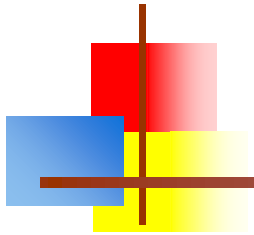
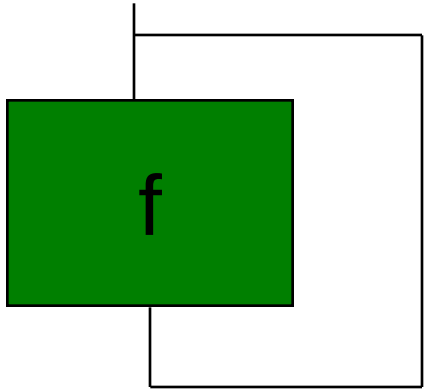
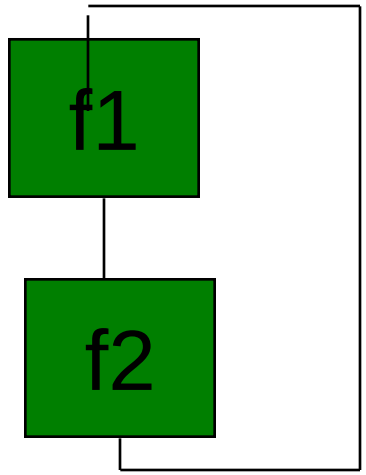


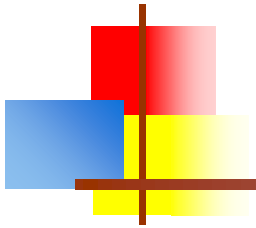
RECURSION

What is recursion?

- a function calls itself
  - direct recursion







Recursion

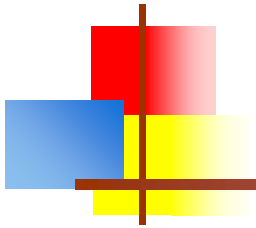
- is an alternative to iteration (loop)
  - often more "elegant" and concise than a loop
  - sometimes very inefficient
- recursion should be considered when
  - a problem can be defined in terms of successive smaller problems of the same type, i.e. recursively
  - eventually the problem gets small enough that the answer is known (base case)
  - reducing the problem size leads to the base case
- sometimes the "work" of the algorithm is done after the base case is reached

Outline of a Recursive Function

```
if (answer is known)
    provide the answer
else
    make a recursive call
to
    solve a smaller version
    of the same problem
```

base case

recursive case - “leap of faith”



Two well-defined properties of a recursive procedure are:

1. There must be certain base criteria for which the procedure does not call itself.
2. Each time the procedure does call itself, it must be closer to the base criteria.

- Example:

Factorial Function

(a) If n = 0, then n! = 1

(b) If n > 0, then n! = n.(n-1)!

Calculate 4!

1. 4! = 4 . 3!
2. 3! = 3 . 2!
3. 2! = 2 . 1!
4. 1! = 1. 0!
5. 0! = 1
6. 1! = 1 . 1 =1
7. 2! = 2 . 1 = 2
8. 3! = 3 . 2 = 6
9. 4! = 4 . 6 = 24

Result: 4! = 24

Fibonacci Sequence [0 1 1 2 3 5 8 13 21 34 55 89……]

Fibonacci(N)

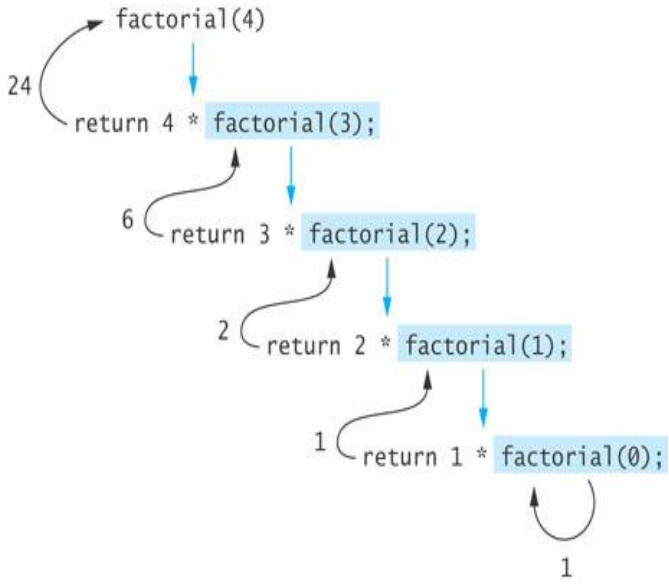
1. If N = 0 or N = 1 then Set Fib := N and Return.
2. Set Fib := Fibonacci(N-2) + Fibonacci(N-1)
3. Return

Example: Fibonacci(6)

1. Fib(6) := Fib(4) + Fib(5)
2. Fib(6) := Fib(2) + Fib(3) + Fib(5)
3. Fib(6) := Fib(0) + Fib(1) + Fib(3) + Fib(5)
4. Fib(6) := 0 + 1 + Fib(3) + Fib(5)
5. Fib(6) := 1 + Fib(1) + Fib(2) + Fib(5)
6. Fib(6) := 1 + 1 + 1 + Fib(3) + Fib(4)
7. Fib(6) := 3 + 2 + Fib(4)
8. Fib(6) := 5 + Fib(2) + Fib(3)
9. Fib(6) := 5 + 1 + 2
10. Fib(6) := 8

Recursive Call Tree

```
int factorial(int n) {
if (n == 0)
return 1;
else
return n * factorial(n – 1);
}
```



Factorial (n) - iterative

Factorial (n) = n * (n-1) * (n-2) * ... * 1

for n > 0

Factorial (0) = 1

```
int IterFact (int n)
{
    int fact =1;
    for (int i = 1; i <= n; i++)
        fact = fact * i;
    return fact;
}
```

Factorial (animation 1)

- x = factorial(3)

3 is put on stack as n

- static int factorial(int n) { //n=3

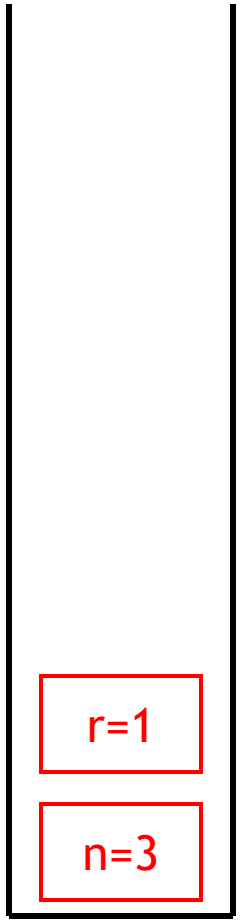
```
    int r = 1;
    if (n <= 1) return r;
    else {
        r = n * factorial(n - 1);
        return r;
    }
}
```

r is put on stack with value 1

All references to r use this r

All references to n use this n

Now we recur with 2...



Factorial (animation 2)

- r = n * factorial(n - 1);

- static int factorial(int n) {//n=2
  int r = 1;
  if (n <= 1) return r;
  else {
  r = n * factorial(n - 1);
  return r;
  }
  }

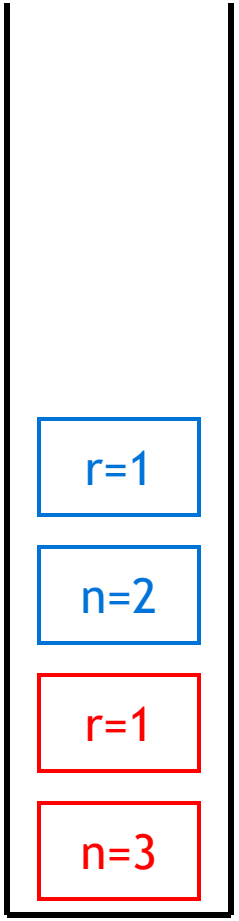
2 is put on stack as n

r is put on stack with value 1

Now using this r

And this n

Now we recur with 1...



Factorial (animation 3)

```
r = n * factorial(n - 1);

static int factorial(int n) {
    int r = 1;
    if (n <= 1) return r;
    else {
        r = n * factorial(n - 1);
        return r;
    }
}
```

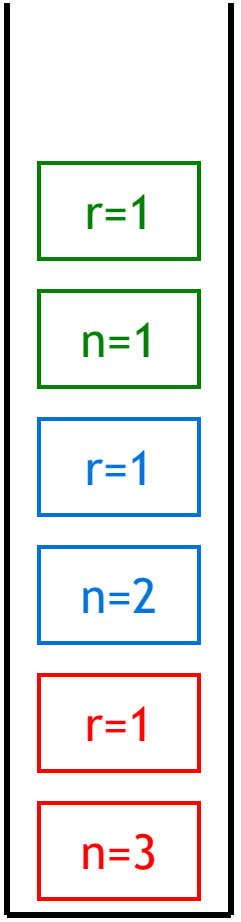
1 is put on stack as n

Now using this r

r is put on stack with value 1

And this n

Now we pop r and n off the stack and return 1 as factorial(1)



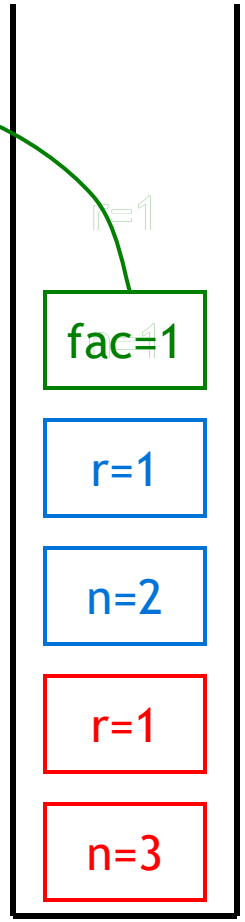
Factorial (animation 4)

- r = n * factorial(n - 1);

- static int factorial(int n) {
  int r = 1;
  if (n <= 1) return r;
  else {
  r = n * factorial(n - 1);
  return r;
  }
  }

Now using this r

And this n



Now we pop r and n off the stack and return 1 as factorial(1)

Factorial (animation 5)

- r = n * factorial(n - 1);

- static int factorial(int n) {
  int r = 1;
  if (n <= 1) return r;
  else {
  r = n * factorial(n - 1);
  return r;
  }
  }

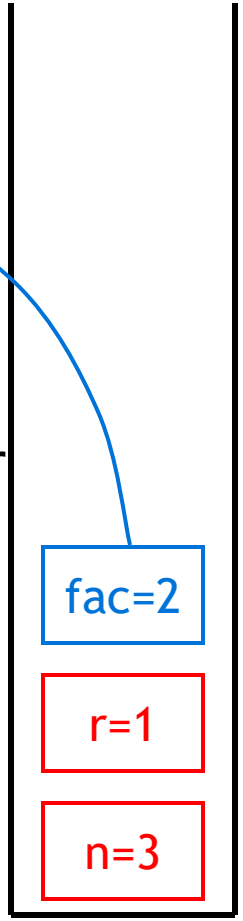
Now using this r

And this n

1

2 * 1 is 2;
Pop r and n;

Return 2



Factorial (animation 6)

- x = factorial(3)

- static int factorial(int n) {
  int r = 1;
  if (n <= 1) return r;
  else {
  r = n * factorial(n - 1);
  return r;
  }
  }

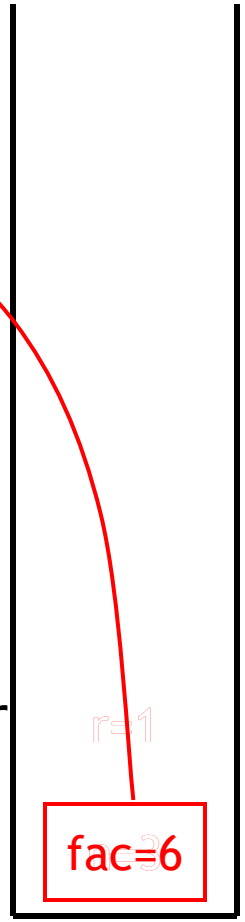
2

3 * 2 is 6;
Pop r and n;

Return 6

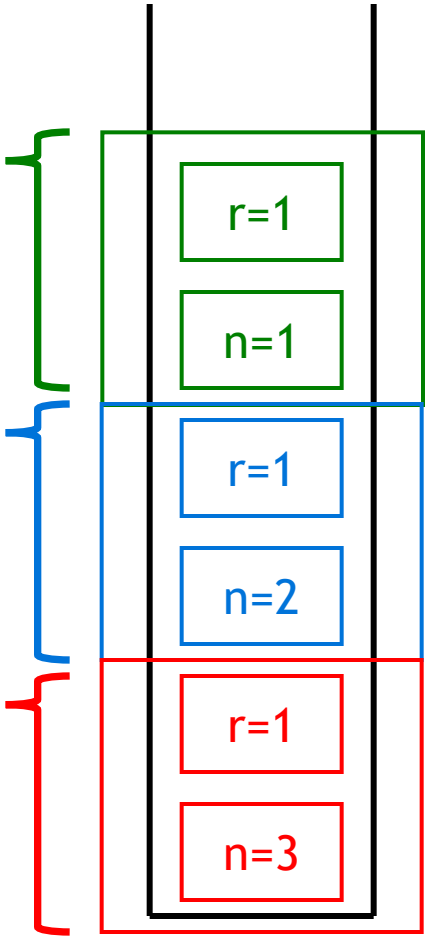
Now using this r

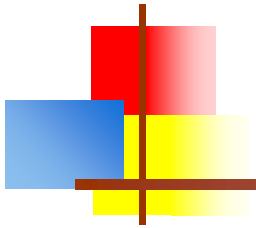
And
this n



Stack frames

- Rather than pop variables off the stack one at a time, they are usually organized into stack frames
- Each frame provides a set of variables and their values
- This allows variables to be popped off all at once
- There are several different ways stack frames can be implemented





Example

```
#include<stdio.h>
int fact(int);
int main(){
  int num,f;
  printf("\nEnter a number: ");
  scanf("%d",&num);
  f=fact(num);
  printf("\nFactorial of %d is: %d",num,f);
  return 0;
}
int fact(int n){
   if(n==1)
       return 1;
   else
       return(n*fact(n-1));
 }
```

Binary search

```
int main(){
    int a[10],i,n,m,c,l,u;

   printf("Enter the size of an array: ");
   scanf("%d",&n);
   printf("Enter the elements of the array: " );
   for(i=0;i<n;i++){
       scanf("%d",&a[i]);
   }
   printf("Enter the number to be search: ");
   scanf("%d",&m);
    l=0,u=n-1;
   c=binary(a,n,m,l,u);
   if(c==0)
       printf("Number is not found.");
   else
       printf("Number is found.");
    return 0;
}
```

```
int binary(int a[],int n,int m,int l,int u){

    int mid,c=0;

    if(l<=u){
        mid=(l+u)/2;
        if(m==a[mid]){
            c=1;
        }
        else if(m<a[mid]){
            return binary(a,n,m,l,mid-1);
        }
        else
            return binary(a,n,m,mid+1,u);
    }
    else
      return c;
}
```

Any Question?