// übungsaufgabe 2.1

function Fibonacci(n: int): int

requires n >= 0;

ensures Fibonacci(n) >= 0;

decreases n; //n muss vermindert werden

{

if n == 0 then 1

else if n == 1 then 1

else Fibonacci(n - 2) + Fibonacci(n - 1)

}

method Compute\_Fib(n:int) returns (x:int)

requires n >= 0; //vorbedinung muss eingehalten werden

ensures x == Fibonacci(n); //nachbedienung

{

x := Compute\_Fib\_Rec(n, 0, 0, 1);

}

method Compute\_Fib\_Rec(n:int, i:int, var\_a:int, var\_b:int) returns (x:int)

requires 0 <= i <= n;

requires i==0 ==> (var\_a == 0 && var\_b == 1);

requires i==1 ==> (var\_a == 1 && var\_b == 0);

requires i>1 ==> (var\_a == Fibonacci(i-1) && var\_b == Fibonacci(i-2));

ensures x == Fibonacci(n);

decreases n-i; //increases i; --> i musst erhöht werden.

{

if n==i { //1. durchlauf i = 0, 2.durchlauf i = 1 --(durch rekursive aufruf z.31)

x := var\_a+var\_b;

} else {

x := Compute\_Fib\_Rec(n, i+1, var\_a+var\_b, var\_a); //1.durchlauf (n, i=1, var\_a=1, var\_b=0)

}

}

method Main()

{

var result:int := 0;

var i:int := 0;

while(i<10) {

result := Compute\_Fib(i);

print "Compute\_Fib(";

print i;

print "):= ";

print result;

print "\n";

i := i + 1;

}

}

// übungsaufgabe 2.2

function pow(base:int, exp:int): int

requires exp >= 0;

decreases exp;

{

if exp==0 then 1

else pow(base, exp-1) \* base

}

method f(n:int) returns (x:int)

requires n>=0;

ensures x==pow(3,n+1)-1; //geschlossene form

decreases n;

{

if n==0 {

x := 2; // { 2 n == 0

} else { //<|

var temp := f(n-1); // |

x := 3\*temp+2; // { 3 · f(n − 1) + 2 n > 0

}

}

method Main()

{

var result:int := 0;

var i:int := 0;

while(i<10) {

result := f(i);

print "f(";

print i;

print "):= ";

print result;

print "\n";

i := i + 1;

}

}

// übungsaufgabe 2.3

function pow(base:int, exp:int): int

requires exp >= 0;

decreases exp;

{

if exp==0 then 1

else pow(base, exp-1) \* base

}

function algo1(A:seq<int>, i:int):int

requires 0 <= i <= |A|;

{

if(i==0) then 0

else algo1(A, i-1) + A[i-1]

}

method algorithm\_1(A:seq<int>, n:int) returns (x:int)

requires 0 <= n <= |A|;

ensures x == algo1(A, n);

{

var counter:int := 0;

x := 0;

var i := 0;

while(i<n)

invariant 0 <= i <= n;

invariant x == algo1(A, i);

invariant counter == i;

{

x := x + A[i];

i := i + 1;

counter := counter + 1;

}

assert counter==n;

}

method algorithm\_2(x:int, k:int) returns (r:int)

requires k>=0;

ensures r==pow(x,k);

{

var counter:int := 0;

r := 1;

var i := 0;

while (i<k)

invariant 0 <= i <= k;

invariant r==pow(x,i);

invariant counter == i;

{

r := r \* x;

i := i + 1;

counter := counter + 1;

}

assert counter==k;

}

method algorithm\_3(n:int) returns (C:array<int>)

requires n >= 0;

ensures C != null;

ensures C.Length == n;

ensures forall x :: 0 <= x < C.Length ==> C[x] == n-1;

{

var A:array<int> := new int[n];

var i := 0;

while (i<n)

invariant 0 <= i <= n == A.Length;

invariant forall x :: 0 <= x < i <= n ==> A[x] == x;

{

A[i] := i;

i := i + 1;

}

C := new int[n];

i := 0;

while (i<n)

invariant 0 <= i <= n;

invariant n > 0 ==> A[n-1] == n-1;

invariant forall x :: 0 <= x < n-1 ==> A[x] < A[n-1];

invariant forall x :: 0 <= x < i ==> C[x] == n-1;

{

C[i] := 0;

var j := n-1;

while (j>=0)

invariant -1 <= j <= n-1;

invariant n > 0 ==> A[n-1] == n-1;

invariant forall x :: 0 <= x < n-1 ==> A[x] < A[n-1];

invariant forall x :: 0 <= x < i ==> C[x] == n-1;

invariant j==n-1 ==> C[i] == 0;

invariant j<n-1 ==> C[i] == A[n-1];

{

if(A[j] > C[i]) {

C[i] := A[j];

}

j := j - 1;

}

i := i + 1;

}

}

method Main()

{

var result:int := 0;

var i:int := 0;

while(i<10) {

var a:seq<int> := [7, 5, 1, 0, 9, 3, 8, 2, 4, 6];

result := algorithm\_1(a, i);

print "algorithm\_1([7, 5, 1, 0, 9, 3, 8, 2, 4, 6], ";

print i;

print "):= ";

print result;

print "\n";

i := i + 1;

}

print "\n\n";

i := 0;

while(i<10) {

var a:int := 2;

result := algorithm\_2(a,i);

print "algorithm\_2(";

print a;

print "^";

print i;

print "): ";

print result;

print "\n";

i := i + 1;

}

print "\n\n";

i := 0;

var a:int := 10;

var arr\_result:array<int> := new int[a];

print "algorithm\_3(";

print a;

print "): [ ";

while(i<a) {

arr\_result := algorithm\_3(a);

print arr\_result[i];

print " ";

i := i + 1;

}

print "]";

}