```
1. long iterative_factorial(int n)
2. {
3.   int i; long p=1;
4.   for(i=1; i<=n; i++)
5.       p=p*i;
6.   return p;
7. }
```

Loop invariant: The variable p stores the factorial of $(i-1)$ at the start of each iteration.

Initialization: Before the 1st iteration, p is initialized to 1. On line 4, we have $i=1$. The factorial of $(i-1=0)$ is 1. $\therefore$ Loop invariant holds

Maintenance: Suppose, before the iteration when $i=K$, the loop invariant holds. $\therefore p=(K-1)!$. When $i=K$, we reinitialize p as $p*i$. $\therefore$ New value of $p=(K-1)!*K=K!$. The loop variable gets updated as $i=K+1$. The loop invariant holds $\because$ p stores the value of $(i-1)!=((K+1)-1)!$ $=K!$

Termination: The loop terminates when $i=n+1$. By loop invariant, p stores the factorial of $(i-1)=((n+1)-1)=n$. This is what the function wanted to compute. The value of p is returned in line 6. □

8) The Fibonacci numbers are defined as: $f_0=0, f_1=1$, and $f_i=f_{i-1}+f_{i-2}$ for $i>1$. Write both a recursive and an iterative C function to compute f_i.

```
int recursive_nthFibonacci(int n)
{ if(n==0)
      return 0;
  else if(n==1)
      return 1;
  else
      return recursive_nthFibonacci(n-1)+ recursive_nthFibonacci(n-2);
}
```

Claim: The recursive_nthFibonacci(n) function correctly computes the nth Fibonacci number.