Recursion

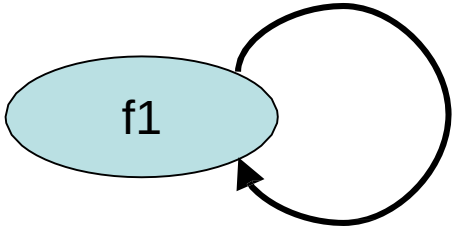
Refat Othman

Computer Science Department

Comp 133

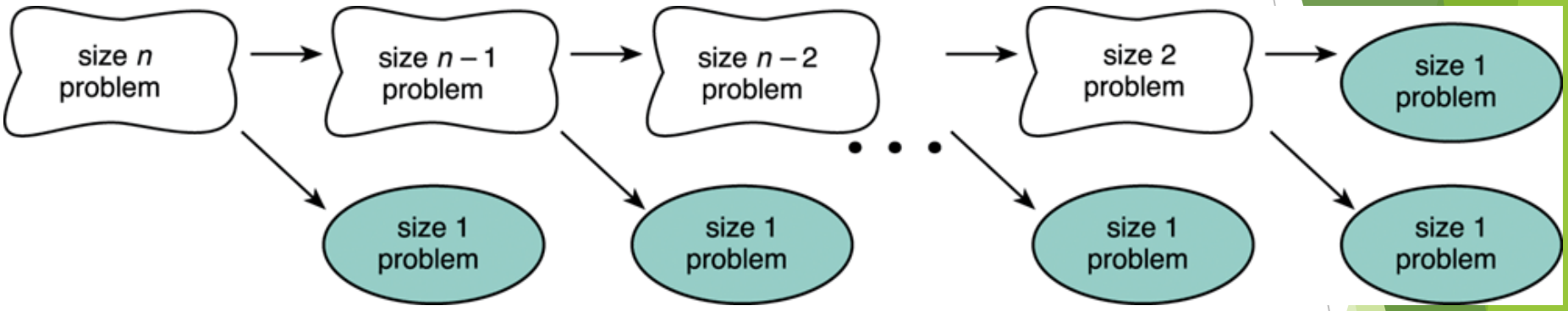
Introduction to Recursion

- A recursive function is one that calls itself.



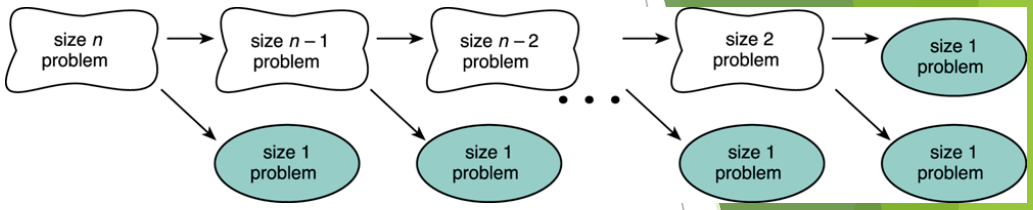

```
void message()
{
        printf("This is a recursive function.\n");
        message();
}
```

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

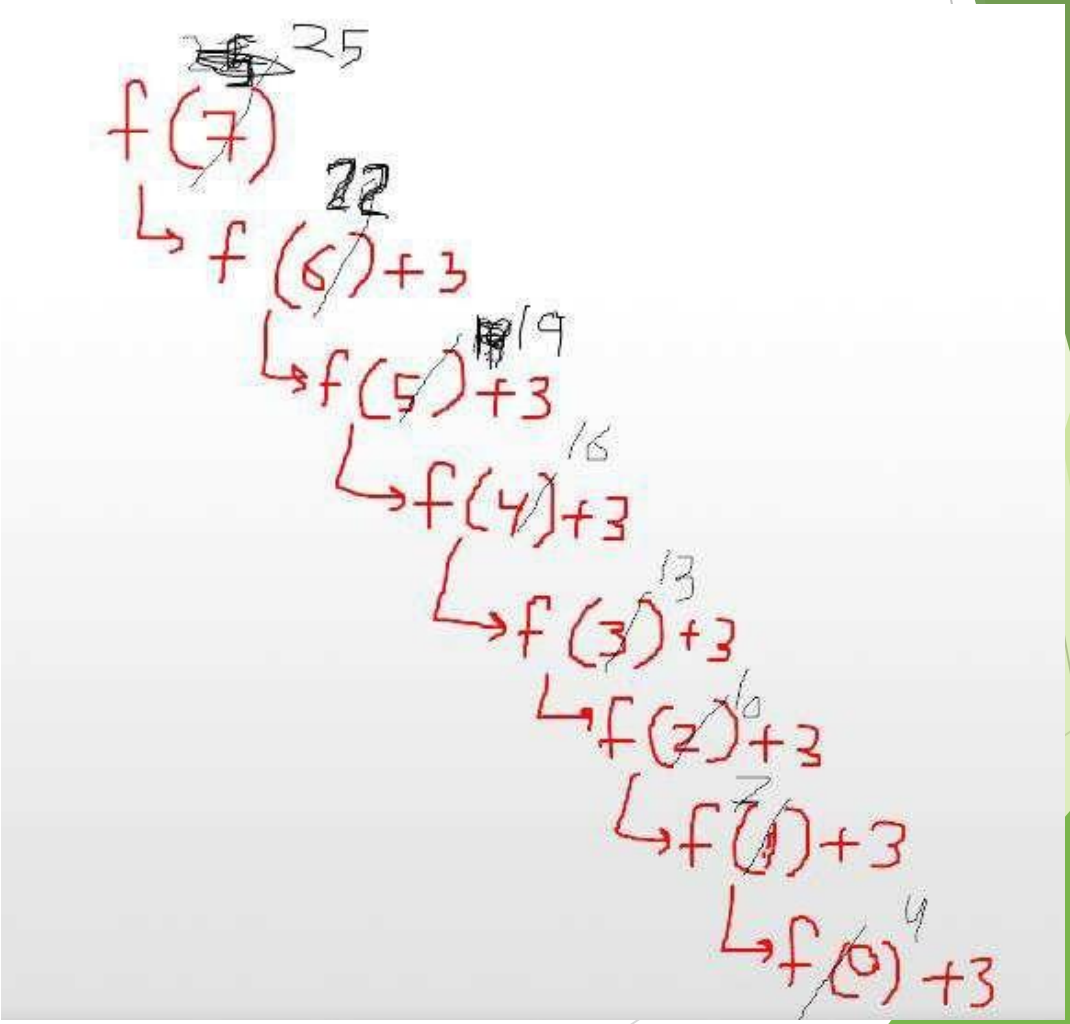
Let f(x)=f(x -1)+3 , f(0)=4 , find f(7)

```
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



Recursive Function multiply

We can implement the multiplication by addition.

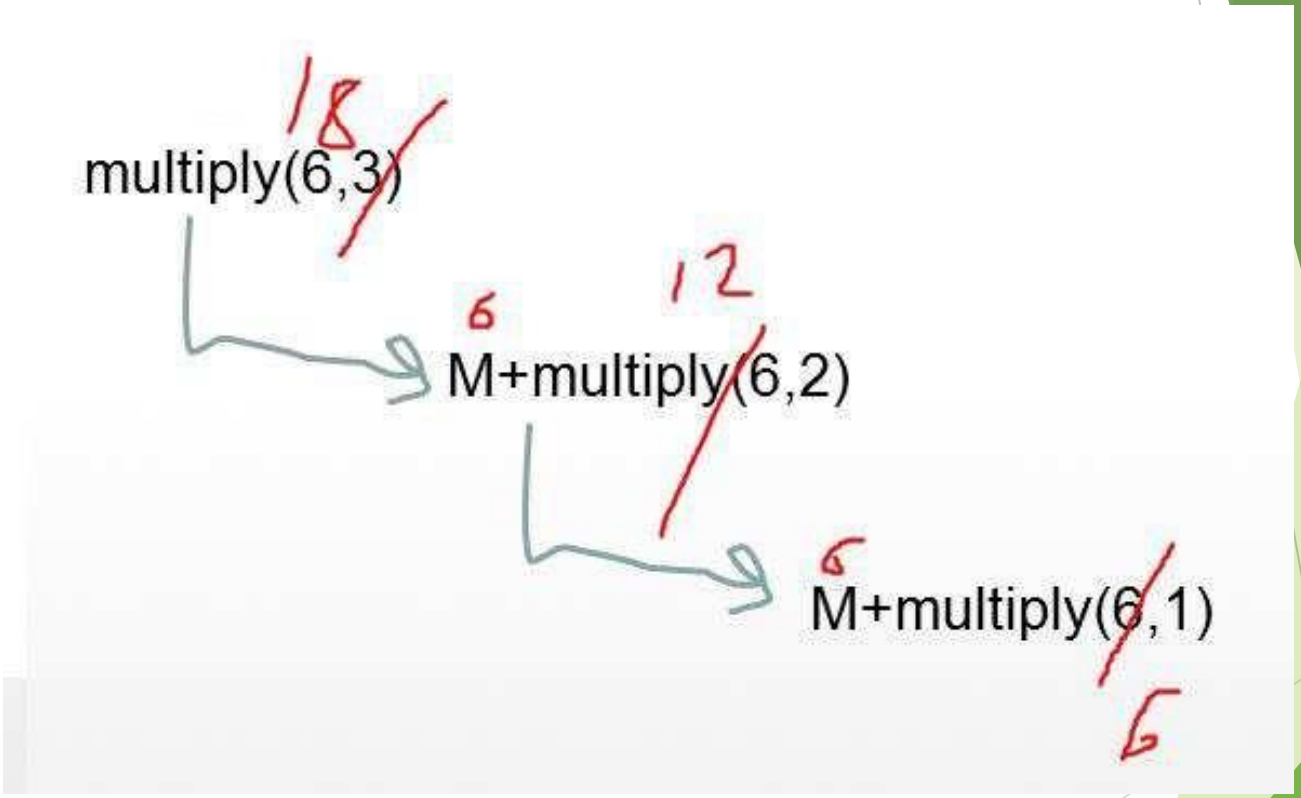
```
1.  /*
2.   *  Performs integer multiplication using + operator.
3.   *  Pre:   m and n are defined and n > 0
4.   *  Post:  returns m * n
5.   */
6.  int
7.  multiply(int m, int n)
8.  {
9.
10.
11.       if (n == 1)
12.            return m;      /* simple case */
13.       else
14.            return m + multiply(m, n - 1);  /* recursive step */
15.
16.
17. }
```

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)

.



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
```


BIRZEIT UNIVERSITY

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:

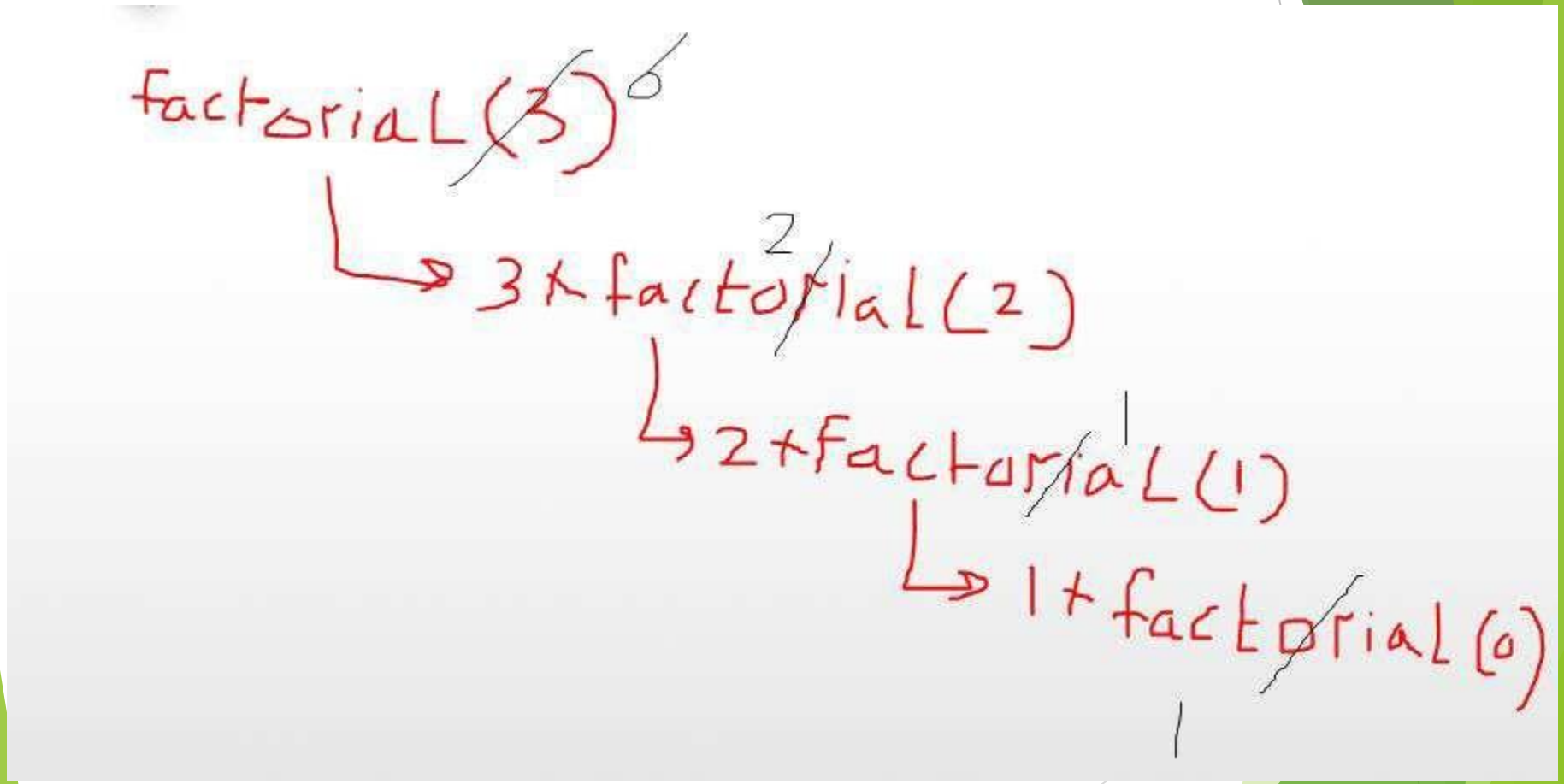
```
int factorial(int num)
{
      if (num == 0)
            return 1;
      else
            return num * factorial(num - 1);


}
```

Recursive Function Factorial

```
1.  /*
2.   *  Compute n! using a recursive definition
3.   *  Pre:  n >= 0
4.   */
5.  int
6.  factorial(int n)
7.  {
8.
9.
10.       if (n == 0)
11.            return 1;
12.       else
13.            return n * factorial(n - 1);
14.
15.
16. }
```

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)
mystery(14, 10)
mystery(37, 10)
mystery(8, 2)
mystery(50, 7)
```

6

4

7

0

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;
}
```

Recursive Function fibonacci

the Fibonacci sequence 1, 1, 2,3, 5, 8, 13, 21, 34,……

a_n: 1, 1, 2, 3, 5, 8, 13, 21, 34,……
n: 1, 2, 3, 4, 5, 6, 7, 8 , 9,……

$a_1 = 1$, $a_2 = 1$, $a_3 = a_1 + a_2 = 2$, $a_n = a_{n-1} + a_{n-2}$

Recursive Function fibonacci

the Fibonacci sequence 1, 1, 2,3, 5, 8, 13, 21, 34,……

```
1.  /*
2.   *  Computes the nth Fibonacci number
3.   *  Pre: n > 0
4.   */
5.  int
6.  fibonacci(int n)
7.  {
8.        int ans;
9.
10.       if (n == 1  ||  n == 2)
11.             ans = 1;
12.       else
13.             ans = fibonacci(n - 2)  +  fibonacci(n - 1);
14.
15.       return (ans);
16. }
```

Recall:

What is the output of the following statements.
Try each statement separately.

```
#include <stdio.h>
int main()
{
    char s[10]="hello";          // s=&s[0]
    char *v;
    v=s;
    printf("%s",v);// hello
    printf("%s",&v[1]);//ello
    printf("%s",&v[2]);//llo
    printf("%c",*v);//h
    printf("%c",*(v+1));//e
    printf("%c",*s);//h
   printf("%c",*(s+1));//e
    return 0;
}
```

Print Hello

Reverse Word

```c
#include <stdio.h>
void reverseWord(char*str)
{
    if (*str!='\0')
    {

      reverseWord(str+1);
      printf("%c",*str);
    }
}
int main()
{
    reverseWord("Hello");
    return 0;
}
```

Print Word

```c
#include <stdio.h>
void PrintWord(char*str)
{
    if (*str!='\0')
    {
      printf("%c",*str);
      PrintWord(str+1);

    }
}
int main()
{
    PrintWord("Hello");
    return 0;
}
```

Tracing recursive methods

Consider the following method:

```
void mystery(int n) {
  if (n <= 1)
  printf("%d",n);
  else  {
  mystery(n/2);
  printf(" , %d ",n);
  }
}
```

For each call below, indicate what value is returned:

```
mystery2(1)
mystery2(2)
mystery2(3)
mystery2(4)
mystery2(16)
mystery2(30)
mystery2(100)
```

~~1~~

~~1 , 2~~

~~1 , 3~~

~~1 , 2 , 4~~

~~1 , 2 , 4 , 8 , 16~~

~~1 , 3 , 7 , 15 , 30~~

~~1 , 3 , 6 , 12 , 25~~ , 50 , 100