# PSOFT
# Homework 2

Brian Wang-chen

Due February 13, 2024 @ 11:59PM

## Problem 1: Sum of natural numbers

```
// Precondition: n >= 0
int sumn(int n) {
    int i = 0, t = 0
    while (i < n) {
        i = i + 1;
        t = t + i;
    }
    return t;
}
// Postcondition: t = n * (n + 1) / 2
```

a) Find a suitable loop invariant

$\boxed{LI : i \leq \text{n}}$

$\boxed{LI : t = \frac{(i)(i+1)}{2}}$

b) Show that the invariant holds before the loop (base case)

**Base Case:**

when n $\geq 0$

$0 \leq$ n ✓since n is greater or equal to zero from preconditon while i = 0 before loop

Plug i into t equation, end up with sumn(0)

$t = \frac{(0)(0+1)}{2} = 0$, ✓

c) Show by induction that if the invariant holds after k-th iteration, and execution takes a k+1-st iteration, the invariant still holds (inductive step)

Assume that the invariant holds after kth iteration, prove that it holds after k+1 iteration

**Inductive step:** n + 1

i $\leq$ n+1 ✓while condition guarantees n to be greater while iterating

---

sumn(0) == 0, sumn(1) == 1, sumn(2) == 3

$0 + 1 + 3 + ... + \text{n} = \frac{(n)(n+1)}{2}$

$0 + 1 + 3 + ... + \text{n} + (\text{n} + 1) = \frac{(n+1)(n+2)}{2}$

$\frac{(n)(n+1)}{2} + (\text{n} + 1) = \frac{(n+1)(n+2)}{2}$

$\frac{2(n+1)+n(n+1)}{2} = \frac{(n+1)(n+2)}{2}$

$\frac{n^2+3n+2}{2} = \frac{n^2+3n+2}{2}$ ✓

**Proven that t(i + 1) is equal to t(i) + (i + 1), therefore by induction we have proven invariant holds**

d) Show that the loop exit condition and the loop invariant imply the postcondition t = n(n+1)/2

**Exit condition:** i == n

n ≤ n, holds

t = $\frac{(n)(n+1)}{2}$ → postcondition : t = n(n+1) / 2

**The Exit condition and LI imply postcondition because when exit condition is met, the loop invariant is the same equation as the post condition**

e) Find a suitable decrementing function. Show that the function is non-negative before loop starts, that it decreases at each iteration and that when it reaches 0 the loop is exited.

- Decrementing function: D == 0 → n - i == 0 → n == i
- Before loop: n = 0, i = 0, both set to zero before loop, non-negative
- Each iteration: Each iteration, i is incremented by 1 till it reaches n. It is decreasing because n - i will become smaller after each iteration as i increases until reached 0
- When reaches 0: When D reaches zero, n - i == 0 implying n is equal to i which should exit the loop because the while condition runs when n is greater than i.

f) Implement the sum of natural numbers in Dafny

- Comment out method Main in your Dafny code before you submit your code on Submitty.
- Method that implements the sum of natural numbers must be named sumn and have the following header: method sumn(n: int) returns (t: int)
- Make sure to include the precondition and the postcondition, as well as your invariant and the decrementing function.
- Verify your code with Dafny before submitting.
- Submit your Dafny code as a file named problem1.dfy in the answers/ folder

## Problem 2: Loopy square root

```
method loopysqrt(n:int) returns (root:int)
requires n >= 0
ensures root * root == n
{
    root := 0;
    var a := n;
    while (a > 0)
        //decreases //FILL IN DECREMENTING FUNCTION HERE
        //invariant //FILL IN INVARIANT HERE
        {
            root := root + 1;
            a := a - (2 * root - 1);
        }
}
```

a) Test this code by creating the Main() method and calling loopysqrt() with arguments like 4, 25, 49, etc. to convince yourself that this algorithm appears to be working correctly. In your answer, describe your tests and the corresponding output

I tested loopysqrt with a few perfect squares such as 49 and 9, and It did print the correct outputs of 7 and 3.

b) Yet, the code given above fails to verify with Dafny. One of the reasons for this is that it actually does have a bug. More specifically, this code may produce the result which does not comply with the specification. Write a test (or tests) that reveals the bug. In your answer, describe your test(s), the corresponding outputs, and the bug that you found. Also, indicate which part of the specification is violated

The post condition is violated. Specifically it does not work for non perfect squares, for example if 50 is inputted, 8 is returned which is incorrect because 8 squared is 64
Test: when n == 50, root == 8, post condition: root * root == n
**will not hold because 8 * 8 does not equal 50**

c) Now find and fix this bug. Note that there might be several different ways of fixing the bug. Use the method that you think would be the best. You are not allowed to change the header of loopysqrt() or add, remove, or change any specifications or annotations. Do not worry about Dafny verifying the code for now, just fix the bug and convince yourself that loopysqrt() is now correct. You are also required to keep the overall algorithm the same as in the original version of the code. In your answer, describe how you fixed the bug and show the output of the same tests you ran before after fixing the bug

> To fix bug I altered the while loop condition. I changed the while loop condition to (root + 1) * (root + 1) ≤ n && a >= 0 instead of a > 0. This considers both perfect and non perfect squares .
>
> Test: n == 50, root == 7, ✓which is correct since 50 square root is decimal around 7 to 8 not including them, which should translate to 7 since we are dealing with ints

d) Update the specification of loopysqrt() to match the way you fixed the bug. If you changed the postcondition, make sure that it is the strongest possible postcondition. In your answer, describe your changes and explain why they were necessary

> Changes:
> Postconditon:
> **ensures root * root ≤ n**
> **ensures (root + 1) * (root + 1) > n**
>
> The post conditions was changed from root * root == n because that will not consider non perfect squares, so it will not hold. The new post condition I created ensures that whatever the root is, adding one to it and squaring it will always be greater than the actual number, perfect or not perfect square. This condition will always hold. For root * root ¡= n, it should hold too because root * root should either give the perfect square number if n was a perfect square, or a number less than n if n was not a perfect square.

e) Does your Dafny code verify now? Why or why not? If it doesn't verify, does it mean that your code still has bugs in it?

> My dafny code still does not verify because although my preconditons and post conditions logically makes sense and should hold, my loops are missing invariants and a decrementing function which is causing it to fail.

f) If your Dafny code doesn't verify, uncomment invariant and/or decreases annotations and supply the actual invariant and/or decrementing function. Make sure your code now verifies. In your answer, describe how you guessed the invariant and/or decrementing function. Explain why your code was failing Dafny verification earlier but does verify now, despite the fact that you have not made any changes to your actual code (annotations are not part of the code)

> **LI: root * root ≤ n**
> **Decrementing function: n - (root * root)**
> My code was failing Dafny verification earlier but verifies now because adding the invariant and decrementing function confirms to Dafny that my loop actually terminates and produces the correct outputs from beginning of the loop, each iteration to the end of the loop. I came with the LI from the post condition because root * root has to be less than or equal to n at each iteration including the beginning and end otherwise the whole program would be incorrect. The decrementing function is simply is basically stating once the next n - (root * root) == 0 which means we have reached the perfect square, then the loop should exit.

g) Finally, remove the precondition from your Dafny code and make necessary changes to the remaining annotations and/or code ensure that code still verifies. You are not allowed to change the header of loopysqrt(). You are also required to keep the overall algorithm the same as in the previous versions of the code. As before, make sure that your postcondition is the strongest. Did removing the precondition make your code more difficult? What effect did removing the precondition have on the client of loopysqrt()?

Changes: I changed the invariant and post condition
ensures n $\geq$ 0 ==> root * root $\leq$ n
ensures n < 0 ==> root * root > n
invariant n $\geq$ 0 ==> root * root $\leq$ n
invariant a <= n
invariant n < 0 ==> root * root > n

Removing the precondition did make my code more difficult because without it loopy-sqrt allows negative numbers as inputs which made finding the square root difficult since that is impossible and the function will return zero. Removing the precondition broke my invariant because my invariant stated that root * root $\leq$ n, which will not hold if n is negative. Therefore I had to change my post condition and invariant. I changed my post condition to consider if n is a negative number, considering positives at the same time, using forall and implications. My invariants also used implications to ensured that negative numbers are also considered.

h) Use computational induction to prove by hand the total correctness of the final version of your Dafny code

**Base Case:**
ensures n $\geq$ 0 ==> root * root $\leq$ n ✓
ensures n < 0 ==> root * root > n ✓

---

**Post conditions holds because if n is 0 or greater, then the first post condition implies that n $\geq$ 0 which is true, implying root * root $\leq$ n which is true since the square root is always less or equal to n depending if it is a perfect square or not (decimals get rounded down). If n is less than zero, then the second post condition is satisfied because n < 0 is true, implying root * root > n which is true because if negative n, root == 0 and 0 * 0 will be greater than any negative number. This considers all possible values of n. **

---

invariant n $\geq$ 0 ==> root * root $\leq$ n ✓
invariant n < 0 ==> root * root > n ✓
invariant a <= n ✓

---

**Invariant hold for all values of n including negatives. when n >= 0, first invariant holds that n $\geq$ 0, true, implying that root * root $\leq$ true as we proved in the post condition. If n < 0 which is true proved in our post condition proof. invariant a <= n holds since a is set to n in beginning**

---

Assuming my code holds for kth iteration, prove k + 1 and k - 1using induction.
**Inductive steps: n + 1 or n - 1 (negative)**
ensures n $\geq$ 0 ==> (root+1) * (root+1) $\leq$ n ✓
ensures n < 0 ==> root * root > n ✓

---

** Post conditions hold on k + 1 iteration. First considering postive numbers including zero, n $\geq$ 0 holds. This implies that (root+1) * (root+1) $\leq$ n which holds because my while loop : (root + 1) * (root + 1) $\leq$ n && a >= 0 prevents root * root to ever be greater than n as it tests root + 1 * root + 1 before loopy body. It also ensures that a is greater or equal to 0 to ensure that as a is being decremented by odd numbers, it will not become negative. By induction it is proven that (root+1) * (root+1) $\leq$ n and that there will never reach iteration where root + 1 * root + 1 is greater than n ensuring that a will not be less than zero which is the tracker of what n is after decrementing it by each odd number. Applies to negatives as n $\geq$ 0 holds since a negative always less than zero. That implies root * root > n, since root cannot be less than zero cause our while loop does not run, which holds since 0 * 0 is greater than any negative number.**

---

invariant n $\geq$ 0 ==> (root+1) * root $\leq$ ✓
invariant n < 0 ==> root * root > n ✓
invariant a <= n ✓

---

** Invariant holds too as our invariant are the similar as our post condition, therefore our proofs for K + 1 and negatives in our post condition applies to our loop invariant too. After each iteration root + 1 * root + 1 still be less than or equal to n because root + 1 * root + 1 is checked in the while conditon while also checking the value of a to ensure it will never be greater than n. Invariant a >= 0 holds since a will not be greater than n because our while loop will stop before a becomes greater than n. Negatives apply too as root * root will always be greater than n since 0 * 0 is greater than any negative no matter how many decrements.

---

decreases n - (root * root)
** Decrementing function does decrement and eventually exit the while loop since as root increments, it will get closer to n, showing loop is decrementing, and will exit the loop once the root has reached the intended perfect square.

## Problem 3: Array of differences

```
Precondition: arr != null  arr.Length > 0
int[] difference(int[] arr) {
    diffs ← new int[arr.Length - 1]
    a ← 0
    while (a < diffs.Length)}
    {
        diffs[a] ← arr[a + 1] - arr[a]
        a ← a + 1
    }
    return diffs
}
Postcondition: diffs.Length = arr.Length - 1 ^
    for all k, s.t. 0 <= k < diffs.Length: diffs[k] = arr[k + 1] - arr[k]
```

a) Find a suitable loop invariant.

$\boxed{LI : 0 \leq \text{a} \leq \text{diffs.Length}}$

$\boxed{\text{LI: diffs.Length == arr.Length - 1}}$

$\boxed{\text{LI: for all K: int:: } 0 \leq \text{k} < \text{a} \rightarrow \text{diffs[k] = arr[k+1] - arr[k]}}$

b) Show that the invariant holds before the loop (base case).

**Base Case:**
$0 \leq$ a $\leq$ diffs.Length ✓
** precondition states arr.Length > 0, diffs.Length = arr.Length - 1. So diffs.Length is at least zero and a is 0 at beginning of loop so holds**
diffs.Length == arr.Length - 1 ✓
** Still holds as we are not changing the size of diffs and arr
$0 \leq$ k < a $\rightarrow$ diffs[k] = arr[k+1] - arr[k] ✓
** No possible number, skips this invariant **

c) Show by induction that if the invariant holds after k-th iteration, and execution takes a k+1-st iteration, the invariant still holds (inductive step).

**Inductive Step: a + 1**
$0 \leq a + 1 \leq$ diffs.Length, holds since while loop guarantees that after each iteration and in the beginning and end, a will be less than or equal to diffs.Length after increment

---

diffs.Length == arr.Length - 1 ✓
** Still holds as we are not changing the size of diffs and arr

---

diffs[a] == arr[a+1] - arr[a]. With new value of k = a + 1, diffs[k] = diffs[k+1] - diffs[k], won't change so still holds

d) Show that the loop exit condition and the loop invariant imply the postcondition result = i1+i2

**Loop exit condition: a == diffs.Length**
plug in for a
postcondition: LI: for all K: int:: $0 \leq$ k < a $\rightarrow$ diffs[k] = arr[k+1] - arr[k]
** loop exit condition plugged into loop invariant is same as post condition **
diffs.Length == arr.Length - 1 ✓

** Still holds as we are not changing the size of diffs and arr

e) Find a suitable decrementing function. Show that the function is not negative before loop starts,that it decreases at each iteration and that when it reaches 0 the loop is exited

**Decrementing function: D == 0 → diffs.Length - a == 0**

- Before Loop starts:
  a begins 0
  precondition states arr.Length > 0, diffs.Length = arr.Length - 1. So diffs.Length is at least zero
- Each iteration: diffs.Length - a, a is incremented until reached diffs.Length, so after each iteration, diffs.Length - a becomes smaller.
- Reaches 0: When diffs.Length - a == 0, indicates a == diffs.Length, then while loop condition no longer holds so loop exits.

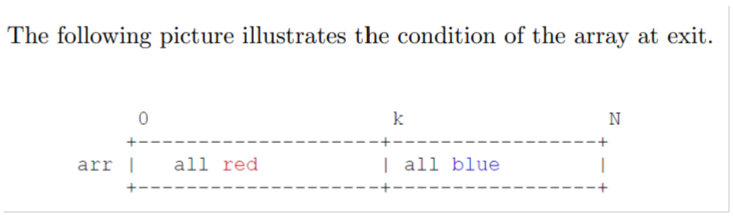
f) Implement the array of differences in Dafny

g) Extra credit (3 pts.) Implement difference in Dafny using sequences instead of arrays.

## Problem 4: The simplified Dutch National Flag Problem

a) Given an array arr[0..N-1] where each of the elements can be classified as red or blue, write pseudocode to rearrange the elements of arr so that all occurrences of blue come after all occurrences of red and the variable k indicates the boundary between the regions. That is, all arr[0..k-1] elements will be red and elements arr[k..N-1] will be blue. You might need to define method swap(arr, i, j)which swaps the ith and jth elements of arr.

```
    # rearrange array to be red then blue
def rearrange(arr):
    i = 0
    j = len(arr)
    k = 0
    # iterate arr with a front and back pointers until reached middle
    while (Bpointer > Fpointer):
        if arr[j - 1] == 'b':
            j -= 1
        elif arr[i] == 'r':
            i += 1
        else:
            arr[i] = 'r'
            arr[j - 1] = 'b'
            j -= 1
    k = j
    return k
```


The following picture illustrates the condition of the array at exit.

```
        0                   k                N
        +-------------------+-----------------+
    arr |   all red         | all blue        |
        +-------------------+-----------------+
```

b) Write an expression for the postcondition.

$0 \leq k \leq$ arr.Length
forall n: int :: $0 \leq n < k$ ==> arr[n] == 'r'
forall m: int :: $k \leq m <$ arr.Length ==> arr[m] == 'b'

c) Write a suitable loop invariant for all loops in your pseudocode

```
LI: 0 ≤ i ≤ arr.Length  0 ≤ j ≤ arr.Length
LI: forall f: int :: 0 ≤ f < i ==> arr[f] == 'r'
LI: forall b: int :: j ≤ b < arr.Length ==> arr[b] == 'b'
```

d) Implement your pseudocode in Dafny