Vor Abgabe <http://rise4fun.com/Dafny/UeOz>

Nach Abgabe

* (Saeed) <http://rise4fun.com/Dafny/hHe>
* (Dimitri)

//autor: Saeed

// Übungsaufgabe 3.1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

method Max(arr: array<int>) returns (max: int)

requires arr != null && arr.Length > 0;

ensures forall x: int :: 0 <= x < arr.Length ==> max >= arr[x];

ensures exists x: int :: 0 <= x < arr.Length && max == arr[x];

{

var i : int := 1;

max := arr[0];

while(i < arr.Length)

invariant 0 < i <= arr.Length;

invariant forall y: int :: 0 <= y < i ==> max >= arr[y];

invariant exists x: int :: 0 <= x < i && max == arr[x];

decreases arr.Length - i;

{

if (max < arr[i])

{

max := arr[i];

}

i := i+1;

}

}

// Übungsaufgabe 3.2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

method Search(a: array<int>, x: int) returns (i: int)

requires a != null;

ensures 0 <= i ==> i < a.Length && a[i] == x;

ensures i == -1 ==> forall y :: 0 <= y < a.Length ==> a[y] != x;

ensures -1 <= i;

{

i := 0;

while (i < a.Length)

invariant 0 <= i <= a.Length;

invariant forall y :: 0 <= y < i ==> a[y] != x;

{

if (a[i] == x) { return i; }

i := i + 1;

}

i := -1;

}

// Übungsaufgabe 3.3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

predicate sorted(a: array<int>, m:int, n:int) // prädikat - zum überprüfung

requires a != null;

requires a.Length >= n >= 0;

requires a.Length >= m >= 0;

reads a; //ohne das kann er "a" nicht lesen

{

forall x, y :: m <= x < y < n ==> a[x] <= a[y]

}

method MaxSort(b: array<int>)

modifies b;

requires b != null;

ensures sorted(b, 0, b.Length);

ensures multiset(b[..]) == multiset(old(b[..]));

{

var i := b.Length;

var m := 0;

while (i > 1)

invariant 0 <= m <= i <= b.Length;

invariant sorted(b, i, b.Length);

invariant forall x, y :: 0 <= x < i <= y < b.Length ==> b[x] <= b[y];

invariant multiset(b[..]) == multiset(old(b[..]));

decreases i;

{

//für Test

print multiset(b[..]);

print "\n";

//für Test

var m := MaxIndex(b, i - 1); //der maximale erreichbare index wird gesucht

b[m], b[i - 1] := b[i - 1], b[m];

i := i - 1;

}

}

method MaxIndex(arr: array<int>, j: int) returns (imax: int)

requires arr != null;

requires 0 <= j < arr.Length;

ensures 0 <= imax <= j;

ensures forall x :: 0 <= x <= j ==> arr[x] <= arr[imax];

{

imax := 0;

var i := 0;

while (i <= j)

invariant 0 <= j < arr.Length;

invariant 0 <= i <= j + 1;

invariant 0 <= imax <= j;

invariant forall x :: 0 <= x && x < i ==> arr[imax] >= arr[x];

decreases j - i;

{

if (arr[i] > arr[imax]) { imax := i; }

i := i + 1;

}

}

// Übungsaufgabe 3.4\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

method InsertionSort(b: array<int>)

modifies b; // veränderung der variable "a" ist erlaubt

requires b != null;

ensures sorted(b, 0, b.Length);

ensures multiset(old(b[..])) == multiset(b[..])

{

if (b.Length < 2) { return; }

var i := 1;

var j := 0;

while i < b.Length

invariant 0 < i <= b.Length;

invariant sorted(b, 0, i);

invariant multiset(b[..]) == multiset(old(b[..]));

decreases b.Length - i;

{

//für Test

print multiset(b[..]);

print "\n";

//für Test

j := i;

while j > 0 && b[j] <= b[j - 1]

invariant forall x, y :: 0 <= x < j < y <= i ==> b[x] <= b[y];

invariant sorted(b, 0, j) && sorted(b, j, i + 1);

invariant multiset(b[..]) == multiset(old(b[..]));

decreases j;

{

b[j], b[j - 1] := b[j - 1], b[j];

j := j - 1;

}

i := i + 1;

}

}

//Ausgaben\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

method Main()

{

print "\*\*\*\*\*\*\*\*\*\*MaxTest\*\*\*\*\*\*\*\*\*\*\*\n";

var a := new int[10];

var i := 0;

while i < a.Length

{

a[i] := i + 1;

i := i + 1;

}

var result := Max(a);

print "maximalen Element in Array: ";

print result;

print "\n\n\n";

print "\*\*\*\*\*\*\*\*\*SearchTest\*\*\*\*\*\*\*\*\*\*\*\n";

var x := 5;

i := 0;

a := new int[10];

while i < a.Length

{

a[i] := i + 1;

i := i + 1;

}

result := Search(a, x);

print "Index der gesuchte Element ist: ";

print result;

print "\n\n\n";

print "\*\*\*\*\*\*\*\*\*MaxSortTest\*\*\*\*\*\*\*\*\*\*\n";

a := new int[10];

i := 10;

while i > 0

{

a[a.Length - i] := i;

i := i - 1;

}

print "Array VOR MaxSort:\n[";

i := 0;

while i < a.Length

{

print a[i];

if i + 1 != a.Length

{

print ", ";

}

i := i + 1;

}

print "]";

print "\n\n";

MaxSort(a);

if a != null

{

print "\nArray NACH MaxSort:\n[";

i := 0;

while i < a.Length

{

print a[i];

if i + 1 != a.Length

{

print ", ";

}

i := i + 1;

}

print "]";

print "\n\n\n";

}

print "\*\*\*\*\*\*\*\*\*\*InsertionSortTest\*\*\*\*\*\*\*\*\*\*\*\*\n";

a := new int[10];

i := 10;

while i > 0

{

a[a.Length - i] := i;

i := i - 1;

}

print "Array VOR InsertionSort:\n[";

i := 0;

while i < a.Length

{

print a[i];

if i + 1 != a.Length

{

print ", ";

}

i := i + 1;

}

print "]";

print "\n\n";

InsertionSort(a);

if a != null

{

print "\nArray NACH InsertionSort:\n[";

i := 0;

while i < a.Length

{

print a[i];

if i + 1 != a.Length

{

print ", ";

}

i := i + 1;

}

print "]";

print "\n\n\n";

}

}