***Proofs, Outlines, and Total Correctness***

*CS 536: Science of Programming*

# Problems [50 points total]

For Problems 1 and 2, first study the (incomplete) formal proof of correctness that follows. In it,

* *p* ≡ 1 ≤ k ≤ n ∧ r = k² ∧ s = ssq(k)
* ssq(k) = 1² + 2² + … + k² if k ≥ 1 (If k ≤ 0, then ssq(k) = 0)
* *q* ≡ s = ssq(n).

The proof:

1. {n ≥ 1} k := 1 {n ≥ 1 ∧ k = 1} Assignment
2. {n ≥ 1 ∧ k = 1} r := 1 {*p*₁} Assignment
3. {*p*₁} s := 1 {*p*₂} Assignment
4. *p*₂ → *p* Predicate logic
5. {*p*₁} s := 1 {*p*} Postcond. weak 3, 4
6. {n ≥ 1} k := 1; r := 1 {*p*₁} Sequence 1, 2
7. {n ≥ 1} k := 1; r := 1; s := 1 {*p*} Sequence 6, 5
8. {*p*₃} k := k+1 {*p*} Assignment
9. {*p*₄} s := s+r {*p*₃} Assignment
10. {*p*₄} s := s+r; k := k+1 {*p*} Composition 9, 8
11. {*p*₅} r := r+2\*k+1 {*p*₄} Assignment
12. *p* ∧ k ≠ n → *p*₅ Predicate logic
13. {*p* ∧ k ≠ n} r := r+2\*k+1 {*p*₄} Precond Str 12, 11
14. {*p* ∧ k ≠ n} *S*₁ {*p*} Composition 13, 10

where *S*₁ ≡ r := r+2\*k+1; s := s+r; k := k+1

1. { **inv** *p*} **while** k ≠ n **do** *S*₁ **od** {*p*₆} while, 14
2. *p*₆ → *q* Predicate logic
3. { **inv** *p*} *W* {*q*} Postcond Weak 15, 16
4. { n ≥ 1} s := 1; r := 1; k := 1; *W* {*q*} Composition, 7, 17
5. [12 = 6\*2 pts] Give definitions for the requested predicates. Use substitution notation if it applies but then also show the expansion of the substitution.
   1. *p*₁
   2. *p*₂
   3. *p*₂
   4. *p*₃
   5. *p*₄
   6. *p*₅

*Ans:*

1. *p*₁ ≡ n ≥ 1 ∧ k = 1 ∧ r = 1
2. *p*₂ from {*p*₁} s := 1 {*p*₂} forward assignment

*p*₂ ≡ n ≥ 1 ∧ k = 1 ∧ r = 1 ∧ s = 1

1. {*p*₃} k := k+1 {*p*}≡ {*p*₃} k := k+1 { n ≥ 1 ∧ k = 1 ∧ r = 1 ∧ s := 1}

*p*₃≡ n ≥ 1 ∧ k+1 = 1 ∧ r = 1 ∧ s := 1

1. *p*₄ from {*p*₄} s := s+r {*p*₃} ≡ {*p*₄} s := s+r {n ≥ 1∧k+1=1∧r=1∧ s:= 1}

*p*₄ ≡ n ≥ 1∧k+1=1∧r=1∧ s+r:= 1

1. *p*₅ from {*p*₅} r := r+2\*k+1 {*p*₄} ≡{*p*₅} r := r+2\*k+1 { n ≥ 1∧k+1=1∧r=1∧ s+r:= 1}

*p*₅≡n ≥ 1∧k+1=1∧ r+2\*k+1 =1∧ s+ r+2\*k+1:= 1

1. *p*₆ ≡ k ≠ n ∧ *p*
2. [8 pts] Show the full proof outline the the proof for Problem 1. (You can just use *p*₁, *p*₂, etc. instead of their expansions, since you just gave them in Problem 1.)

Ans:

{n ≥ 1}

k := 1 {n ≥ 1 ∧ k = 1}

r := 1 {*p*₁}

s := 1 {*p*₂}

{ **inv** *p*}

**while** k ≠ n

**do**

{*p* ∧ k ≠ n}

{*p*₅}

r := r+2\*k+1 {*p*₄}

s := s+r; {*p*₃}

k := k+1 {*p*}

**od**

{*p*₆}

{*q*}

1. [4 pts] Show the minimal proof outline for the full outline for Problem 2.

Ans: {n ≥ 1}

k := 1

r := 1

s := 1

{ **inv** *p*}

**while** k ≠ n

**do**

r := r+2\*k+1

s := s+r;

k := k+1

**od**

{*q*}

1. [6 points] Expand the minimal outline below into a full proof outline for total correctness. Be sure to define the initial condition *p*₀, and don’t forget to add domain predicates *D*(…) as necessary.

{*p*₀} **if** b[j] ≥ 0 **then** x := sqrt(b[j]) **fi** {x = sqrt(b[j])}

Ans : Let p₀ ≡ D(sqrt(b[j])) ⇔ (0 ≤ j < size(b) ∧ b[j] ≥ 0). In the else branch, p₀ ∧ b[j] < 0 is a contradiction, which lets us conclude x = sqrt(b[j]).

{p₀}

if b[j] ≥ 0 then{p₀ ∧ b[j] ≥ 0}  
{D(sqrt(b[j])) ∧ sqrt(b[j] = sqrt(b[j])) }  
x := sqrt(b[j]) {x = sqrt(b[j])}  
else{p₀ ∧ b[j] < 0} {F} {x = sqrt(b[j])} skip {x = sqrt(b[j])}  
fi {x = sqrt(b[j])}

1. [10 = 5 \* 2 points] Given that *p* → K > 0 ∧ 0 ≤ i - K ≤ n (where K is a constant), consider the loop {**inv** *p*} {**bd** *t*} **while** i < n **do** … i := i+1 **od**.

For each of the following expressions, say whether or not it can be used as the bound expression *t* above (and if it can't, briefly explain why not.)

* 1. n (b) n - i (c) n - i + K (d) n + i + K (e) n² - i + K

Ans:

1. n fails as a bound because it's constant. (It does meet the criterion *p* → n ≥ 0.)
2. n - i fails because it can be < 0 (e.g., if n = 1 and i = K = 2). (It is decreased by the loop body, however.)
3. n - i + K can be used as a bound: The invariant implies that it's nonnegative, and i := i+1 decreases it.
4. n + i + K fails because it increases, not decreases. (The invariant does imply it's nonnegative, however.)
5. n²-i+K can be used as a bound: (1) Since *p* → n ≥ 0 → n² ≥ n, we know n²-i+K ≥ n-i+K ≥ 0 and (2) it’s decreased by the increment of i
6. [10 points] Expand the minimal outline below into a full proof outline for total correctness. (I.e., avoid runtime errors and loop divergence.) Feel free to extend our preliminary loop invariant *p* ≡ x = 2^n ≤ b[i] or to define other predicates. Show the expansion of any substitutions somewhere. (You can omit implications of adjacent conditions.)

{*p*₀} *S*₀;

{**inv** *p* ≡ x = 2^n ≤ b[i] ∧ ??? } {**bd** ???} **while** 2\*x ≤ b[i] **do**

x := 2\*x; n:= n+1

**od** {x = 2^n ≤ b[i] < 2^(n+1)}

Ans:

{y ≥ 1}

{1 = 2^0 ≤ y}

x:= 1; {x = 2^0 ≤ y}

n:= 0;

{**inv** *p* ≡ (x = 2^n ≤ y ∧ *q*)} where *q* ≡ *D*(b[i]) ⇔ 0 ≤ i < size(b)  
{**bd** y-x}  
**while** 2\*x ≤ b[i] **do**{*p* ∧ 2\*x ≤ y ∧ y-x = t₀}  
{2\*x = 2^(n+1) ≤ y ∧ *q* ∧ y-2\*x < t₀}  
x := 2\*x; {x = 2^(n+1) ≤ y ∧ *q* ∧ y-x < t₀}  
n:= n+1 {*p* ∧ y-x < t₀}  
**od**

{*p* ∧ 2\*x ≤ b[i]} {x = 2^n ≤ b[i] < 2^(n+1)}