

Aufgabe 1 - „Behälter“ Vector als ADT

Lösungsidee:

Wie bei dem Beispiel wo die Größe des Feldes erst zur Laufzeit fixiert wird ein Feld auf die gleiche Weise erstellen, allerdings immer wenn die Größe überschritten werden sollte, wird die Funktion GrowVector aufgerufen, die die Kapazität des Feldes verdoppelt und somit auch den allokierten Speicher und die Werte des alten Vektors in den neuen kopiert.

Zeitaufwand: ~1h 45min

Code:

```
unit VectorUnit;

interface

type
    Vector = Pointer;

procedure InitVector(var v: Vector);
procedure DisposeVector(var v: Vector);
procedure Add(var v: Vector; val: Integer);
procedure SetElementAt(var v: Vector; pos: Integer; val: Integer);
function ElementAt(v: Vector; pos: Integer): Integer;
procedure RemoveElementAt(var v: Vector; pos: Integer);
function Size(v: Vector): Integer;
function Capacity(v: Vector): Integer;

implementation

// using MaxInt for the intArray here so i dont have to disable rangecheck
// error everytime i try to access the array if i used array[0..0] instead.
// There is no allocation of more memory as a consequence because i manually
// set the memory with capacity * SizeOf(Integer) so there shouldnt be a
// problem, also i choose MaxInt because my count and capacity are int anyway
type
    IntArray = array[0..MaxInt] of Integer;
    PIntArray = ^IntArray;
    VecRec = record
        data: PIntArray;
        count: Integer;
        capacity: Integer;
    end;
    PVector = ^VecRec;

procedure InitVector(var v: Vector);
```

```

var
  pv: PVector;
begin
  new(pv);
  pv^.count := 0;
  pv^.capacity := 1;
  GetMem(pv^.data, pv^.capacity * SizeOf(Integer));
  if pv^.data = nil then
  begin
    WriteLn('Error: Heap overflow.');
```

Halt(1);

```
  end;
  v := pv;
end;

procedure DisposeVector(var v: Vector);
var
  pv: PVector;
begin
  pv := PVector(v);
  FreeMem(pv^.data, pv^.capacity * SizeOf(Integer));
  Dispose(v);
end;

procedure GrowVector(var v: VecRec);
var
  newCapacity: longInt;
  newData: PIntArray;
  i: Integer;
begin
  newCapacity := v.capacity * 2;
  if newCapacity >= MaxInt then
  begin
    WriteLn('Error: Vector overflow.');
```

Halt(1);

```
  end;
  GetMem(newData, newCapacity * SizeOf(Integer));
  for i := 0 to v.count - 1 do
    newData^[i] := v.data^[i];
  FreeMem(v.data, v.capacity * SizeOf(Integer));
  v.data := newData;
  v.capacity := newCapacity;
end;

procedure Add(var v: Vector; val: Integer);
var
  pv: PVector;
begin
```

```

    pv := PVector(v);
    if pv^.count = pv^.capacity then
        GrowVector(pv^);
    pv^.data^[pv^.count] := val;
    Inc(pv^.count);
end;

procedure SetElementAt(var v: Vector; pos: Integer; val: Integer);
var
    pv: PVector;
begin
    pv := PVector(v);
    if (pos < 0) or (pos >= pv^.count) then
        begin
            WriteLn('Error: Index out of range.');
            Halt(1);
        end;
    pv^.data^[pos] := val;
end;

function ElementAt(v: Vector; pos: Integer): Integer;
var
    pv: PVector;
begin
    pv := PVector(v);
    if (pos < 0) or (pos >= pv^.count) then
        begin
            WriteLn('Error: Index out of range.');
            Halt(1);
        end;
    ElementAt := pv^.data^[pos];
end;

procedure RemoveElementAt(var v: Vector; pos: Integer);
var
    i: Integer;
    pv: PVector;
begin
    pv := PVector(v);
    if (pos < 0) or (pos >= pv^.count) then
        begin
            WriteLn('Error: Index out of range.');
            Halt(1);
        end;
    for i := pos to pv^.count - 2 do
        pv^.data^[i] := pv^.data^[i + 1];
    Dec(pv^.count);
end;

```

```

end;

function Size(v: Vector): Integer;
var
    pv: PVector;
begin
    pv := PVector(v);
    Size := pv^.count;
end;

function Capacity(v: Vector): Integer;
var
    pv: PVector;
begin
    pv := PVector(v);
    Capacity := pv^.capacity;
end;

end.

```

Eine mögliche Verbesserung meines Codes wäre den Vektor wieder zu verkleinern, wenn der Count kleiner als die Hälfte von der Kapazität ist, oder den Vektor nicht jedes Mal ums doppelte zu vergrößern sondern eine bessere gewählten wert zu verwenden. Aber das ist immer usecase spezifisch, je nachdem, was man mit dem Vektor anfangen will.

Test Code:

```

program VectorTests;

uses VectorUnit;

var
    v: Vector;
begin
    // Initialize an empty vector
    InitVector(v);

    // Test adding elements
    WriteLn('Test adding elements:');
    writeln('0 elements - size: ', Size(v), ', capacity: ', Capacity(v));
    Add(v, 1);
    writeln('1 element - size: ', Size(v), ', capacity: ', Capacity(v));
    Add(v, 2);
    writeln('2 elements - size: ', Size(v), ', capacity: ', Capacity(v));
    Add(v, 3);

```

```

writeln('3 elements - size: ', Size(v), ', capacity: ', Capacity(v));
Add(v, 4);
writeln('4 elements - size: ', Size(v), ', capacity: ', Capacity(v));
Add(v, 5);
writeln('5 elements - size: ', Size(v), ', capacity: ', Capacity(v));
WriteLn;
WriteLn;

// Test getting and setting elements
WriteLn('Test getting and setting elements:');
WriteLn('Element at position 2: ', ElementAt(v, 2)); // should print 3
SetElementAt(v, 2, 6);
WriteLn('Element at position 2 after setting it to 6: ', ElementAt(v, 2));
// should print 6
WriteLn;
WriteLn;

// Test removing elements
WriteLn('Test removing elements:');
WriteLn('Vector size before removing an element: ', Size(v)); // should
print 5
RemoveElementAt(v, 2);
WriteLn('Vector size after removing an element: ', Size(v)); // should print
4
WriteLn('Element at position 2 after removing element at position 2: ',
ElementAt(v, 2)); // should print 4
WriteLn;
WriteLn;

// Test disposing of vector
DisposeVector(v);
end.

```

Test Ausgabe:

› Test adding elements:

0 elements - size: 0, capacity: 1
1 element - size: 1, capacity: 1
2 elements - size: 2, capacity: 2
3 elements - size: 3, capacity: 4
4 elements - size: 4, capacity: 4
5 elements - size: 5, capacity: 8

Test getting and setting elements:

Element at position 2: 3

Element at position 2 after setting it to 6: 6

Test removing elements:

Vector size before removing an element: 5

Vector size after removing an element: 4

Element at position 2 after removing element at position 2: 4

Heap dump by heaptrc unit of C:\Repos\2023SS_ADF\UE5\hu\vector-tests.exe

103 memory blocks allocated : 2332/2600

103 memory blocks freed : 2332/2600

0 unfreed memory blocks : 0

True heap size : 163840 (96 used in System startup)

True free heap : 163744