: ~**1h

**Code:**

unit MLLi;

interface

uses

  MLObj, MLColl;

type

  MLListNodePtr = ^MLListNode;

  MLListNode = record

    obj: MLObject;

    next: MLListNodePtr;

  end;

  (\* === class MLList === \*)

  MLList = ^MLListObj;

  MLListObj = object(MLCollectionObj)

    head: MLListNodePtr;

    constructor Init;

    destructor Done; virtual;

    function Size: INTEGER; virtual;

    procedure Add(o: MLObject); virtual;

    function Remove(o: MLObject): MLObject; virtual;

    function Contains(o: MLObject): BOOLEAN; virtual;

    procedure Clear; virtual;

    function NewIterator: MLIterator; virtual;

    procedure Prepend(o: MLObject);

  end;

  (\* === class MLListIterator === \*)

  MLListIterator = ^MLListIteratorObj;

  MLListIteratorObj = object(MLIteratorObj)

    curNode: MLListNodePtr;

    constructor Init(l: MLList);

    destructor Done; virtual;

    function Next: MLObject; virtual;

  end;

function NewMLList: MLList;

implementation

function NewMLList: MLList;

var

  l: MLList;

begin

  New(l, Init);

  NewMLList := l;

end;

(\* === class MLList === \*)

constructor MLListObj.Init;

begin

  inherited Init;

  Register('MLList', 'MLCollection');

  head := NIL;

end;

destructor MLListObj.Done;

begin

  Clear;

  inherited Done;

end;

function MLListObj.Size: INTEGER;

var

  count: INTEGER;

  curNode: MLListNodePtr;

begin

  count := 0;

  curNode := head;

  while curNode <> NIL do

  begin

    curNode := curNode^.next;

    Inc(count);

  end;

  Size := count;

end;

procedure MLListObj.Add(o: MLObject);

var

  newNode: MLListNodePtr;

  curNode: MLListNodePtr;

begin

  New(newNode);

  newNode^.obj := o;

  newNode^.next := NIL;

  if head = NIL then

    head := newNode

  else begin

    curNode := head;

    while curNode^.next <> NIL do

      curNode := curNode^.next;

    curNode^.next := newNode;

  end;

end;

function MLListObj.Remove(o: MLObject): MLObject;

var

  prevNode, curNode: MLListNodePtr;

begin

  if head = NIL then

  begin

    Remove := NIL;

    Exit;

  end;

  prevNode := NIL;

  curNode := head;

  while (curNode <> NIL) and (curNode^.obj <> o) do

  begin

    prevNode := curNode;

    curNode := curNode^.next;

  end;

  if curNode = NIL then

  begin

    Remove := NIL;

    Exit;

  end;

  if prevNode = NIL then

    head := curNode^.next

  else

    prevNode^.next := curNode^.next;

  Remove := curNode^.obj;

  Dispose(curNode);

end;

function MLListObj.Contains(o: MLObject): BOOLEAN;

var

  curNode: MLListNodePtr;

begin

  curNode := head;

  while curNode <> NIL do

  begin

    if curNode^.obj^.IsEqualTo(o) then

    begin

      Contains := TRUE;

      Exit;

    end;

    curNode := curNode^.next;

  end;

  Contains := FALSE;

end;

procedure MLListObj.Clear;

var

  curNode, nextNode: MLListNodePtr;

begin

  curNode := head;

  while curNode <> NIL do

  begin

    nextNode := curNode^.next;

    Dispose(curNode^.obj, Done);

    Dispose(curNode);

    curNode := nextNode;

  end;

  head := NIL;

end;

function MLListObj.NewIterator: MLIterator;

var

  iterator: MLListIterator;

begin

  New(iterator, Init(@Self));

  NewIterator := iterator;

end;

procedure MLListObj.Prepend(o: MLObject);

var

  newNode: MLListNodePtr;

begin

  New(newNode);

  newNode^.obj := o;

  newNode^.next := head;

  head := newNode;

end;

(\* === class MLListIterator === \*)

constructor MLListIteratorObj.Init(l: MLList);

begin

  inherited Init;

  curNode := l^.head;

end;

destructor MLListIteratorObj.Done;

begin

  inherited Done;

end;

function MLListIteratorObj.Next: MLObject;

begin

  if curNode <> NIL then

  begin

    Next := curNode^.obj;

    curNode := curNode^.next;

  end

  else

    Next := NIL;

end;

end.

**Tests:**

program MLListTest;

uses MLLi, MLObj, MLInt, MLColl, MetaInfo;

procedure RunMLListTests;

var

  list: MLList;

  int2: MLInteger;

  iterator: MLIterator;

  next: MLObject;

begin

  list := NewMLList;

  int2 := NewMlInt(2);

  list^.Add(NewMLInt(1));

  list^.Add(int2);

  list^.Add(NewMLInt(3));

  writeln('Size: ', list^.Size); // Output: 3

  writeln('Removed int2: ', list^.Remove(int2)^.AsString);

  writeln('Contains int2: ', list^.Contains(int2)); // Output: False

  Dispose(int2, Done);

  iterator := list^.NewIterator;

  next := iterator^.Next;

  while next <> NIL do

  begin

    writeln('Iterator value: ', next^.asString); // Output: 1, 3

    next := iterator^.Next;

  end;

  writeln;

  Dispose(iterator, Done);

  list^.Prepend(NewMLInt(4));

  iterator := list^.NewIterator;

  next := iterator^.Next;

  while next <> NIL do

  begin

    writeln('Iterator value: ', next^.asString); // Output: 4, 1, 3

    next := iterator^.Next;

  end;

  writeln;

  Dispose(iterator, Done);

  list^.Clear;

  writeln('Size: ', list^.Size); // Output: 0

  Dispose(list, Done);

end;

begin

  RunMLListTests;

  WriteMetaInfo;

end.

