- In the previous lectures (Week 7) we introduced recursive definitions. From these definitions we constructed:
  - a recursive implementation (using recursion)
  - an iterative implementation (using a while loop)

- Note the terminology here:
  - A recursive definition is a mathematical definition of a function (using a base case and a recurrence relation)
  - An implementation (either recursive or iterative) involves writing Java code.

- Here is a reminder of the recursive definition, recursive implementation and iterative implementation of the factorial function.

- Recursive definition:

```
Factorial(0)=1
Factorial(n+1)=(n+1)*Factorial(n)
```

- Recursive Implementation:

```
int fac(int n)
{
    if n==0 return 1; else return n*fac(n-1);
}
```

- Iterative Implementation:

```
int fac(int n)
{
    int i, result;
    i=0; result=1;
    /* initialisation corresponding
                to the base case */
    while(i<n)
    {
        i=i+1; result=i*result;
        // implementation of recurrence relation
    }
    return result;
}
```

- Since the recursive definition is a mathematical definition we assume that this definition is correct.

- How do we know that the recursive and iterative implementations are correct?

  - The recursive implementation is very similar to the recursive definition.

  - The iterative implementation is more complex. Therefore, we need to show that the iterative implementation is correct.

- Two methods of proof:

  - Use the while rule (previous lecture)

  - Proof by induction (this lecture...see below)

- We want to prove that this program (iterative implementation) computes the same value as the function defined inductively. To do this follow the following steps:

  1. First construct a flowchart for the program (This will be done in class).

  2. Annotate the flowchart with *Assertions*.

  3. Consider the scenarios when the loop is never executed, is executed n and $n+1$ times.

- Some points about Assertions:
  - An assertion is a logical statement about the values in the programming variables.
  - An assertion is not a statement about the program variables. Programming variables are containers for values
  - The values in program variables are change over time.
  - The values in assertions (mathematical variables) do not change over time.
  - The effect of executing a command is captured by an assertion

- Notations for assertions.
  - $=$ instead of $==$ for equals
  - $\geq$ instead of $>=$ for greater than or equal to
  - $\wedge$ instead of && for and etc...

- How to distinguish between mathematical variables(in assertions) and programming variables(in Java code)
  - Names of mathematical variables stand for the *current* values of the program variables
  - We use lower case for program variables and upper case for mathematical variables (simply to help distinguish between the two)
  - When the effect of a command changes the value of a variable, we must be able to distinguish between the old value and the new value contained in the programming variable. From now on we will do this by adding a dash to the name of the mathematical variable to denote the new value.

- When we exit from a context where the old value is no longer needed we can let the undashed name stand for the dashed variable value

- Consider an example.

- Reasoning about the Program: to prove that the program implements the inductive definition answer the following questions:

  1. Does the initialisation satisfy the base case

  2. Does the loop maintain the recurrence relation between adjacent values of the inductive definition.

- After the initialisation the following condition is satisfied
  $I = 0 \wedge RESULT = 1$
  Therefore $I = 0 \wedge RESULT = fac(0)$, since $fac(0) = 1$
  $I = 0 \wedge RESULT = fac(I)$, since we can substitute I for 0 because I denotes the value 0 at this point

- There are 2 choices on encountering the loop condition for the first time:

  1. $N = 0$ and the test fails

  2. $N > 0$, the test succeeds and we enter the loop with $I = 0$

- If the test fails:
  $(I = 0) \wedge (I = N) \wedge (RESULT = fac(I))$ as a result of initialisation
  $(I = N) \wedge (RESULT = fac(N))$ as a consequence of $I = N = 0$

- If the test succeeds at least once, immediately prior to the loop body we assume the inductive hypothesis:
  $(I \leq N) \wedge (RESULT = fac(I))$

- As a result of executing the commands:

```
{ i = i+1;  result = i * result;}
```

We have

- $(I' = I + 1) \wedge (RESULT' = I' * RESULT)$

- $(I' = I + 1) \wedge (I' \leq N) \wedge (RESULT' = (I + 1) * RESULT)$

- $(I' = I + 1) \wedge (I' \leq N) \wedge (RESULT' = (I + 1) * fac(I))$

- $(I' = I + 1) \wedge (I' \leq N) \wedge (RESULT' = fac(I + 1))$

- $(I' \leq N) \wedge (RESULT' = fac(I'))$

- If the inductive hypothesis holds for I on entering the loop body, it also holds for I' on leaving the loop body.

- Note: $I' = I + 1$

- We've shown that:

  - The property holds for $I = 0$ and if it holds for a value of I which is $< N$ then it holds for $I' \leq N$

  - Therefore it holds for all values that can be assumed by I

- Summary. In this lecture we have:

  1. Created a recursive definition for a function.

  2. Used this definition to write some Java code to implement this function in two different ways (using a recursive and an iterative approach).

  3. Since the iterative approach is considerably different from the recursive definition we have proved that the iterative implementation implements the recursive definition correctly.

  4. Note that in this proof we used flowcharts which were constructed in class.