10.1 Loop invariants

Program correctness

The field of ***program verification*** is concerned with formally proving that programs perform correctly. A program's correct behavior is defined by stating that if a ***pre-condition*** is true before the program starts, then the program will end after a finite number of steps and a ***post-condition*** is true after the program ends. Ex: If a program must sort 3 numbers like (50, 22, 93):

- Pre-condition: The input is a sequence of three numbers.
- Post-condition: The output is a reordering of the three numbers so that each number is less than or equal to the next number in the sequence. For input (50, 22, 93), the correct output would be (22, 50, 93).

Table 10.1.1: Examples of program pre-conditions and post-conditions.

Program description	Pre-condition	Post-condition
Sort a list of numbers.	Input is a positive integer n, and a list of n numbers $a_1, \ldots, a_n$.	Output is $b_1, b_2, \ldots, b_n$, a reordering of $a_1, \ldots, a_n$ such that $b_j \leq b_{j+1}$, for every $j \in \{1, \ldots, n-1\}$.
Find a given number in a list of numbers.	Input is a positive integer n, a number x, and a list of n numbers $a_1, \ldots, a_n$.	Output is an integer m such that $1 \leq m \leq n$ and $a_m = x$ or -1 if $x \neq a_j$ for every $j \in \{1, \ldots, n\}$
Compute the square root of a non-negative real number.	Input is a real number x such that $x \geq 0$.	Output is a real number y such that $y \geq 0$ and $y^2 = x$.

PARTICIPATION ACTIVITY 10.1.1: Selecting a pre-condition for a program.

Match each program description to the best pre-condition.

If unable to drag and drop, refresh the page.

Compute the cube root of a number **Compute the log base 2 of a number**

Multiply two numbers **Determine whether a sequence of numbers is increasing**

The input is a real number x

The input variables x and y are two real numbers

The input variables are n, a positive integer, and a sequence of n numbers $a_1, \ldots, a_n$.

The input is a real number x such that $x > 0$.

Reset

PARTICIPATION ACTIVITY | 10.1.2: Selecting a post-condition for a program.

Match each program description to the best post-condition. The pre-condition for each program is that the input is a positive integer n and a list of n numbers $a_1, \ldots, a_n$.

If unable to drag and drop, refresh the page.

Find a minimum number in a list of numbers

Find a maximum number in a list of numbers **Compute the sum of a list of numbers**

Compute the average of a list of numbers

m is an integer such that $1 \leq m \leq n$ and $a_m \geq a_j$ for every $j \in \{1, \ldots, n\}$.

m is an integer such that $1 \leq m \leq n$ and $a_m \leq a_j$ for every $j \in \{1, \ldots, n\}$.

A number x such that $x = \sum_{j=1}^{n} a_j$.

A number x such that $x = \frac{1}{n} \sum_{j=1}^{n} a_j$.

Reset

A program analyst can show that a large program performs correctly by breaking the program into smaller segments, each segment having a pre-condition and post-condition. The post-condition for one segment is the pre-condition for the next segment. Then, for each segment, the analyst must *prove* that if the pre-condition for that segment is true before the segment executes, then the post-condition for the segment is true after the segment executes.

Figure 10.1.1: Pre-conditions and post-conditions for a program that computes the minimum of three values.

ComputeMin(x, y, z)

// Pre-condition for the program = pre-condition for Segment 1
// [x, y, z are three numbers]

min := x; //Segment 1

// Post-condition for Segment 1 = pre-condition for Segment 2
// [$min \in \{x\}$ and $min \leq x$]

If ($y \leq min$), then min := y; //Segment 2

// Post-condition for Segment 2 = pre-condition for Segment 3
// [$min \in \{x, y\}$, $min \leq x$, and $min \leq y$]

If ($z \leq min$), then min := z; //Segment 3

// Post-condition for Segment 3 = post-condition for the program
// [$min \in \{x, y, z\}$, $min \leq x$, $min \leq y$, and $min \leq z$]

Return(min)

PARTICIPATION ACTIVITY | 10.1.3: Pre-conditions and post-conditions for a program that computes the sum of three numbers.

Select the statement that corresponds to each assertion.

If unable to drag and drop, refresh the page.

sum = 0 **x, y, and z are numbers** **sum = x + y** **sum = x** **sum = x + y + z**

ComputeSum(x, y, z)

// [Assertion 1] Pre-condition for the program = pre-condition for Segment 1

sum := 0 //Segment 1

// [Assertion 2] Post-condition for Segment 1 = pre-condition for Segment 2

sum := sum + x //Segment 2

// [Assertion 3] Post-condition for Segment 2 = pre-condition for Segment 3

sum := sum + y //Segment 3

// [Assertion 4] Post-condition for Segment 3 = pre-condition for Segi

Reset

sum := sum + z //Segment 4

// [Assertion 5] Post-condition for Segment 4 = post-condition for the program

Return(sum)

Loop invariants for while loops

This material focuses on showing that a program segment consisting of a while-loop performs correctly. A while loop has the form shown below, where C is a loop condition that evaluates to true or false. If the ***loop condition*** for a while loop is true, the instructions inside the loop are executed. Otherwise, the loop terminates and the next statement after the while loop is executed.

```
While (C)
  [List of instructions]
End-while
```


A ***loop invariant*** is an assertion that is true before each iteration of a loop. The animation below illustrates the use of a loop invariant to establish that the given while-loop correctly computes x^n. There are four steps in using a loop invariant to show that a while loop performs correctly.

Figure 10.1.2: Steps in using a loop invariant.

Given a while loop with condition C and a loop invariant I, the four steps below are sufficient to establish that if the pre-condition is true before the loop, then the post-condition is true after the loop:

1. Show that if the pre-condition is true before the loop begins, then I is also true.
2. Show that if C and I are both true before an iteration of the loop, then I is true after the iteration.
3. Show that the condition C will eventually be false.
4. Show that if ¬C and I are both true, then the post-condition is true.

PARTICIPATION ACTIVITY 10.1.4: The four steps of using a loop invariant.

Animation captions:

1. Pre-condition is that n is a non-negative integer, j = 0, and power = 1. Post-condition says that power = x^n. The loop invariant is that j is an integer $j \leq n$, and power = x^j.
2. Step 1 assumes the pre-condition and proves the loop invariant. For the power function, we assume n is non-negative, j = 0, and power = 1 and prove that j is an integer, $j \leq n$, and power = x^j.
3. Step 2. A subscript 1 denotes values of variables j and power before an iteration and subscript 2 denotes values after the iteration.
4. Assume $j_1 < n$, j_1 is an integer, and $power = x^{j_1}$. Prove that j_2 is an integer such that $j_2 \leq n$, and $power_2 = x^{j_2}$.
5. Step 3 shows that the loop condition $j < n$ will eventually be false.
6. Step 4 shows that if the loop condition is false and the loop invariant is true then the post-condition will be true.

PARTICIPATION ACTIVITY 10.1.5: Completing the proof using a loop invariant for the power function: Step 1.

The loop to compute x^n is given below:

Step 1 shows that if the pre-condition is true, then the loop invariant is true before the first iteration of the loop.

Pre-condition: n is a non-negative integer, $j = 0$, and $power = 1$
Loop invariant: j is an integer such that $j \leq n$ and $power = x^j$.

While ($j < n$)
  power := power * x
  j := j + 1
End-while

1) Step 1, part A.
Assume: n is a non-negative integer, j = 0, and power = 1
Prove: $power = x^j$

x^j = x^0 = 1 = power.

Which of the three equalities does *not* rely on the assumption?

- x^j = x^0
- x^0 = 1
- 1 = power

2) Step 1, part B.
Assume: n is a non-negative integer, j = 0, and power = 1
Prove: j is an integer and $j \leq n$.

(Line 1) If n is non-negative then
$j = 0 \leq n$
(Line 2) Since j = 0, then $j \leq n$
(Line 3) Since j = 0, then j is an integer

Which fact is *not* used in any of the lines of the proof?

- n is an integer
- n is non-negative
- j = 0

PARTICIPATION ACTIVITY | 10.1.6: Completing the proof using a loop condition for the power function: Step 2.

The loop to compute x^n is given below:

Step 2 shows that if the loop condition and the loop invariant are both true before an iteration of the loop, then the loop invariant is true after the iteration of the loop.

Variables j and power are the two variables whose value changes during an iteration of the loop. j_1 and $power_1$ will denote the values before the iteration, and j_2 and $power_2$ will

```
While (j < n)
  power := power * x
  j := j + 1
End-while
```

denote the values after the iteration.
Loop condition: $j_1 < n$
Loop invariant before the iteration: j_1 is an integer, $j_1 \leq n$, and $power_1 = x^{j_1}$
Loop invariant after the iteration: j_2 is an integer, $j_2 \leq n$, and $power_2 = x^{j_2}$.

1) What is the correct relationship between j_1 and j_2?

- $j_2 = j_1 + 1$
- $j_2 = j_1$
- $j_1 = j_2 + 1$

2) What is the correct relationship between $power_1$ and $power_2$?

- $power_2 = power_1 \cdot x$
- $power_1 = power_2 \cdot x$.
- $power_2 = x^{j_2}$

3) Step 2, part A.
Assume: $j_1 < n$, j_1 is an integer, and $power_1 = x^{j_1}$.
Prove: $power_2 = x^{j_2}$.

$$power_2 = power_1 \cdot x = x^{j_1} \cdot x = x^{j_1+1} = x^{j_2}$$

Which fact in the assumptions is used in the argument?

- $j_1 < n$
- j_1 is an integer
- $power_1 = x^{j_1}$

4) Step 2, part B.
Assume: $j_1 < n$, j_1 is an integer, and $power_1 = x^{j_1}$.
Prove: j_2 is an integer and $j_2 \leq n$.

(Line 1) Since $j_2 = j_1 + 1$ and j_1 is an integer, then j_2 is also an integer.
(Line 2) If j_1 is an integer and

$j_1 < n$, then $j_1 \leq n - 1$.
(Line 3) Since $j_1 \leq n - 1$, then
$j_1 + 1 \leq n$.
(Line 4) Since $j_2 = j_1 + 1$ and
$j_1 + 1 \leq n$, then $j_2 \leq n$.

Which fact in the assumptions is not used in the argument?

- $j_1 < n$
- j_1 is an integer
- $power_1 = x^{j_1}$

PARTICIPATION ACTIVITY | 10.1.7: Completing the proof using a loop condition for the power function: Steps 3 and 4.

The loop to compute x^n is given below:

```
While (j < n)
  power := power * x
  j := j + 1
End-while
```

Step 3 shows that the loop condition will eventually be false. Step 4 shows that if the loop condition is false and the loop invariant is true then the post-condition is true.

Loop condition: $j < n$
Loop invariant: j is an integer, $j \leq n$, and $power = x^j$
Post-condition: $power = x^n$.

1) Step 3.
Prove: The condition $j < n$ will eventually be false.

Which fact can be used to justify that fact $j < n$ will eventually be false?

- After n iterations, $power = x^n$,
- After $n - 1$ iterations, $j = n - 1$
- After n iterations, $j = n$.

2) Step 4.
Assume: j is an integer, $j \leq n$, $power = x^j$, and $\neg(j < n)$.
Prove: $power = x^n$

(Line 1) Since $j < n$ is false, then $j \geq n$.
(Line 2) Since $j \geq n$ and $j \leq n$, then $j = n$.
(Line 3) Since $j = n$ and $power = x^j$, then $power = x^n$.

Which fact in the assumptions is not used in the argument?

- j is an integer
- $j \leq n$
- $power = x^j$
- $\neg(j < n)$

Here are all the steps put together showing that the loop correctly computes the x^n.

```
While (j < n)
  power := power * x
  j := j + 1
End-while
```

Proof 10.1.1: A complete proof using a loop invariant.

- Pre-condition: n is a non-negative integer, $j = 0$, and $power = 1$
- Post-condition: $power = x^n$
- Loop invariant: j is an integer such that $j \leq n$ and $power = x^j$.

Proof.

Step 1. Assume that n is a non-negative integer, $j = 0$, and $power = 1$. We will prove that j is an integer such that $j \leq n$ and $power = x^j$.

Since j = 0 and power = 1, x^j = x^0 = 1 = power. Since n is a non-negative integer, $n \geq 0 = j$. Also, since j = 0, j is an integer.

Step 2. Let j_1 and $power_1$ denote the values of j and power before an iteration of the loop. Let j_2 and $power_2$ denote the values of j and power after the iteration. Assume that $j_1 < n$, j_1 is an integer, and $power_1 = x^{j_1}$. Prove that j_2 is an integer such that $j_2 \leq n$, and $power_2 = x^{j_2}$.

The first line of the loop multiplies power by x, so $power_2 = power_1 \cdot x$. The second line increments j by 1, so $j_2 = j_1 + 1$.

Since j_1 is an integer and $j_1 < n$, then $j_1 \leq n - 1$. Adding 1 to both sides of the inequality yields that $j_1 + 1 \leq n$, which means that $j_2 \leq n$. Also,

$$power_2 = power_1 \cdot x = x^{j_1} \cdot x = x^{j_1+1} = x^{j_2}$$

Finally, since j_1 is an integer, then j_2 is also an integer.

Step 3. Since the value of j is n after n iterations of the loop, the condition $j < n$ will be false after n iterations. Therefore the loop will eventually terminate.

Step 4. Assume that $j \leq n$, $power = x^j$, and the condition $j < n$ is false. We will prove that $power = x^n$.

Since $j < n$ is false, then $j \geq n$. If $j \geq n$ and $j \leq n$, then $j = n$. Therefore $power = x^j = x^n$. ■

PARTICIPATION ACTIVITY | 10.1.8: Identifying the steps in using a loop invariant.

The loop below computes the sum of a list of numbers.

- Pre-condition: j = 1, $sum = a_1$, n is a positive integer, $a_1, \ldots, a_n$ is a list of n numbers.
- Post-condition: $sum = \sum_{k=1}^{n} a_k$.

While ($j < n$)
  sum := sum + a_{j+1}
  j := j + 1
End-while

- Loop invariant: j is an integer, $j \leq n$, and $sum = \sum_{k=1}^{j} a_k$

Fill in the blanks to express what must be proven in each step. j_1 and sum_1 denote the values of j and sum before an iteration of the loop, and j_2 and sum_2 denote the values after the iteration.

1) Step 1.
Assume that n is a positive integer, j = 1, and $sum = a_1$.
Prove (?).

- $sum = \sum_{k=1}^{n} a_k$
- j is an integer, $j \leq n$, and $sum = \sum_{k=1}^{j} a_k$.
- $j < n$

2) Step 2. Assume that (?), j_1 is an integer, and $sum_1 = \sum_{k=1}^{j_1} a_k$.
Prove that j_2 is an integer, $j_2 \leq n$, and $sum_2 = \sum_{k=1}^{j_2} a_k$.

- $j_1 < n$
- $j_2 < n$
- $j_1 \geq n$

3) Step 3. After a finite number of iterations (?).

- $j < n$
- $j \leq n$
- $j \geq n$

4) Step 4. Assume that (?), $j \leq n$, and $sum = \sum_{k=1}^{j} a_k$.
Prove that $sum = \sum_{k=1}^{n} a_k$.

- $j < n$
- $j \leq n$
- $j \geq n$

Additional exercises

EXERCISE 10.1.1: Proving the correctness of while loops using loop invariants.

For each while loop, use the loop invariant given to show that if the pre-condition is true before the loop then the post-condition is true after the loop.In each step, clearly state what facts are assumed and what facts will be proven.

(a) The loop below computes the product of two non-negative integers.

While ($j < n$)
prod := prod + m
j := j + 1
End-while

- Pre-condition: m and n are non-negative integers. j = 0 and prod = 0.
- Post-condition: $prod = m \cdot n$
- Loop invariant: j is an integer, $j \leq n$, and $prod = m \cdot j$

(b) The loop below computes the sum of a list of numbers.

While ($j < n$)
sum := sum + a_{j+1}
j := j + 1
End-while

- Pre-condition: j = 1, $sum = a_1$, n is a positive integer, $a_1, \ldots, a_n$ is a list of n numbers.
- Post-condition: $sum = \sum_{k=1}^{n} a_k$.
- Loop invariant: j is an integer, $j \leq n$, and $sum = \sum_{k=1}^{j} a_k$

(c) The loop below finds a maximum value in a list of numbers.

While ($j < n$)
If ($a_{max} < a_{j+1}$), then max := j+1
j := j + 1
End-while

- Pre-condition: j = 1, max = 1, n is a positive integer, $a_1, \ldots, a_n$ is a list of n numbers.
- Post-condition: $1 \leq max \leq n$ and $a_k \leq a_{max}$ for every $k \in \{1, \ldots, n\}$.
- Loop invariant: j and max are integers such that $1 \leq max \leq j \leq n$. Also, for every $k \in \{1, \ldots, j\}$, $a_k \leq a_{max}$.

10.2 Programming Loop Invariants

PA2: Loop Invariants

This second python programming assignment is about loop invariants. You will write a function `eExp(left,right)` that computes exponentials `left ** right` in a similar fashion as the `egyptian_multiplication` function computes products, as discussed in the loop invariants lecture.

The starter code contains some skeleton code. Study egyptian multiplication in the lecture slides. The program logic in egyptian_multiplication, and thus the loop invariant is based on the fact that

```
a * b = if odd(a): b + (a//2)*(b*2)
        else: (a//2)*(b*2)
```

and that p stepwise gathers the product.

For your exponentiation code, the program logic, and thus the loop invariant, is based on the fact that

```
n ** k = if odd(k): n * (n*n)**(k//2)
         else (n*n)**(k//2)
```

and that e stepwise gathers the exponential.

Your job is to complete the code **INCLUDING** the correct assert statements to check the loop invariant, loop test and their combination, as indicated in the skeleton code. Leave the print statements in place. Be sure to use the invariant function, as we will call this directly with different values to check that it is correct. A correct implementation of `eExp`:

```
python3 eExp.py 2 11
```

produces


```
program: eExp.py 2 11
   n: 2 k: 11 e: 1
   n: 4 k: 5 e: 2
   n: 16 k: 2 e: 8
   n: 256 k: 1 e: 8
k: 0 e: 2048
2 ** 11 = 2048
```

422102.2723990.qx3zqy7

LAB ACTIVITY | 10.2.1: Programming Loop Invariants 0 / 100

eExp.py

Load default template...


```
import sys

def invariant(n, k, e, left, right):
  return True

def eExp(left, right):
# precondition: left>0 AND right>0
  if left <= 0 or right <= 0 : raise "eExp: invalid inputs!"
  n=left; k=right; e=1  #e: the exponent
  assert invariant(n, k, e, left, right) # fill in the proper loop invariant
  while (False) : # the loop test
    assert True and invariant(n, k, e, left, right) # fill in the proper loop test and loo
    print("    n:",n,"k:",k,"e:",e)
    # body
    assert invariant(n, k, e, left, right) # fill in the proper loop invariant
  print("k:",k,"e:",e)
```

Develop mode **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run command

`python3 eExp.py` Additional arguments

Run program

Input (from above) → **eExp.py** (Your program) → O

Program output displayed here

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.