Examples for Recursive Functions, Function Overloading and Function Templates

Recursive functions:

1. **Printing a Sequence of Numbers in Reverse**

```
void print(int n) {
    if ( n < 0 ) return; //Terminating condition

     cout << n << " "; //Prints number n

     print(n-1); //Calls itself with (n-1)

     return; //Returns from the function
}
```

This will print a sequence of numbers in reverse, for example, print(3) will print out 3 2 1 0

```
void printrev(int n) {
    if ( n <= 0 ) return; //Terminating condition

    //Previous location of our print statement

    printrev(n-1); //Calls itself with (n-1)

    cout << n << " "; //Prints number n

    return;
}
```

A change of the order of the statements will make difference. For example, printrev(3) will print out 0 1 2 3

2. **Creating a factorial function**

```
int factorial(int n) {

     //Return 1 when n is 0
     if ( n == 0 ) return 1;

     //factorial(n) = n * factorial(n-1);
     return n * factorial(n-1);
}
```

3. **Creating a sum function**

```
int sum(int n) {

    //Return 0 when n is 0
    if ( n <= 0 ) return 0;
    else
        return n + sum(n-1);

}
```

4. **Creating a power function using Recursion**

```
int power(int base, int exp) {

    if ( exp == 0 ) {
        return 1;
    }

    //Initial value for the result is the base
    int result = base;

    //Multiplies the base by itself exp number of times
    for ( int i = 1; i < exp; i++ ) {
        result = result * base;
    }

    return result;
}
```

5. **Creating a function to determine the Fibonacci Sequence**

Fibonacci Formula

Fibonacci(0) = 0;

Fibonacci(1) = 1;

Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2);

```
int fibonacci(int n) {

    // Base cases
    if ( n == 0 ) return 0;
    else if ( n == 1 ) return 1;

    //This is returning fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)
    else
        return fibonacci(n-1) + fibonacci(n-2);

}
```

Function Overloading:

Function overloading by having different types of argument

```
// overloaded functions
#include <iostream>
using namespace std;

int sum (int a, int b)
{
  return a+b;
}

double sum (double a, double b)
{
  return a+b;
}

int main ()
{
  cout << sum (10,20) << '\n';
  cout << sum (1.0,1.5) << '\n'
  return 0;
}
```

Function overloading by having different number of argument

```
// overloaded functions
#include <iostream>
using namespace std;

int sum (int a, int b)
{
  return a+b;
}

int sum (int a, int  b, int c)
{
  return a+b+c;
}

int main ()
{
  cout << sum (10,20) << '\n';
  cout << sum (10,20,20) << '\n'
  return 0;
}
```

Function Templates:

```
template <typename T>
T sum (T a, T b)
{
  T result;
  result = a + b;
  return result;
}

int main () {
  int i=5, j=6, k;
  double f=2.0, g=0.5, h;
  k=sum(i,j);
  h=sum(f,g);
  cout << k << '\n';
  cout << h << '\n';
  return 0;
}
```

Comparison between Function Overloading and Function templates

```
#include <iostream>
using namespace std;

int square (int x)
{
  return x * x;
};

float square (float x)
{
  return x * x;
};

int main()
{
   int    i, ii;
   float  x, xx;

   i = 2;
   x = 2.2;

   ii = square(i);
   cout << i << ": " << ii << endl;

   xx = square(x);
   cout << x << ": " << xx << endl;

   return 0;
}
```

```
#include <iostream>
using namespace std;
template <typename T>

inline T square(T x)
{
   T result;
   result = x * x;
   return result;
};
int main()
{
   int    i, ii;
   float  x, xx;

   i = 2;
   x = 2.2;

   ii = square<int>(i);
   cout << i << ": " << ii << endl;

   // Explicit use of template
   xx = square<float>(x);
   cout << x << ": " << xx << endl;

   // Implicit use of template
   xx = square(x);
   cout << x << ": " << xx << endl;
}
```