جامعة بيرزيت
BIRZEIT UNIVERSITY

Recursion

Abdallah Karakra

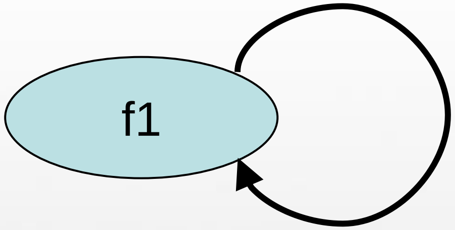
Computer Science Department

Comp 230

Sunday, January 10, 2016

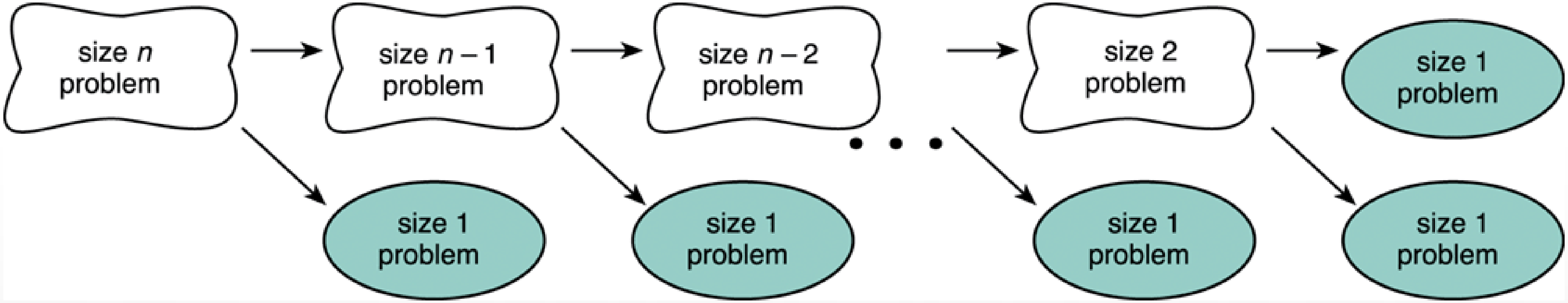
Introduction to Recursion

- A recursive function is one that calls itself.



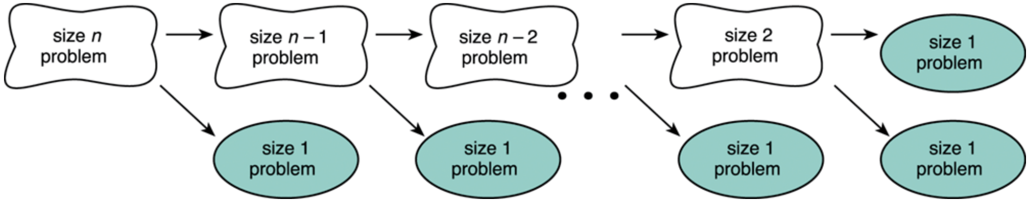
```
void message()
{
          printf("This is a recursive function.\n“);
          message();
}
```

Splitting a Problem into Smaller Problems



- Assume that the problem of size 1 can be solved easily (i.e., the simple case).
- We can recursively split the problem into a problem of size 1 and another problem of size n-1.

Splitting a Problem into Smaller



Let f(x)=f(x -1)+3 , f(0)=4 , find f(7)

f(7) = f(7-1)+3 → f(7)=f(6)+3

f(6) = f(6-1)+3 → f(6)=f(5)+3

f(5) = f(5-1)+3 → f(5)=f(4)+3

f(4) = f(4-1)+3 → f(4)=f(3)+3

f(3) = f(3-1)+3 → f(3)=f(2)+3

f(2) = f(2-1)+3 → f(2)=f(1)+3

f(1) = f(1-1)+3 → f(1)=f(0)+3

f(7)=22+2=25

f(6)=19+3=22

f(5)=16+3=19

f(4)=13+3=16

f(3)=10+3=13

f(2)=7+3=10

f(1)=4+3=7

f(0)=4

Base case

Recursive Problem

The function below displays the string "This is a recursive function.\n", and then calls itself.

```
void message()
{
        printf("This is a recursive function.\n“);
        message();
}
```

Recursive Problem

- The function is like an **infinite loop** because there is **no code to stop it from repeating.**
- Like a loop, a recursive function **must have some algorithm to control the number of times it repeats.**

Recursion

- Like a loop, a recursive function must have some algorithm to control the number of times it repeats. Shown below is a modification of the `message` function. It passes an integer argument, which holds the number of times the function is to call itself.

```
void message(int times)
{
        if (times > 0)
        {
                printf("This is a recursive function.\n“);
                message(times - 1);
        }

}
```

Recursion

- The function contains an `if/else` statement that controls the repetition.
- As long as the `times` argument is greater than zero, it will display the message and call itself again. Each time it calls itself, it passes `times - 1` as the argument.

Recursive Function

Let f(x)=f(x -1)+3 , f(0)=4 , find f(7)

```
int f(int x)
{

  if (x == 0)
      return 4; //base case

  else

      return f(x-1)+3;


}
```

Recursive function terminates when a base case is met.

Trace of f(x)=f(x -1)+3

$f(\cancel{7})$ 25

↳ $f(\cancel{6})+3$ 22

↳ $f(\cancel{5})+3$ 19

↳ $f(\cancel{4})+3$ 16

↳ $f(\cancel{3})+3$ 13

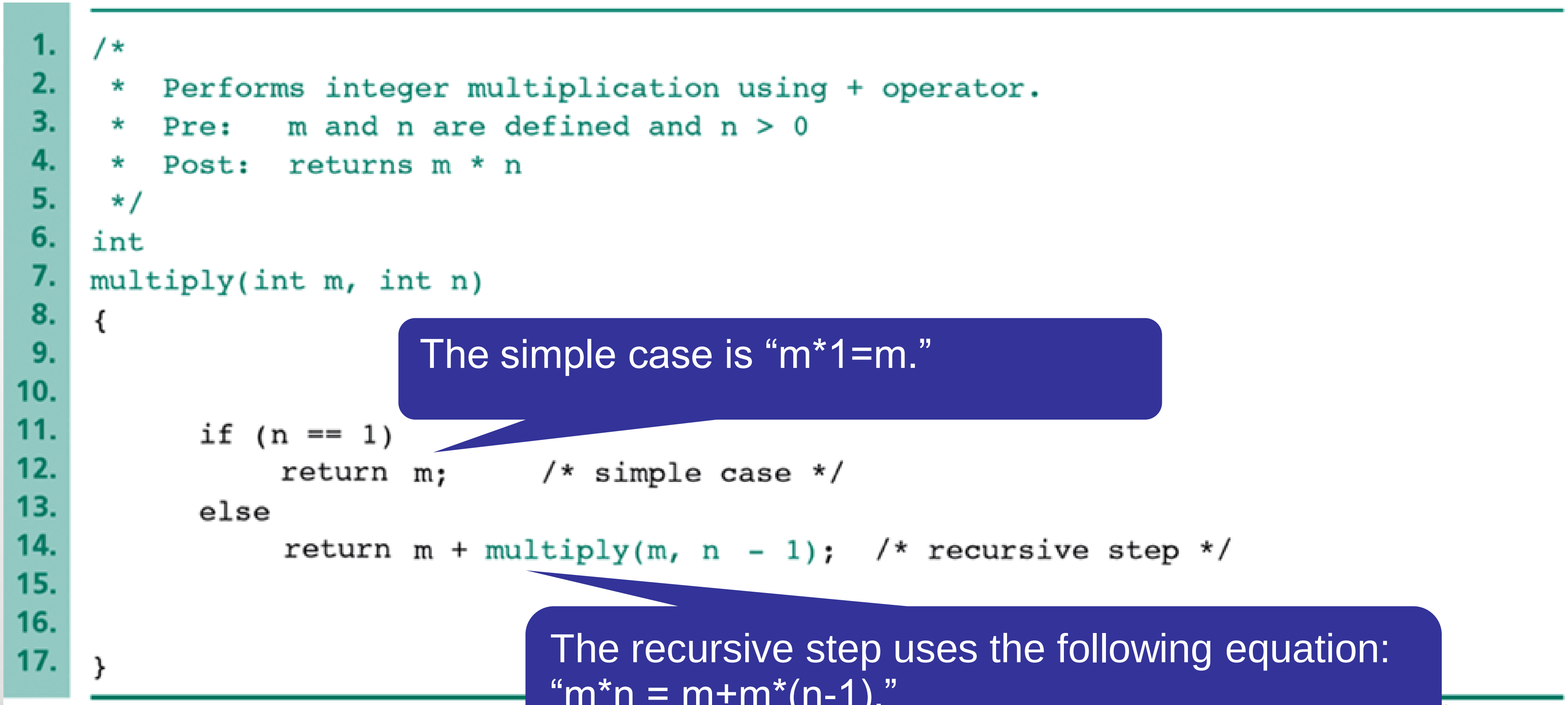
↳ $f(\cancel{2})+3$ 10

↳ $f(\cancel{1})+3$ 7

↳ $f(\cancel{0})+3$ 4

Recursive Function multiply

We can implement the multiplication by addition.



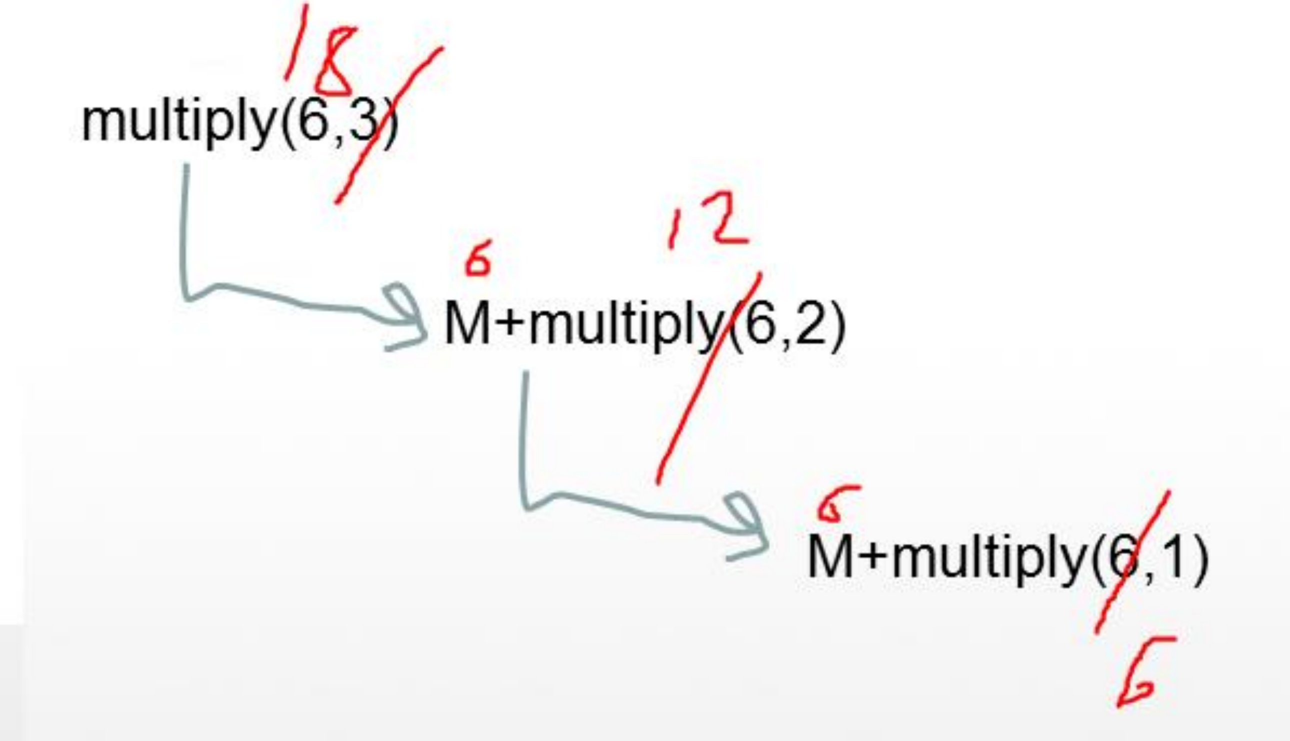
```
/*
 *  Performs integer multiplication using + operator.
 *  Pre:   m and n are defined and n > 0
 *  Post:  returns m * n
 */
int
multiply(int m, int n)
{


      if (n == 1)
          return m;     /* simple case */
      else
          return m + multiply(m, n - 1);  /* recursive step */


}
```

The simple case is “m*1=m.”

The recursive step uses the following equation: “m*n = m+m*(n-1).”

Trace of Function multiply(6,3)

.



multiply(6,3)

M+multiply(6,2)

M+multiply(6,1)

Recursive Function Factorial

In mathematics, the notation n! represents the factorial of the number n. The factorial of a number is defined as:

```
n! =   1 * 2 * 3 * ... * n        if n > 0
       1                           if n = 0
```

Recursive Function Factorial

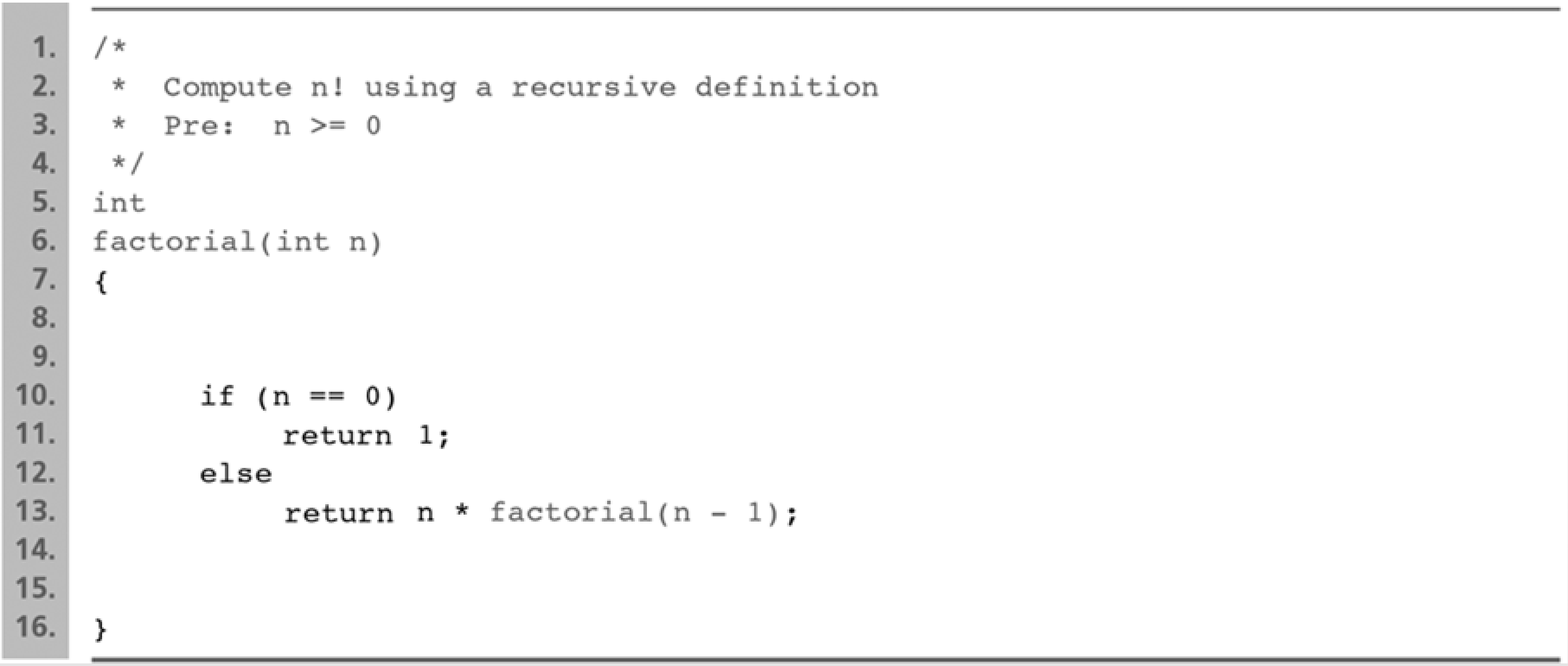
Another way of defining the factorial of a number, using recursion, is:

```
Factorial(n) =      n * Factorial(n - 1)      if n > 0
                                       1      if n = 0
```

The following C function implements the recursive definition shown above:

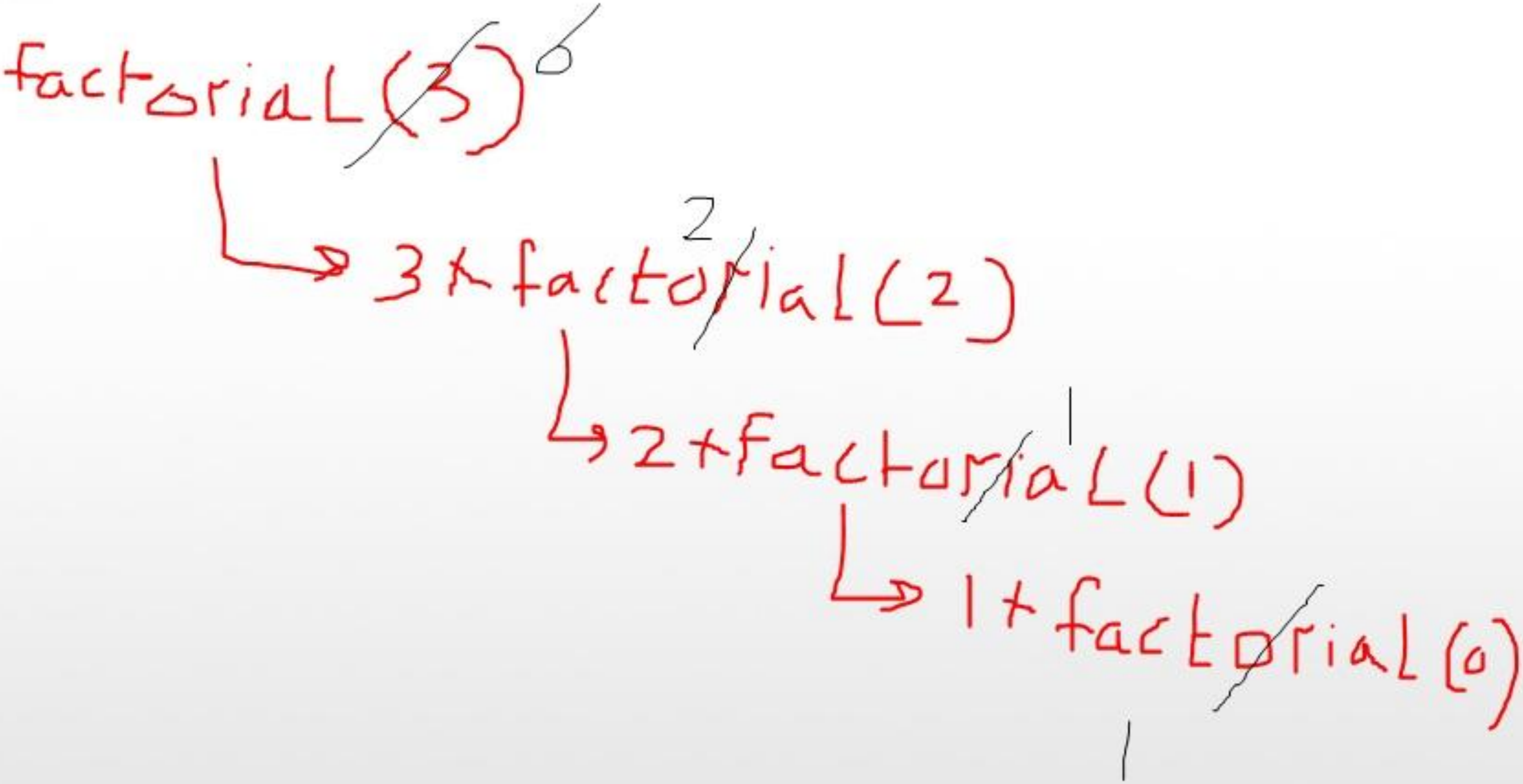
```
 int factorial(int num)
{
        if (num == 0)

                return 1;

        else
                return num * factorial(num - 1);



}
```

Recursive Function Factorial



```
/*
 *  Compute n! using a recursive definition
 *  Pre:  n >= 0
 */
int
factorial(int n)
{


      if (n == 0)
          return 1;
      else
          return n * factorial(n - 1);


}
```

Trace of fact = factorial(3);



Tracing recursive methods

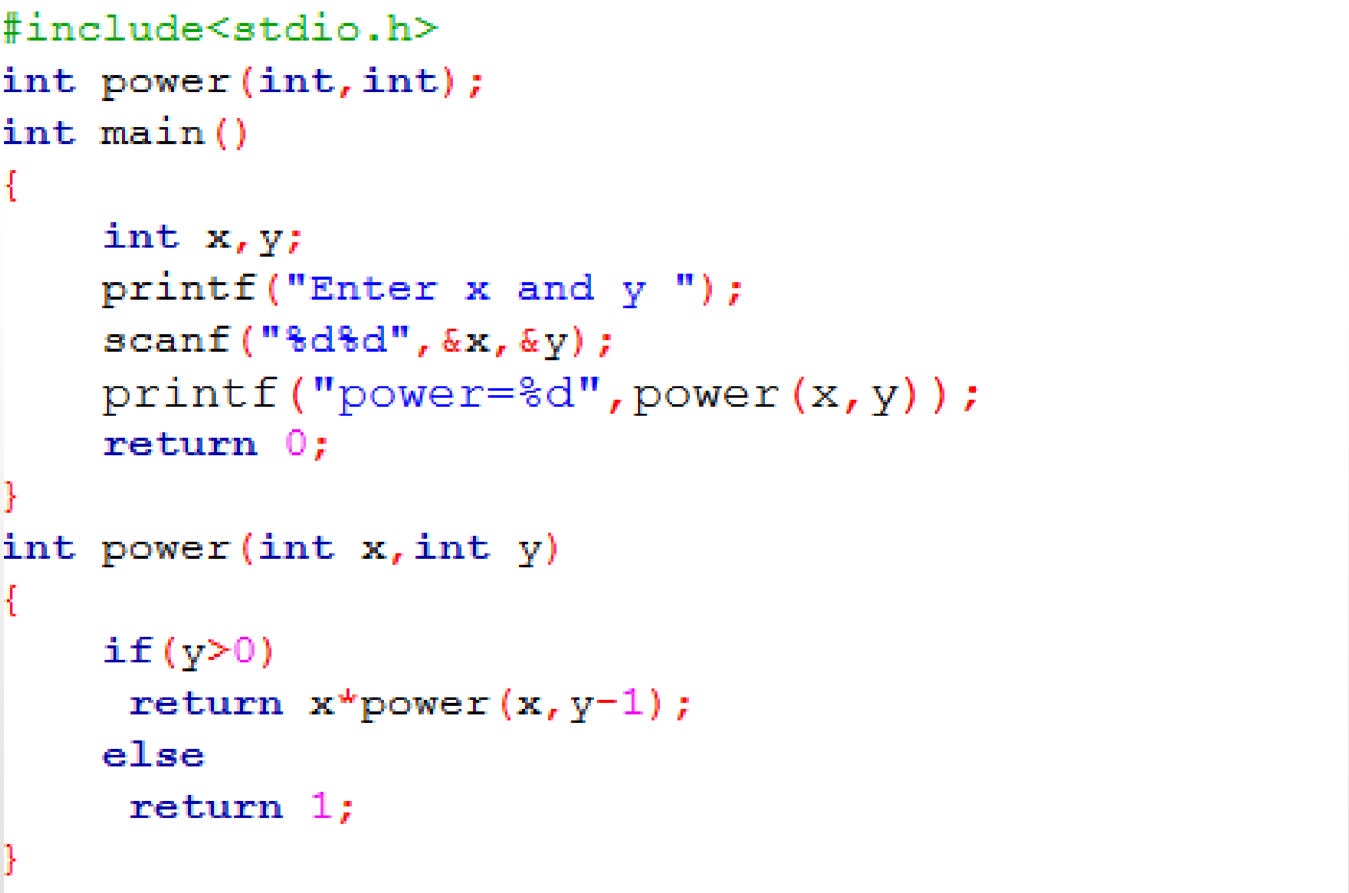
Consider the following method:

```
int mystery(int x, int y) {
if (x < y)
    return x;
  else
    return mystery(x - y, y);
}
```

For each call below, indicate what value is returned:

```
mystery(6, 13)                6
mystery(14, 10)               4
mystery(37, 10)               7
mystery(8, 2)                 0
mystery(50, 7)                1
```

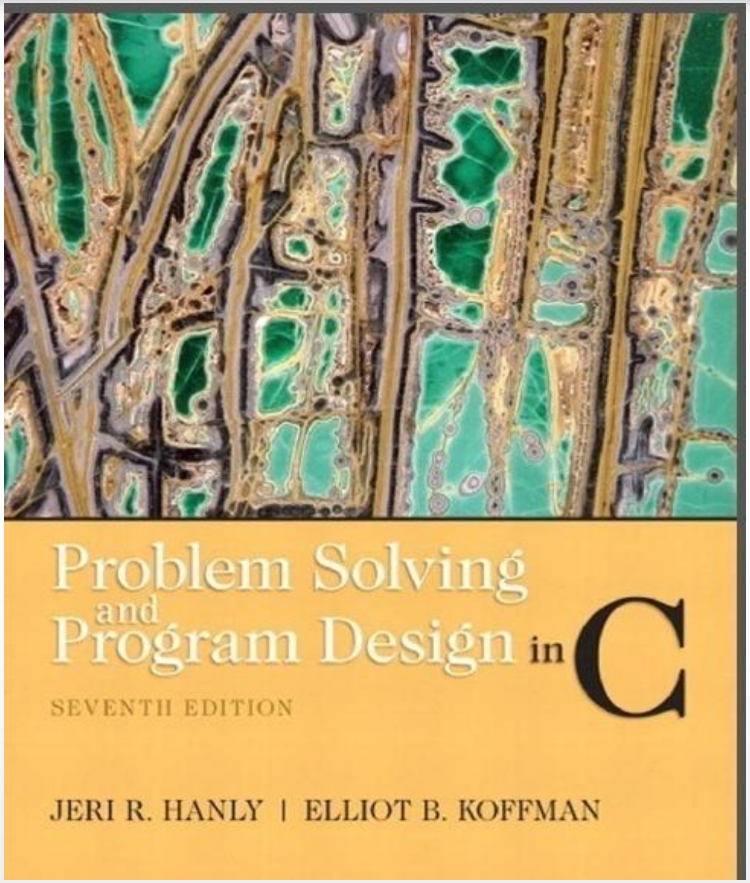
Recursive Function Power



```c
#include<stdio.h>
int power(int,int);
int main()
{
    int x,y;
    printf("Enter x and y ");
    scanf("%d%d",&x,&y);
    printf("power=%d",power(x,y));
    return 0;
}
int power(int x,int y)
{
    if(y>0)
     return x*power(x,y-1);
    else
     return 1;
}
```

Question?

“Success is the sum of small efforts, repeated day in and day out.”
Robert Collier



References:

Problem Solving & Program Design in C (main reference)