<http://rise4fun.com/Dafny/a5sH>

//autor: Saeed

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

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

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

ensures forall k: int :: 0 <= k < arr.Length ==> result >= arr[k];

{

var i : int := 1;

var max : int := arr[0];

while i < arr.Length

invariant i >= 0;

invariant i <= arr.Length;

invariant max >= arr[i-1];

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

decreases (arr.Length-i);

{

if (max < arr[i])

{

max := arr[i];

}

i := i+1;

}

result := max;

}

// 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 < 0 ==> forall k :: 0 <= k < a.Length ==> a[k] != x;

{

i := 0;

while (i < a.Length)

invariant 0 <= i <= a.Length;

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

{

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

i := i + 1;

}

i := -1;

}

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

predicate sorted(a: array<int>, l:int, h:int)

requires a != null;

requires a.Length >= h >= 0;

requires a.Length >= l >= 0;

reads a;

{

forall j, k :: l <= j < k < h ==> a[j] <= a[k]

}

method MaxSort(a: array<int>) returns (b: array<int>)

requires a != null;

modifies a;

ensures b != null;

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

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

{

b := a;

var i := b.Length;

var m := 0;

while (i > 0)

decreases i;

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

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

invariant forall k, l :: 0 <= k <= i - 1 < l < b.Length ==> b[k] <= b[l];

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

{

//für Test

print multiset(b[..]);

print "\n";

var m := MaxIdxTo(b, i - 1);

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

i := i - 1;

}

}

method MaxIdxTo(a: array<int>, j: int) returns (imax: int)

requires a != null;

requires 0 <= j < a.Length;

ensures 0 <= imax <= j;

ensures forall k :: 0 <= k <= j ==> a[k] <= a[imax];

{

imax := 0;

var i := 0;

while (i <= j)

decreases j - i;

invariant 0 <= j < a.Length;

invariant 0 <= i <= j + 1;

invariant 0 <= imax <= j;

invariant forall k :: 0 <= k && k < i ==> a[imax] >= a[k];

{

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

i := i + 1;

}

}

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

method InsertionSort(a: array<int>) returns (b: array<int>)

requires a != null;

modifies a;

ensures b != null;

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

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

{

b := a;

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

var i, j := 1, 0;

while i < b.Length

invariant 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";

j := i;

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

invariant forall k, l :: 0 <= k < j < l <= i ==> b[k] <= b[l];

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";

var result\_2 := MaxSort(a);

if result\_2 != null

{

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

i := 0;

while i < result\_2.Length

{

print result\_2[i];

if i + 1 != result\_2.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";

var result\_3 := InsertionSort(a);

if result\_3 != null

{

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

i := 0;

while i < result\_3.Length

{

print result\_3[i];

if i + 1 != result\_3.Length

{

print ", ";

}

i := i + 1;

}

print "]";

print "\n\n\n";

}

}