```
// recur1.cpp
// Finding the Sum of the Numbers from 1 to n using recursion
//
        #include <iostream>
        using namespace std;
        int   Summation ( /* in */  int   n ) ;

        int main()
        {
        cout << Summation(4) << endl;
        }

          int   Summation ( /* in */  int   n )
//   Computes the sum of the numbers from 1 to n by
//   adding n to the sum of the numbers from 1 to (n-1)
//   Precondition:    n is assigned   &&   n > 0
//   Postcondition:
//  Function value == sum of numbers from 1 to n
{
    if  ( n == 1)                              //  base case
                return  1 ;
        else                                   // general case
                return ( n + Summation ( n - 1 ) ) ;
}
output
10
```

```
// recur2.cpp
//
        #include <iostream>
        using namespace std;

        int   Factorial (  int   number );

        int main()
        {
        cout << Factorial(5) << endl;
        }

int   Factorial (  int   number )
//  Pre:   number is assigned and  number >= 0.
{
         if  ( number == 0)                 //  base case
               return  1 ;
        else                                // general case
               return  number * Factorial ( number - 1 )  ;
}
output
120
```

```
// recur3.cpp
/
        #include <iostream>
        using namespace std;

int  Power (  int   x,   int   n );

        int main()
        {
        cout << Power (3,5) << endl;
        }

        int  Power (  int   x,   int   n )

// Pre:     n >= 0.   x, n are not both zero
// Post:   Function value == x raised to the power n.

{
              if  ( n == 0 )
                    return  1;              //  base case
              else                           // general case
                    return (  x * Power ( x , n-1 ) ) ;
}
output
243
```

```
// recur4.cpp

//
        #include <iostream>
        using namespace std;


float Power (  float   x,   int   n );

        int main()
        {
        cout << Power (10,-3) << endl;
        }

        float  Power   (  /* in */  float   x,   /* in */  int   n )

// Precondition:   x  !=  0   &&   Assigned(n)
// Postcondition:  Function value == x raised to the power n.

{
                if  ( n == 0 )                  //  base case

                        return  1;


                else    if  ( n > 0 )               //  first  general  case

                        return (  x * Power ( x , n - 1 ) ) ;

                else                            //  second general case

                        return (  1.0 / Power ( x , - n ) ) ;

}
output
0.001
```

```
// recur5.cpp
//
        #include <iostream>
        using namespace std;

        void   PrintStars ( /* in */  int   n ) ;

        int main()
        {
          PrintStars(3) ;
        }

void   PrintStars( /* in */  int   n )
//   Prints n asterisks, one to a line
//   Precondition:    n is assigned
//   Postcondition:
//                    IF n > 0, n stars have been printed, one to a line
//                    ELSE no action has taken place
 {
         if  ( n > 0 )                          // general case
         {
               cout  <<  '*'  ;
               PrintStars ( n - 1 ) ;
         }
output
***
```

```
// recur6.cpp
//  print an array in reverse using recursion
        #include <iostream>
        using namespace std;

        void PrintRev( const  int  data[ ], int  first,  int  last );

        int main()
        {
                int data[10];
                for (int index=0 ; index < 10; index++)
                data[index]=index;

          PrintRev(data, 0, 2) ;
          cout << endl;
          PrintRev(data, 3, 9) ;
        }

void   PrintRev ( /* in */  const  int  data [ ] , // Array to be printed
                                        /* in */          int  first ,       // Index of
first element
                                        /* in */          int  last  )       // Index of
last element
//   Prints array elements data [ first. . . last ] in reverse order
//   Precondition:   first assigned  &&  last assigned
//                 &&  if  first <= last then data [first . . last ] assigned
{
         if  ( first  <=  last )              // general case
        {
                cout  <<  data [ last ]  << "    " ;           // print last element
                PrintRev ( data,  first, last - 1 ) ;        // then process the rest
        }
                                    // Base case is empty else-clause
}
output
2   1   0
9   8   7   6   5   4   3
```

```
// recur7.cpp
// A recursive function for a function having one parameter that
// generates the nth Fibonacci number.
// f(i+2)=fi+f(i+1)
#include <iostream>
#include <cmath>
using namespace std;

// The full recursive version:
//
unsigned long Fib1( int n );

int main()
{
char ans;
int N;
do
{
cout << "Display fibonacci numbers 0-N." << endl;
cout << "Enter an limit, please. Be patient! This recursive"
<< endl << "Fibonacci routine will take about 3 "
<< endl << "seconds for N = 46 alone" << endl;
cin >> N;
for ( int i = 0; i < N; i++ )
cout << Fib1(i) << endl;
cout << "Y/y to continue, anything else quits" << endl;
cin >> ans;
} while ( 'Y' == ans || 'y' == ans );
}

unsigned long Fib1( int n )
{
if (n == 0 || n == 1)
return 1;
return Fib1( n - 1 ) + Fib1( n - 2 );
}
output
Display fibonacci numbers 0-N.
```

Enter an limit, please. Be patient! This recursive
Fibonacci routine will take about 3
seconds for N = 46 alone
46
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
10946
17711
28657
46368
75025
121393
196418
317811
514229
832040
1346269
2178309
3524578
5702887

```
9227465
14930352
24157817
39088169
63245986
102334155
165580141
267914296
433494437
701408733
1134903170
1836311903
Y/y to continue, anything else quits
```

```
// recur8.cpp
#include <iostream>
using namespace std;

int fib(int n)
{
  /* Declare an array to store fibonacci numbers. */
  int f[n+1];
  int i;

  /* 0th and 1st number of the series are 0 and 1*/
  f[0] = 0;
  f[1] = 1;

  for (i = 2; i <= n; i++)
  {
     /* Add the previous 2 numbers in the series
       and store it */
     f[i] = f[i-1] + f[i-2];
  }

  return f[n];
}
```

```
int main()
{
char ans;
int N;
do
{
cout << "Display fibonacci numbers 1-N." << endl;
cout << "Enter an limit, please. Be patient! This recursive"
<< endl << "Fibonacci routine will take about 3 "
<< endl << "seconds for N = 46 alone" << endl;
cin >> N;
for ( int i = 1; i < N+1; i++ )
cout << i << " " << fib(i) << endl;
cout << "Y/y to continue, anything else quits" << endl;
cin >> ans;
} while ( 'Y' == ans || 'y' == ans );
return 0;
}
```

```
// recur9.cpp
 #include <iostream>
using namespace std;

bool isPrimeRecursive(int num, int divisor)
{
   cout << "Checking to see if " << num << " is divisible by " << divisor << endl;
   if(divisor == 1)
   {
      cout << num << " must be a prime number, as we got to the case where divisor
= 1";
      return true;
   }

   /*
```

```
   You may not have encountered the % (modulus) before. This operator gives us the
   remainder of a division. For example 10 % 3 returns 1, because 3 divides into 10 three times and has
   a remainder of 1. A remainder of 0 means that we were able to evenly divide two numbers (such as 4 / 2.)
   Since a prime number can only be divisible by itself and 1, we must return false if the result of the modulus is 0, as that
   means we found another number we can divide evenly by.
   */
   if(num % divisor == 0)
   {
      cout << num << " is evenly divisble by " << divisor << " thus it must not be a prime number" << endl;
      return false;
   }
   else
   {
      cout << num << " is not evenly divisible by " << divisor << ", recurse deeper" << endl;
      return isPrimeRecursive(num, divisor - 1);
   }
}

/*
   It sometimes helps to have a function that 'primes the pump' for recursion.
   This function here allows the user to just pass in num and not worry about the divisor param.
*/ bool isPrime(int num)
{
   cout << "Checking to see if " << num << " is prime" << endl;

   //1 is not a prime number.
   if(num <= 1)
   {
      cout << num << " is not prime" << endl;
      return false;
   }
```

```
    return isPrimeRecursive(num, num - 1);
}

int main()
{
  isPrime(2);
  cout << endl << endl;
  isPrime(6);
  cout << endl << endl;
  isPrime(11);
  return 0;
}
```

Output

```
Checking to see if 2 is prime
Checking to see if 2 is divisible by 1
2 must be a prime number, as we got to the case where divisor = 1

Checking to see if 6 is prime
Checking to see if 6 is divisible by 5
6 is not evenly divisible by 5, recurse deeper
Checking to see if 6 is divisible by 4
6 is not evenly divisible by 4, recurse deeper
Checking to see if 6 is divisible by 3
6 is evenly divisble by 3 thus it must not be a prime number


Checking to see if 11 is prime
Checking to see if 11 is divisible by 10
11 is not evenly divisible by 10, recurse deeper
Checking to see if 11 is divisible by 9
11 is not evenly divisible by 9, recurse deeper
Checking to see if 11 is divisible by 8
11 is not evenly divisible by 8, recurse deeper
Checking to see if 11 is divisible by 7
11 is not evenly divisible by 7, recurse deeper
Checking to see if 11 is divisible by 6
11 is not evenly divisible by 6, recurse deeper
```

Checking to see if 11 is divisible by 5
11 is not evenly divisible by 5, recurse deeper
Checking to see if 11 is divisible by 4
11 is not evenly divisible by 4, recurse deeper
Checking to see if 11 is divisible by 3
11 is not evenly divisible by 3, recurse deeper
Checking to see if 11 is divisible by 2
11 is not evenly divisible by 2, recurse deeper
Checking to see if 11 is divisible by 1
11 must be a prime number, as we got to the case where divisor = 1

// recur10.cpp

```
// indirect recursion
#include <iostream>     // std::cout
using namespace std;

bool isOdd(int no);
bool isEven(int no)
       {
           // termination condition
           if (0 == no)
               return true;
           else
               // mutual recursive call
               return isOdd(no - 1);
       }

       bool isOdd(int no)
       {
           // termination condition
           if (0 == no)
               return false;
           else
               // mutual recursive call
               return isEven(no - 1);
       }


int main ()
{
       if(isEven(4))
              cout << "even"  << endl;

       if(!isEven(5))
              cout << "odd"  << endl;

       if(!isOdd(4))
```

```
                    cout << "even"  << endl;

      if(isOdd(5))
                    cout << "odd"  << endl;

  return 0;
}
```

Output

```
even
odd
even
odd
```