**CS 210** **November 22, 2016**
**Day 21**

Go over exam 3

Chapter 9 Recursion

# Recursion

- A recursive function is one that calls itself or that is part of a cycle in the sequence of function calls.

- The ability to invoke itself enables a recursive function to be repeated with different parameter values.
- It can be used an an alternative to looping.

- Recursion is typically used to specify a natural, simple solution that would otherwise be very difficult to solve.

# The Nature of Recursion

- One or more simple cases of the problem have a straightforward, nonrecursive solution.

- The other cases can be redefined in terms of problems that are closer to the simple cases.

# The Nature of Recursion

- By applying this redefinition process every time the recursive function is called, eventually the problem is reduced entirely to simple cases, which are relatively easy to solve.

*if this is a simple case*
    *solve it*
*else*
    *redefine the problem using recursion*

Simple Example – Multiplication by repeated addition

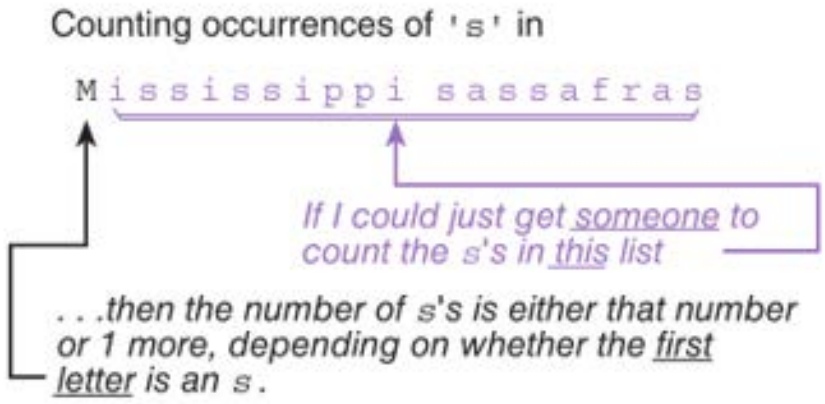
**m * n = m + m *(n – 1)**

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

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

    return (ans);
}
```

# Example

- Develop a function to count the number of times a particular character appears in a string.

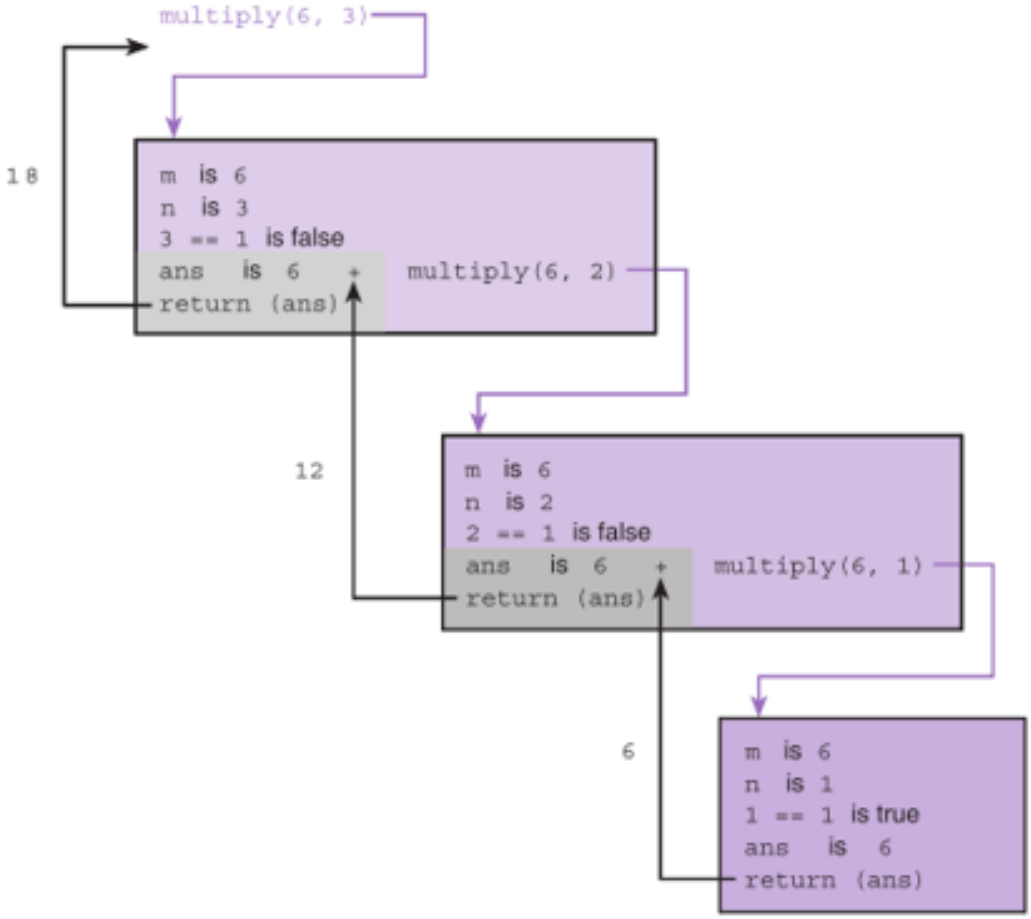
count('s', "Mississippi sassafrs");




```
int
count(char ch, const char *str)
{
      int ans;

      if (str[0] == '\0')                        /* simple case */
          ans = 0;
      else                  /* redefine problem using recursion */
         if (ch == str[0]) /* first character must be counted */
             ans = 1 + count(ch, &str[1]);
         else                /* first character is not counted */
             ans = count(ch, &str[1]);

      return (ans);
}
```

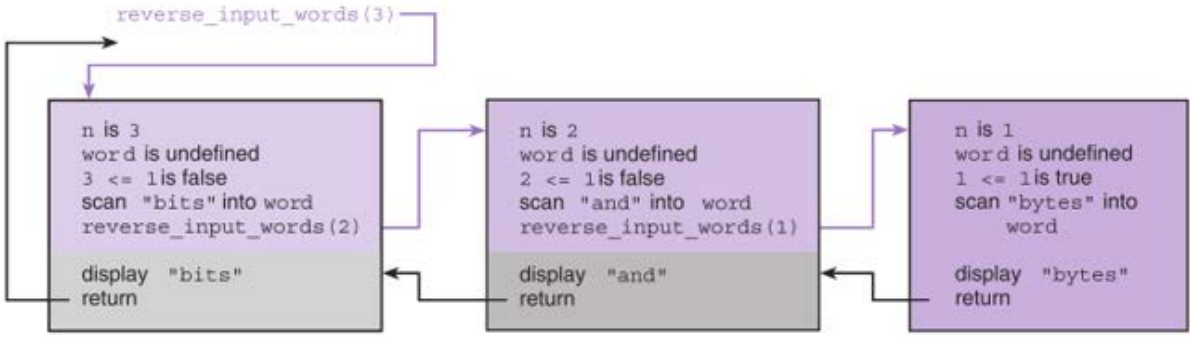
# Tracing Recursive Functions

- activation frame
  - representation of one call to a function
- terminating condition
  - a condition that is true when a recursive algorithm is processing a simple case
- system stack
  - area of memory where parameters and local variables are allocated when a function is called and deallocated when the function returns



**Single step through this program with "bits and bytes" as input**

```
#define WORDSIZE 10
#include<stdio.h>
void ReverseWords(int n);
int main()
{
   ReverseWords(3);
}

void ReverseWords(int n)
{
   char word[WORDSIZE];
   if(n <= 1)
   {
      scanf_s("%s", word, WORDSIZE);
      printf("%s\n", word);
   }
   else
   {
      scanf_s("%s", word, WORDSIZE);
      ReverseWords(n-1);
      printf("%s\n", word);
   }
}
```



Call `reverse_input_words` with n equal to `3`.
    Scan the first word (`"bits"`) into `word`.
    Call `reverse_input_words` with n equal to `2`.
        Scan the second word (`"and"`) into `word`.
        Call `reverse_input_words` with n equal to `1`.
            Scan the third word (`"bytes"`) into `word`.
            Display the third word (`"bytes"`).
            Return from third call.
        Display the second word (`"and"`).
        Return from second call.
    Display the first word (`"bits"`).
    Return from original call.

# Mathematical Functions

- recursive factorial
- iterative factorial
- recursive fibonacci
- recursive gcd

```c
#include<stdio.h>
int Factorial(int n);
int Fibonacci(int n);
int GCD(int m, int n);
int main()
{
   int n, m;
   printf("Enter an integer...");
   scanf_s("%d", &n);
   printf("n! = %d\n", Factorial(n));
   printf("nth Fibonacci number is %d\n", Fibonacci(n));
   printf("Enter two integers... ");
   scanf_s("%d %d", &m, &n);
   printf("GCD of %d and %d is %d\n", m, n, GCD(m, n));
}
int Factorial(int n)
{
    if(n == 1)
       return 1;
    else
       return n*Factorial(n-1);
}
int Fibonacci(int n)
{
    if(n == 1 || n == 2)
       return 1;
    return (n + Fibonacci(n-1));
}
/*
Greatest Common Divisor of m and n is the largest int
  which divides them both evenly.
if m % n is zero the answer is n
otherwise the answers is GCD of (n, and m % n)
*/
int GCD(int m,int n)
{
    if(m % n == 0)
       return n;
    return(GCD(n, m%n));
}
```

**CS 210** **November 22, 2016**

**Recursive Functions**

Write two recursive functions. The first is called *SumOfDigits*. It accepts a single integer and returns the sum of the digits in the integer. For example if the argment is 2345 *SumOfDigits* will return 2+3+4+5 = 14. The second function is called *PowerOfTwo*. This function accepts an integer argument *n* and returns an integer equal to $2^n$