SAMPLE SOLUTION

IMPORTANT FIRST STEPS:

1. **Close your laptops and put them away.**
2. **Form a group of 2-3 students.**
3. **Clearly put your names and IDs on 1 copy of this worksheet.**
4. **Be sure to turn this exercise in at the end of class.**

Proof of Correctness using Loop Invariants

Review of Mathematical Induction:

We can use *mathematical induction,* a style of mathematical proof, to establish the truth of a given statement, *S(n)*, over an infinite set (usually natural numbers):

First, prove the *base case* (usually *n*=0 or *n*=1)

Then, prove that

if *any one* statement in the sequence of statements is true (the *inductive hypothesis)*,

the *very next one* (*n*+1) must be true (the *inductive step)*.

That is, we wish to prove that in a world where S(n) happens to be true, it **must be the case** that S(n+1) holds. I.e.:

S(n) -> S(n+1)

Note that this says nothing about the *actual* truth of S(n), only something about the case where it happens to be true (we likewise could say something about the case where it is false, too, it just so happens that's not interesting to us here).

So we have:

1. **Basis**: Prove the theorem for the simplest case

2. **Inductive Hypothesis**: Assume the theorem holds for some arbitrary element, e

3. **Inductive Step**: Show that the theorem holds for the successor of e

4. **Conclusion**: Together, 1-3 imply the theorem holds for all possible cases (the minimal case and all successors)

...*you must clearly show these steps!*

SAMPLE SOLUTION

Review/Exercise:

Use mathematical induction to show that the sum of the odd integers from 1 to ($2n$-1) is n^2.

P(n): $1 + 3 + 5 + \dots + (2n-1) = n^2$

Base case, P(1).

LHS: 1

RHS: 1

LHS = RHS, therefore P(1) holds.

Inductive Hypothesis.

Assume that P(k) is true (i.e. $1 + 3 + 5 + \dots + (2k-1) = k^2$)

Inductive Step, P(k+1).

LHS: $1 + 3 + 5 + \dots + (2(k+1) -1)$

$1 + 3 + 5 + \dots + (2k - 1) + (2(k+1) - 1)$ (expand to show second-to-last #)

$k^2 + (2(k+1) - 1)$ (by IH)

$k^2 + 2k + 1$

RHS: $(k+1)^2$

$k^2 + 2k + 1$

LHS = RHS, therefore P(n) holds for all integers n >= 1.

Strong vs. Weak?

Weak (Ordinary): If the theorem holds for all cases *at* some arbitrary point n, then it holds for all cases at point n+1

S(n) -> S(n+1)

Strong: If the theorem holds for all cases *up to* some arbitrary point n, then it holds for all cases at point n+1

S(1) & S(2) & ... & S(n) -> S(n+1)

Very similar! In general we'll only worry about weak induction here (in fact they can be shown to be equivalent).

Recursion:

How does this relate to recursion? You must show that your algorithm satisfies the following:

- the base case is recognized and solved correctly

- each recursive case makes progress toward the base case; that is, any new problems generated are smaller versions of the original problem

- if all smaller problems are solved correctly, then the original problem is also solved correctly

How do we know that the *recursive* definition of **factorial** will work (see the code snippet on the final page)?

The base case is when n==0.

The recursive case involves a call to factorial_recur on a smaller size of n so it will terminate.

By multiplying n by one less than n each time, we will calculate the factorial of n (definition of factorial).

SAMPLE SOLUTION

Iteration (Proof of Correctness)

How do we know that the *iterative* definition of **factorial** will work?

We need to show a *proof of correctness...*

To do so, we need to establish a **loop invariant**:

A statement that is true at the beginning of a loop (or at a given point) and remains true throughout the execution of the loop

Allows us to relate the *state of the program* (i.e. the data values) to the current iteration

We can then use mathematical induction on *i*, where *i* is an integer from the set of natural numbers representing the iterations of the program, starting with iteration 0, before you enter the loop.

Let's prove that the iterative definition of factorial from our notes (and provided at the back for reference) in fact correctly produces the correct result for any value of *n*

`factorial_iter(num)` = `num`!

Note that even though we are proving this for n, the actual induction is on the iteration, *i*. This is extremely important and a frequent source of confusion!

First, we need to establish the loop invariant. This can be a tricky step, but generally involves taking a look at a few iterations of the loop to figure out what the pattern is. Complete the following table using the call `factorial_iter(4)`.

Before we enter the loop

i	num	*f*
0	4	1
1	4	1
2	4	2
3	4	6
4	4	24

Can you see a pattern?

What's the invariant?

SAMPLE SOLUTION

I(n): "At the end of the *ith* iteration, *f=i!*"

Why is this the loop invariant?

(Note: instead of S(n), let's use I(n), to remind us it's an invariant proof and we're interested in the iterations; remember that *n=i* here!):

This is the loop invariant because it holds true for after every iteration of the loop, and before entry to the loop. f=i! predicts what `f` will be in the code.

Now let's prove that our loop works using the loop invariant. That is, show that the loop invariant holds, for all $n \in \mathbb{N}$. For the following, simply walk through the proof with your partner, making sure you understand each step clearly. You can find practice questions on page 253 of Epp (note that I'm not proving the eventual falsity of the guard here).

We'll use mathematical induction on *n,* where *n* is the iterations (sometimes people use *i* here).

Base case:

Thus, *f=i!*

	From the code we have:	Loop invariant predicts:
Before we enter the loop →	`int f = 1;`	0! = 1

Therefore, since LHS = RHS, or the code matches what was predicted, I(0) holds.

Inductive Hypothesis:

Assume that I(*n*) holds for *n* = k,
thus f=k!

Inductive Step (I.e. show for next iteration, *n*=*k*+1, given the inductive hypothesis):

During the *n*=*k*+1st iteration, the statement f *= i executes, giving us $f_{new}=f_{old}*(k+1)$.

By our inductive hypothesis $f_{old}=k!$, so the value of f_{new} is therefore k!(k+1)...

...which by the recursive definition of factorial, is equivalent to (k+1)!

Thus, *f*=*i!* (since *n*=*k*+1, recall). QED.

SAMPLE SOLUTION

```
#include <iostream> // provides: cout, endl
#include <cstdlib>  // provides: atoi
using namespace std;

int factorial_iter( int num ) {
  int f = 1;
  for( int i = 1; i <= num; ++i )
     f *= i;
  return f;
}

int factorial_recur( int num ) {
  if (num == 0) return 1;
  return num * factorial_recur(num-1);
}

int main(int argc, char * argv[]){
  int num = atoi(argv[1]);
  cout << num << "! = " << factorial_iter(num) << endl;
  cout << num << "! = " << factorial_recur(num) << endl;
  return 0;
}

// NOTE: argc = number of strings in command line
// e.g., argc is 2 if "factorial 3"
//   where argv[0] is the program name "factorial"
//        argv[1] is the string "3"
```