Some examples of recursive functions

Problem 1. Write a function that takes a positive integer \( n \) as the input parameter and creates a pattern of the following type that has \( 2n \) rows. (For the example pattern, \( n = 4 \).)

\[
****
***
**
*
* 
** 
*** 
****
\]

Method of attack:

- First identify the base case that arises when \( n = 1 \), in which case exactly two rows are printed, each consisting of a single asterisk.

- Next identify the general case when \( n \geq 2 \). It is not difficult to see that the following scheme works:
  
  (i) Print a row of \( n \) asterisks ending with a newline character
  (ii) Invoke the function recursively with an argument of \( n - 1 \), and
  (iii) Print a row of \( n \) asterisks ending with a newline character.

A recursive function follows.

```c
void pattern(int n) // n >= 1
{
    if(n == 1) // base case
    {
        cout << '*' << endl;
        cout << '*' << endl;
    }
    else
    {
        for(int i = 1; i <= n; i++) cout << '*'; cout << endl;
        pattern(n-1); // recursive invocation
        for(int i = 1; i <= n; i++) cout << '*'; cout << endl;
    }
} // pattern
```
The foregoing function may be slightly shortened as follows:

```cpp
void pattern(int n) // n >= 1
{
    if(n >= 1)
    {
        for(int i = 1; i <= n; i++) cout << '*'; cout << endl;
        pattern(n-1); // recursive invocation
        for(int i = 1; i <= n; i++) cout << '*'; cout << endl;
    }
} // pattern
```

**Problem 2.** Write a function that takes a positive integer n as the input parameter and creates a pattern of the following type that has 2n rows. (For the example pattern, n = 3.)

*  
**  
***  
***  
**  
*  

The present problem requires an approach that is slightly different from the one in the preceding problem. This is because the terminating condition corresponds to the case when two rows of n asterisks are to be printed right at the middle of the pattern. Here is a function.

```cpp
void pattern(int i, int n)
// The calling program invokes this function as follows: pattern(1, n); // n >= 1
{ int j;
  if(i == n)
  {
      for(j = 1; j <= n; j++) cout << '*'; cout << endl;
      for(j = 1; j <= n; j++) cout << '*'; cout << endl;
  }
  else
  {
      for(j = 1; j <= i; j++) cout << '*'; cout << endl;
      pattern(i+1, n); // recursive invocation
      for(j = 1; j <= i; j++) cout << '*'; cout << endl;
  }
} // pattern
```

The foregoing function may be slightly shortened as follows.

```cpp
void pattern(int i, int n)
{
    if(i <= n)
    {
        int j;
        for(j = 1; j <= i; j++) cout << '*'; cout << endl;
        pattern(i+1, n); // recursive invocation
        for(j = 1; j <= i; j++) cout << '*'; cout << endl;
    }
} // pattern
```
**Problem 3.** Examine the following pattern of asterisks, and write a recursive function that generates a pattern of this form.

```
*  
* *  
*   
* * * *  
*   
* *  
*   
* * * * * * * *  
*   
* *  
*   
* * * *  
*   
*   
```

**Method of attack:** The first important step is to recognize the progressive growth of the pattern with the stepwise increase in the intrinsic parameter, say $k$. Here are the first four cases:

<table>
<thead>
<tr>
<th></th>
<th>$k = 1$</th>
<th>$k = 2$</th>
<th>$k = 3$</th>
<th>$k = 4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>
Let \( n = 2^k \), and examine the following characteristics of the pattern:

1. The number of rows is equal to \( n - 1 \).
2. The base case occurs when \( n \) is equal to 2.
3. For \( n \geq 4 \), there are \( \frac{n}{2} \) asterisks in the middle row, i.e., the \( \frac{n}{2} \)th row from the top.
4. For \( n \geq 4 \), the pattern above the middle row is identical to the pattern below the middle row, except that the one below is shifted to the right by \( \frac{n}{4} \) blanks. Further, each of the two (sub)patterns is identical to the pattern at the previous (i.e., \( \frac{n}{2} \)) stage. This is a clear signal that the problem is amenable to a recursive solution.

Here is a complete program.

```cpp
// Written by: Pranava K. Jha
// St. Cloud State University
// Date: Sep. 2, 2008

// This program employs a recursive approach to create a nice pattern
// of asterisks.

#include <iostream>
using namespace std;

void pattern(int n, int shift);
// This function prints the pattern having n rows, where the pattern
// itself is shifted to the right by an amount indicated by the
// parameter shift. n is a power of two.

void line(int m, int shift);
// This function prints a line consisting of m asterisks, where the
// line itself is shifted to the right by an amount indicated by the
// parameter shift. m is a power of two.

int main()
{
    int n; // power of 2
    cout << "Please enter an integer that is a power of two: ";
    cin >> n;
    pattern(n, 0); // call the function with shift equal to zero.
    return 0;
} // end of main
```
void pattern(int n, int shift)
{
    int i;
    if(n == 2)
    {
        // print a single asterisk shifted appropriately to the right
        for(i = 1; i <= shift; i++) // printing enough blanks
            cout << " ";
        cout << "*" << endl; // printing a single asterisk
    }
    else
    {
        pattern(n/2, shift); // recursive call
        line(n/2, shift);    // middle row of the pattern
        pattern(n/2, shift + n/4); // second recursive call
    } // end of if - else
} // end of pattern

void line(int m, int shift)
{
    int i;
    for(i = 1; i <= shift; i++) // printing enough blanks
        cout << " ";
    for(i = 1; i <= m; i++) // printing m asterisks
        cout << "*";
    cout << endl;
} // end of line