“C Programming”

Functions

What is the Value of the a and b after swapping call ?

```
void swapping(int c, int d)  {
        int tmp;
        tmp = c;
        c = d;
        d = tmp;
        printf("In function: %d %d\n", c , d);
}
void main() {
        int a,b;
         a=5; b=10;
         printf("input: %d %d\n", a, b);
        swapping(a,b);
        printf("output: %d %d\n", a, b);
}
```

What is the Value of the a and b after swapping call ?

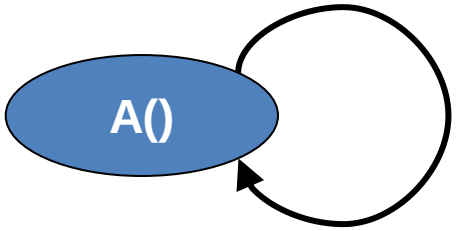
```
void swapping(int *ptr_c, int *ptr_d) {
        int tmp;
        tmp = *ptr_c;
        *ptr_c = *ptr_d;
        *ptr_d = tmp;
        printf("In function: %d %d\n", *ptr_c , *ptr_d);
}
void main( ) {
        int a,b;
        a=5;
        b=10;
        printf("input: %d %d\n", a, b);
        swapping(&a,&b);
        printf("output: %d %d\n", a, b);
}
```

Recursion

- **Definition:** Recursion is a process by which **a function calls itself** repeatedly, until some specified condition has been satisfied.
- To solve a problem recursively
  - The problem must be written in a recursive form.
  - The problem statement must include a stopping condition.
- A recursive function must have the following type of statements :
  - A statement to test and determine whether the function is calling itself again.
  - A statement that calls the function itself and must be argument.
  - A conditional statement (if-else).
  - A return statement.

Recursion

Syntax:



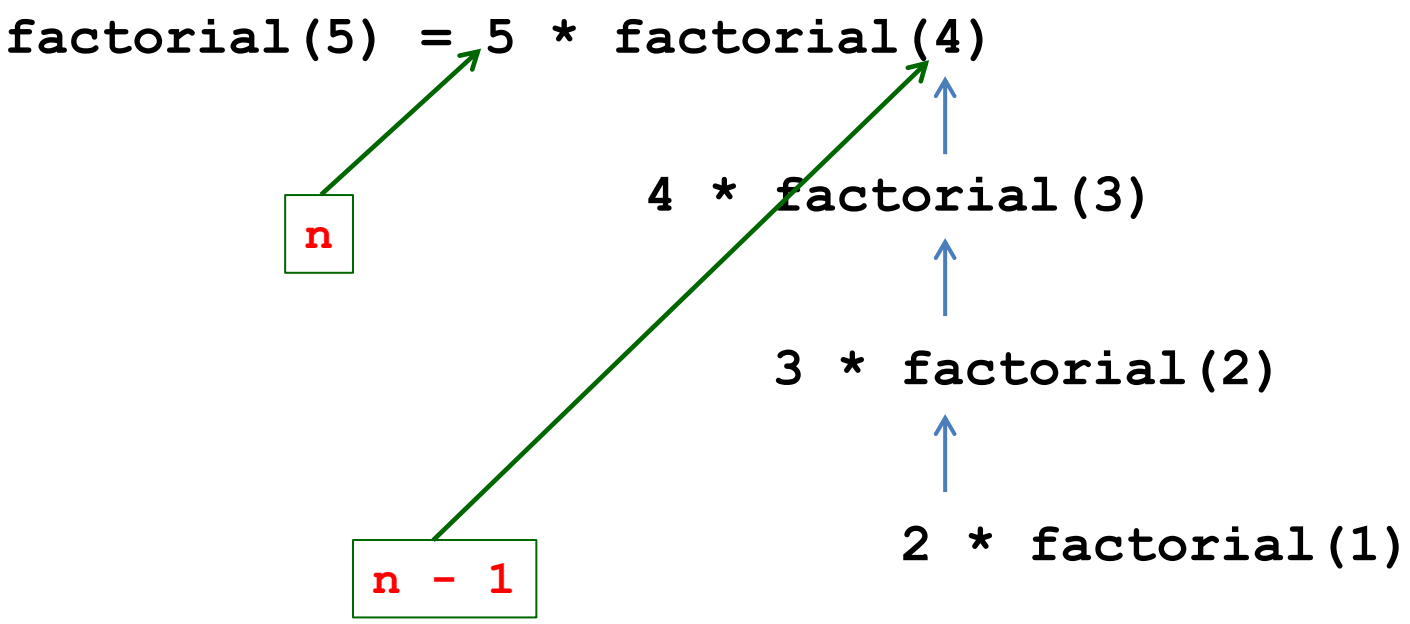

```
void main()
{
    A();
    return;
}
```

```
void A()
{
    if (stopping condition)
        return;
    else
        A();
    return;
}
```

Recursion – Example1 - Factorial

$$\text{fact}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot \text{fact}(n-1) & \text{if } n > 0 \end{cases}$$

Can we compute factorial using recursion?



Recursion – Example1 - Factorial

Example: Recursive function to find factorial

```
#include<stdio.h>
long int fact(int f);
int main()
{
    long int fval;
    int n;
    printf("Enter n value:");
    scanf("%d", &n);
    fval = fact(n);
    printf("Factorial=%ld", fval);
    return 0;
}
```

```
long int fact(int f)
{
   if (f==1 || f==0)
        return 1;
    return (f*fact(f-1));
}
```

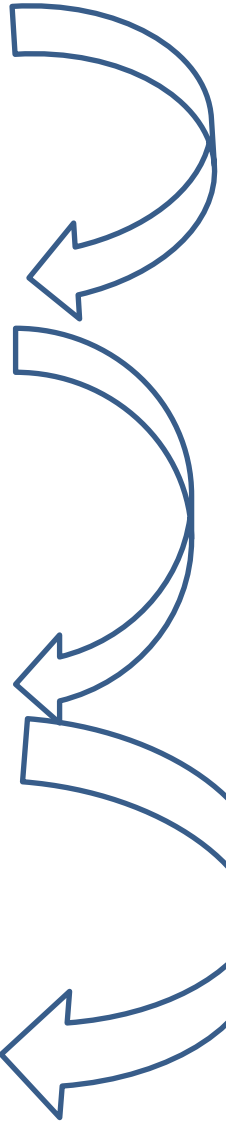
Working of Recursion

```
long int fact(f=1)
{
 if (f==1){ return 1;  }
 return (1*fact(0));
}
```

```
long int fact(f=2)
{
 if (f==1){ return 1; }
 return (2*fact(1));
}
```

```
long int fact(f=3)
{
 if (f==1) { return 1; }
 return (3*fact(2));
}
```

```
long int fact(f=4)
{
 if (f==1) { return 1; }
 return (4*fact(3));
}
```

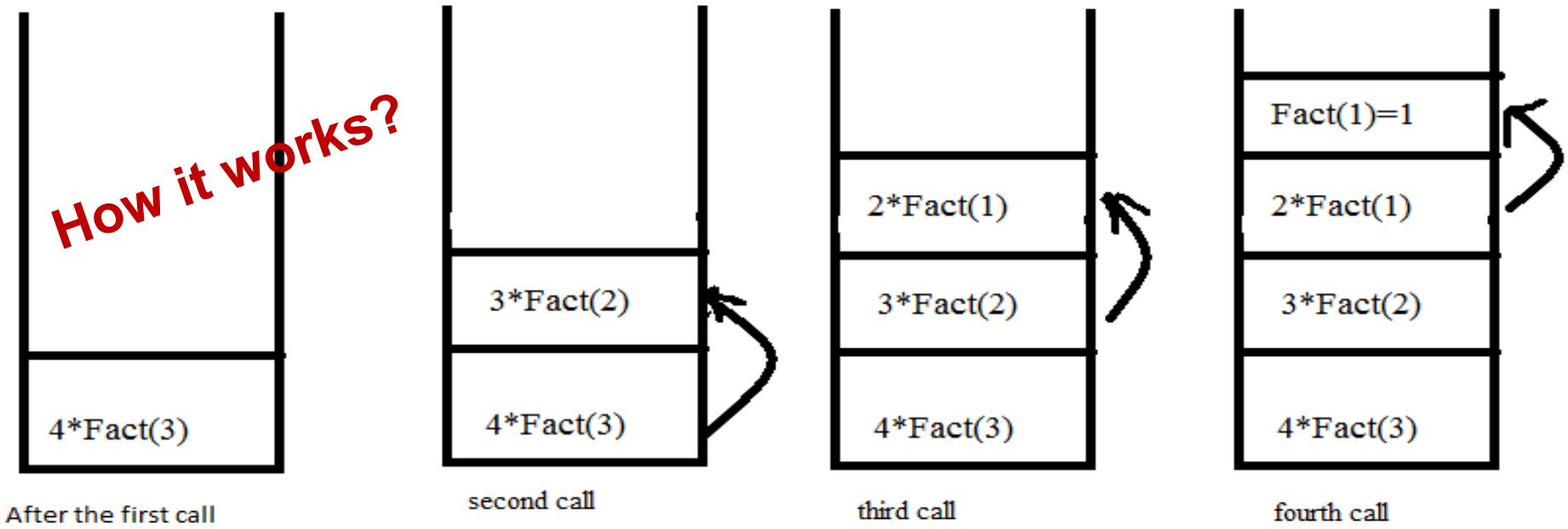


fact (1) Returns 1

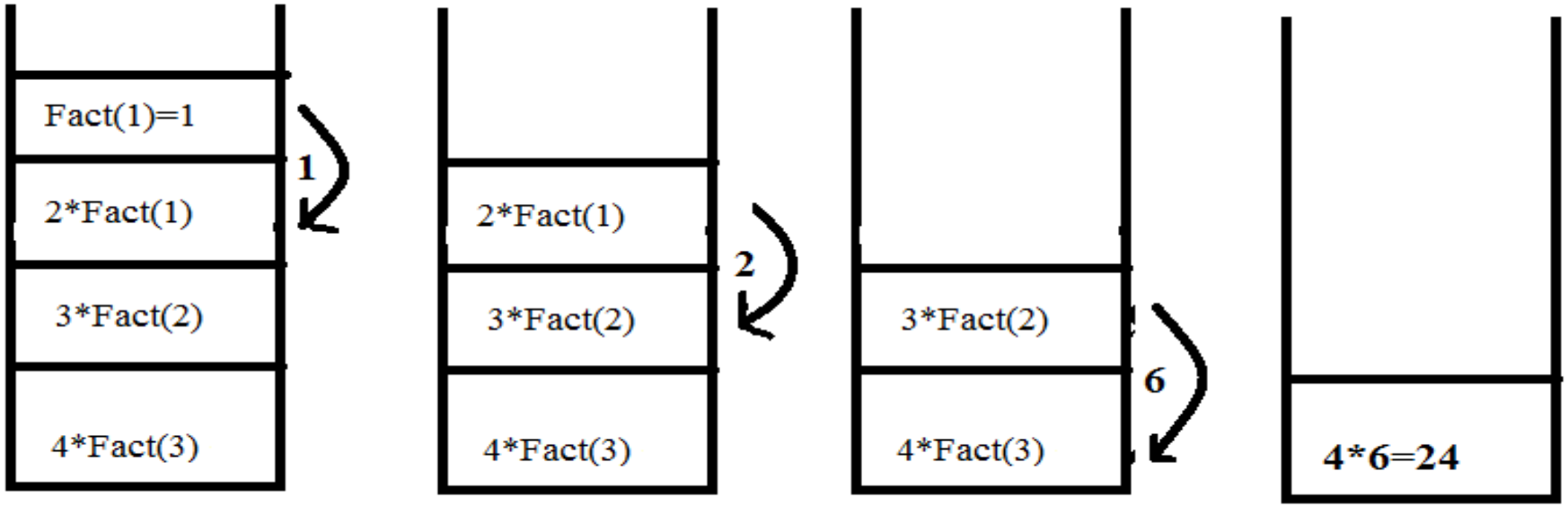
fact(2) returns 2

fact(3) returns 6

When function call happens previous variables gets stored in stack



Returning values from base case to caller function



Recursion – Example2 - Fibonacci

$$f(n) = \begin{cases} n = 0 & 0 \\ n = 1 & 1 \\ n > 1 & f(n-1) + f(n-2) \end{cases}$$

Can we compute Fibonacci series using recursion?

1+1=2
1+2=3
2+3=5
3+5=8
5+8=13
8+13=21
13+21=34
21+34=55
• • •

Do it Now: Write a program to find 'n' Fibonacci numbers using recursion

Recursion – Example2 - Fibonacci

```
#include<stdio.h>
int  main()
{
  int i;
  for (i = 0; i < 10; i++)
  printf("%d\t", fibonacci(i));
  return 0;
}
```

```
int fibonacci(int i)
{
   if(i == 0) return 0;
   if(i == 1) return 1;
   return (fibonacci(i-1) + fibonacci(i-2));
}
```

Recursion

Advantages:

- Simple to code (Not Always)
- Size of the code will be less
- Readable

- **Disadvantages:**
  - Difficult to understand in some algorithms
  - Stack Overflow in case of deep recursion

Class room Exercise - 6

1. Write a program to implement multiplication using addition. Use recursion.
2. Write a program to swap two numbers.
3. Write a recursive function to find the sum of n integers.

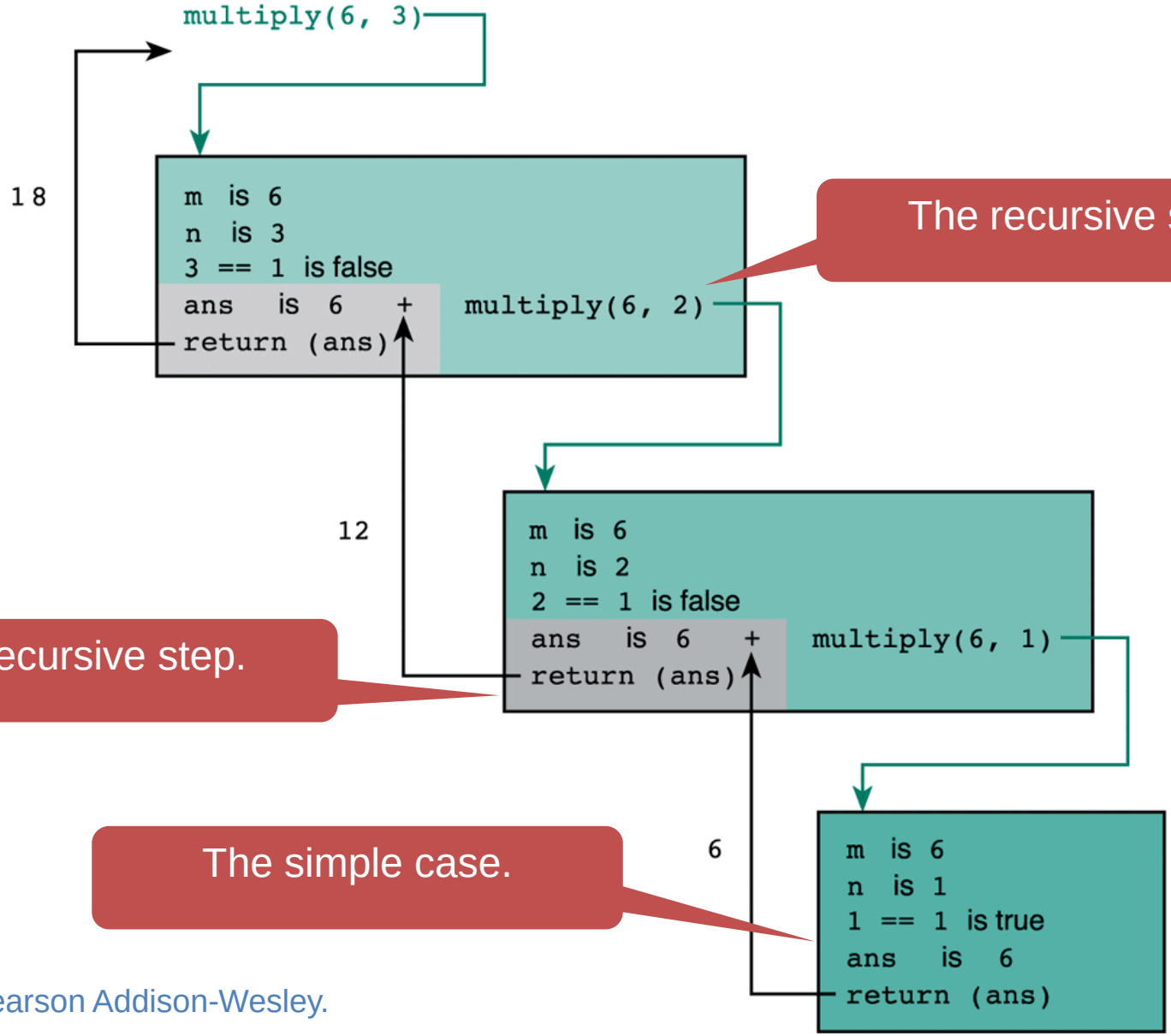
Class room Exercise - 6

```
/*
 *  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);
}
```

Trace of Function multiply(6,3)



Ref: 2004 Pearson Addison-Wesley.

THANK YOU