PSOFT HW 2

Joel Grimaldi

March 2021

1

a)
x*y + result = m*n

b)
Because x is set to value m and y is set to value n and result is initialized at 0, the invariant will hold prior to the loop occurring.
x*y + 0 = m*n
x*y = m*n
x = m, y = n
therefore we can replace x and y:
m*n = m*n, True

c)
x*y + result = m*n

If y is even:
x = 2x, y = y/2
2x*$\frac{y}{2}$ + result = m*n
=> x*y + result = m*n

If y is odd:
result = result + x, y = y-1
x + x(y-1) + result = m*n
x + x*y - x + result = m*n
=> x*y + result = m*n

d)
y = 0
x*0 + result = m*n
=> result = m*n

e)
$D = y$
Y will be cut in half when even
Y will be reduced by 1 when odd
The loop will exit when $y = 0$
Therefore a working decrementing function is $D=y$

2

The following code will initialize k to the amount of REDs in the array and search from i to k for a BLUE, if a BLUE is found, the second while loop will be initiated to look for a RED in the section of the array after k. Once this RED is found, the target RED and BLUE will be swapped. Process will continue until the i reaches k.

-PSEUDOCODE-

```
char[ ] (char[ ] arr){
    if(arr == null){ return null; }
    int k = 0;
    for(char color : arr){
        if(color == 'r') { k++ }
    }
    int i = 0;
    int j = k;
    while(i < k){
        if(arr[i] == 'b'){
            while(j < arr.len){
                if(arr[j] == 'r'){
                    swap(arr,i,j);
                    break;
                }
                j++;
            }
        }
        i++;
    }
    return arr;
}
```

Postcondition: $i = k$ && $k \leq j \leq N$(arr.len)

-WITH INVARIANTS-

```
char[ ] (char[ ] arr){
    if(arr == null){ return null; }
    int k = 0;
    for(char color : arr)
        invariant 0 ≤ k < N(arr.len)*****
    {
        if(color == 'r') { k++ }
    }
    int i = 0;
    int j = k;
    while(i < k)
        invariant 0 ≤ i ≤ k*****
    {
        if(arr[i] == 'b'){
            while(j < arr.len)
                invariant k ≤ j ≤ N(arr.len)*****
            {
                if(arr[j] == 'r'){
                    swap(arr,i,j);
                    break;
                }
                j++;
            }
        }
        i++;
    }
    return arr;
}
```

3 Problem 3

```
function Factorial(n: int): int
    requires n >= 0
{
    if n == 0 then 1 else n * Factorial(n-1)
}

    method LoopyFactorial(n: int) returns (u: int)
    requires n >= 0
    ensures u == Factorial(n)
    {
        u := 1;
        var r := 0;
        while (r < n)
            invariant u == Factorial(r)
            invariant r <= n
            decreases n-r
        {
            var v := u;
            var s := 1;
            while (s<=r)
                invariant u == v*s
                invariant s <= r+1
                decreases r-s
            {
                u:=u+v;
                s:=s+1;
            }
            r:=r+1;
            assert u == Factorial(r) && r == s;
        }
    }
```

4 Proofs

Proof of the base case of inner loop:
Before the base case we set var v := u and set s:= 1
Therefore u == v*1 because we set v to u
Essentially v == v*1 => v == v

Proof for the inner loop induction:
u' = u+v, s' = s+1, v' = v
u' = v'*s'
u + v = v*(s+1)
u + v = vs + v
u = vs + v - v
u = v*s

Proof for the outer loop base case:
u == 1 and !r == 1 => !0 == 1
Therefore u == Factorial(r)

Proof for the outer loop induction:
u' = u + r*v, r' = r + 1, v' = u
u' = Factorial(r')
u + r*v = Factorial(r+1)
u = Factorial(r+1) - Factorial(r)
u = Factorial(r)