ECE 220: Computer Systems & Programming

Lecture 14: Recursion

Thomas Moon

February 27, 2024

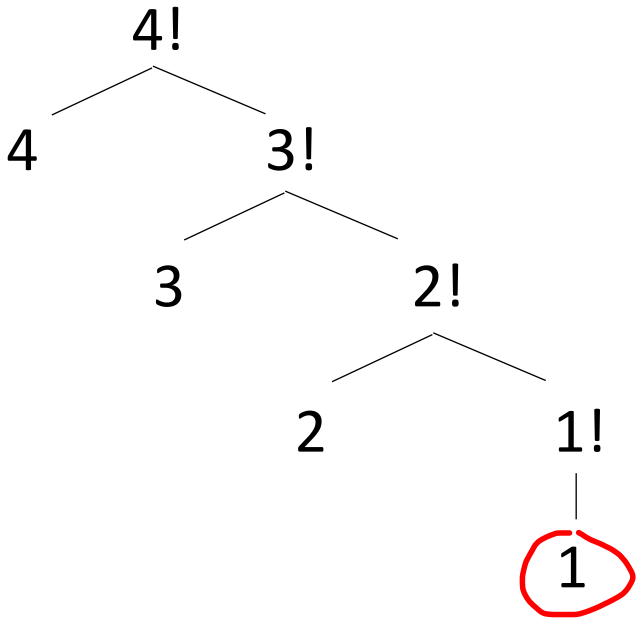
Example: Factorial

- Mathematical definition

$$n! = n \cdot (n-1) \cdot (n-2) \ldots \cdot 2 \cdot 1 \qquad for \quad n > 0$$

- Recursive form

$$n! = \begin{cases} 1, & if\ n = 1 \\ n \cdot (n-1)!, & else \end{cases}$$



Recursion

- A recursive function solves its task by calling itself on smaller pieces of data.
  - Each step 1) calls the same function
    with 2) a variant of the input
    until 3) a certain condition is satisfied.
  - Must have at least one base case (terminating case) that ends the recursive process.
  - Sometimes recursion results in a simpler solution than iteration version (can be used interchangeably).

Recursive Function

1. Base case (= Terminating case)
   - This case is mandatory as it provides the way out from the recurring loop.
   - The base case must provide a condition that will eventually become true and returns from the function. Otherwise, the run-time stack will overflow.

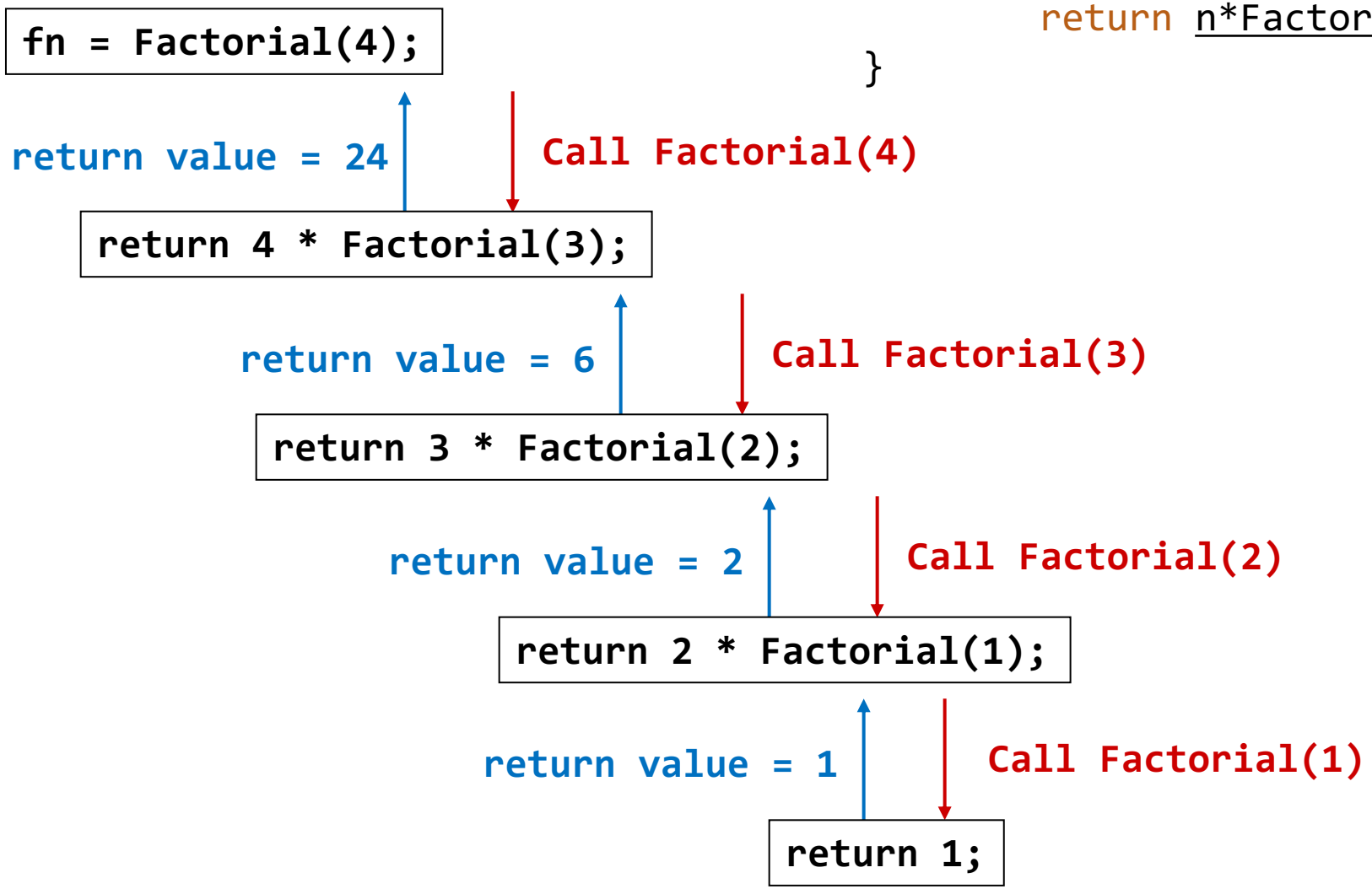
2. Recursive case (= Induction case)
   - This case returns a recursive call to itself. It breaks down the problem into smaller chunks that can be solved over and over by the same function.
   - The input to the next call gets induced gradually.

Example: Factorial

```
int Factorial_iter(int n){
    int i, result = 1;
    for(i=0;i<n;i++)
        result = result*(i+1);

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

Tracing the Factorial Calls

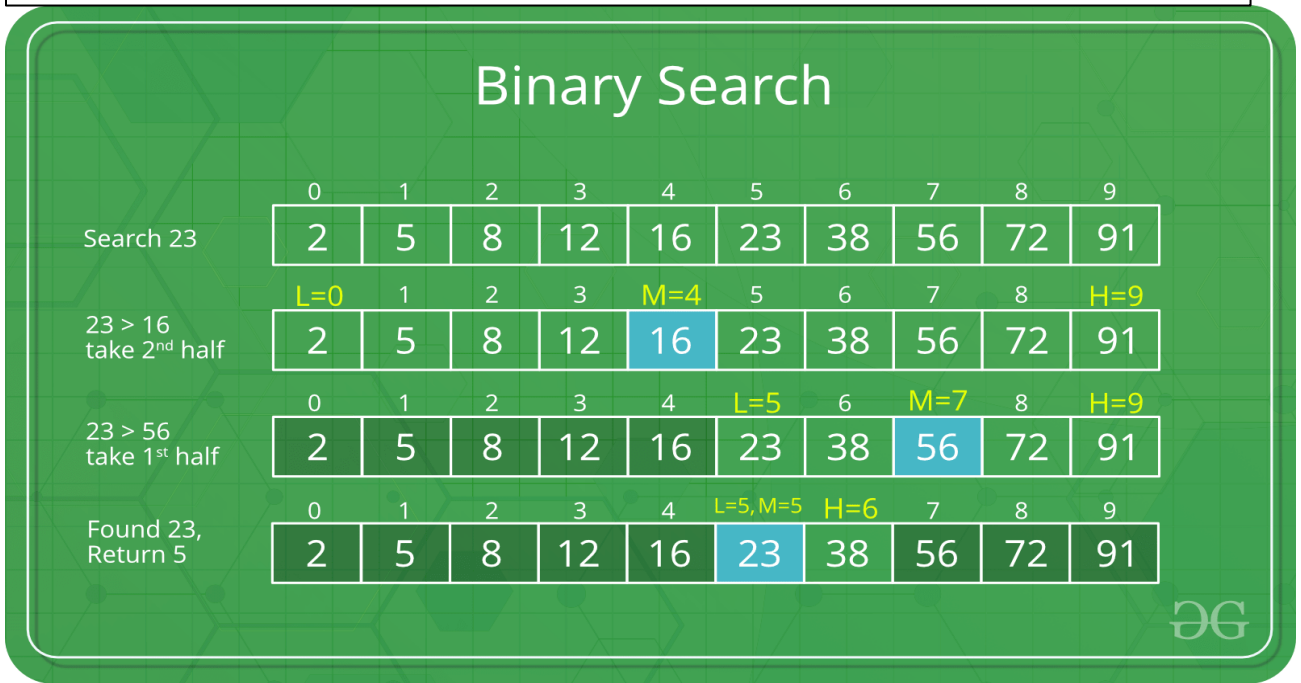
```
int Factorial(int n){
    if( n <= 1)
        return 1;
    else
        return n*Factorial(n-1);
}
```



Exercise- Binary Search

```
int binarySearch(int array[], int l, int h, int key)
{
    int mid;
    while( l <= h){
        mid = (l+h)/2;
        if(key < array[mid])
            h = mid -1;
        else if(key > array[mid])
            l = mid +1;
        else
            return mid;
    }
    return -1;
}
```

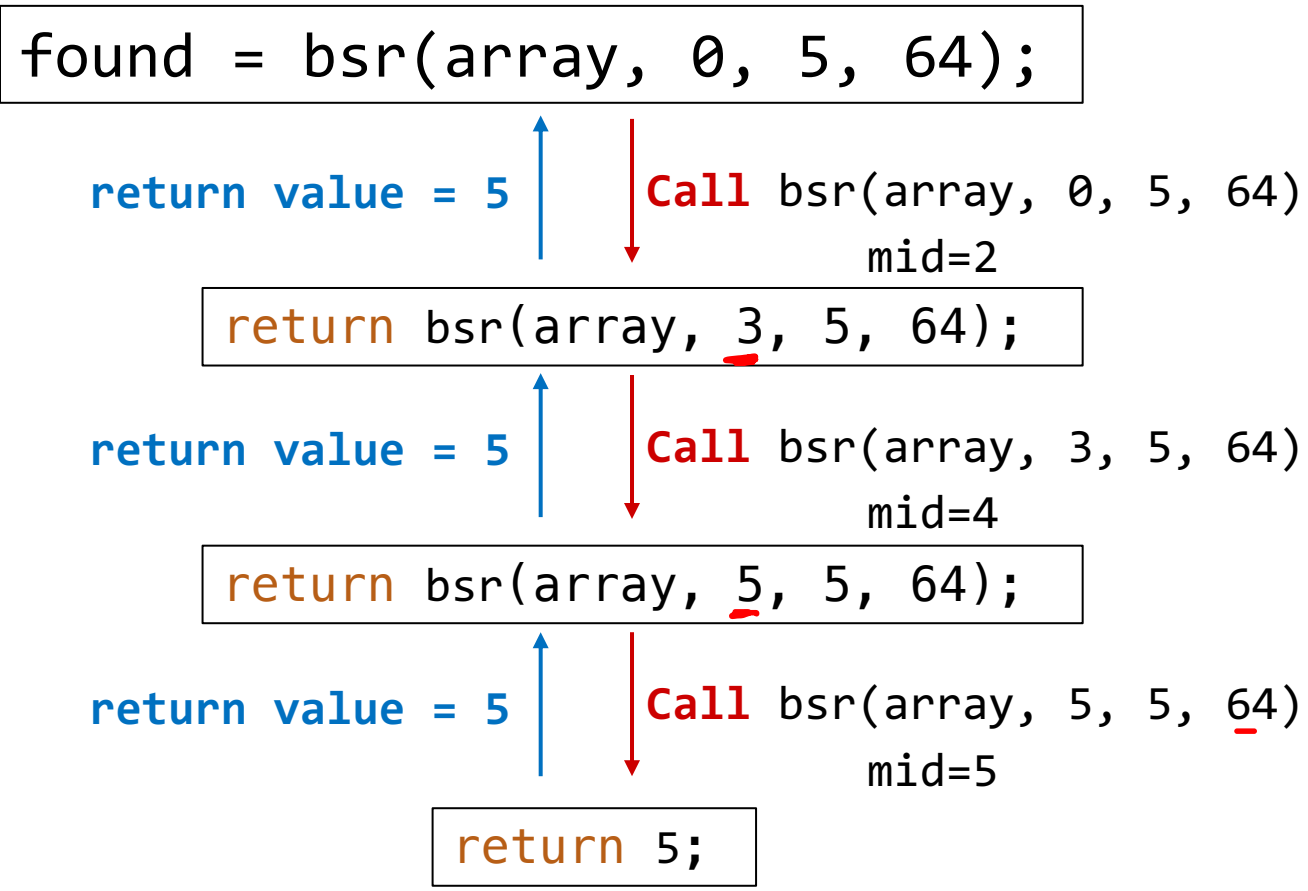
→ Can we implement binary search in **recursive** way?



Exercise- Binary Search: in Recursive way

```
int bsr(int array[], int l, int h, int key)
{
    int mid = (l+h)/2;

    if (l > h) // base case1
        return -1;
    else if(key == array[mid]) // base case2
        return mid;
    else if(key > array[mid])
        return     bsr(array,mid+1 ,h , key);
    else //key < array[mid]
        return     bsr(array,l , mid-1 , key)    ;
}
```

0	1	2	3	4	5
11	12	22	25	34	64



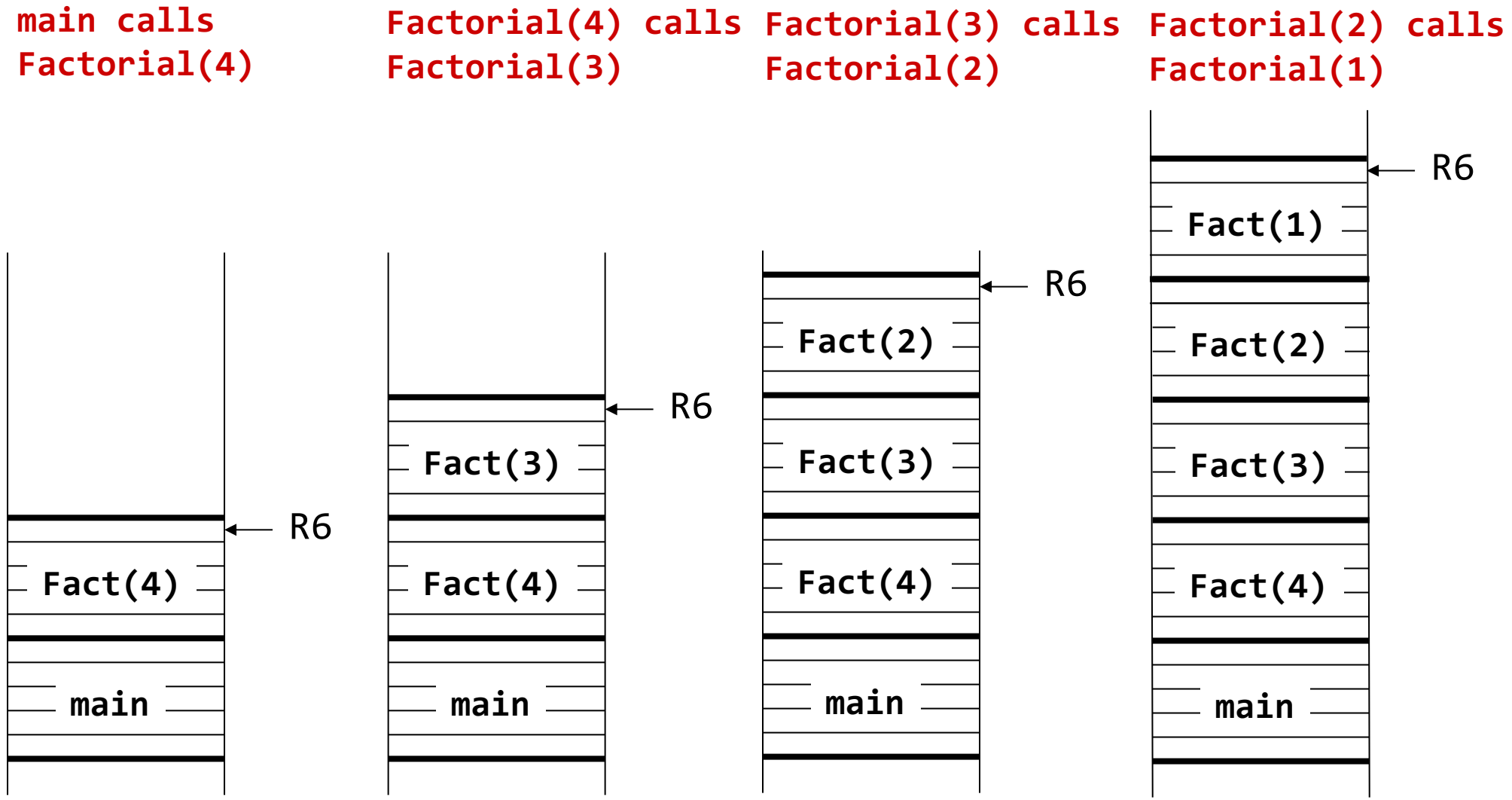

```
int bsr(int array[], int l, int h, int key)
{
    int mid = (l+h)/2;

    if (l > h) // base case1
        return -1;
    else if(key == array[mid]) // base case2
        return mid;
    else if(key > array[mid])
        return     bsr(array,mid+1 ,h , key);
    else //key < array[mid]
        return     bsr(array,l , mid-1 , key)    ;
}
```

Exercise: ???

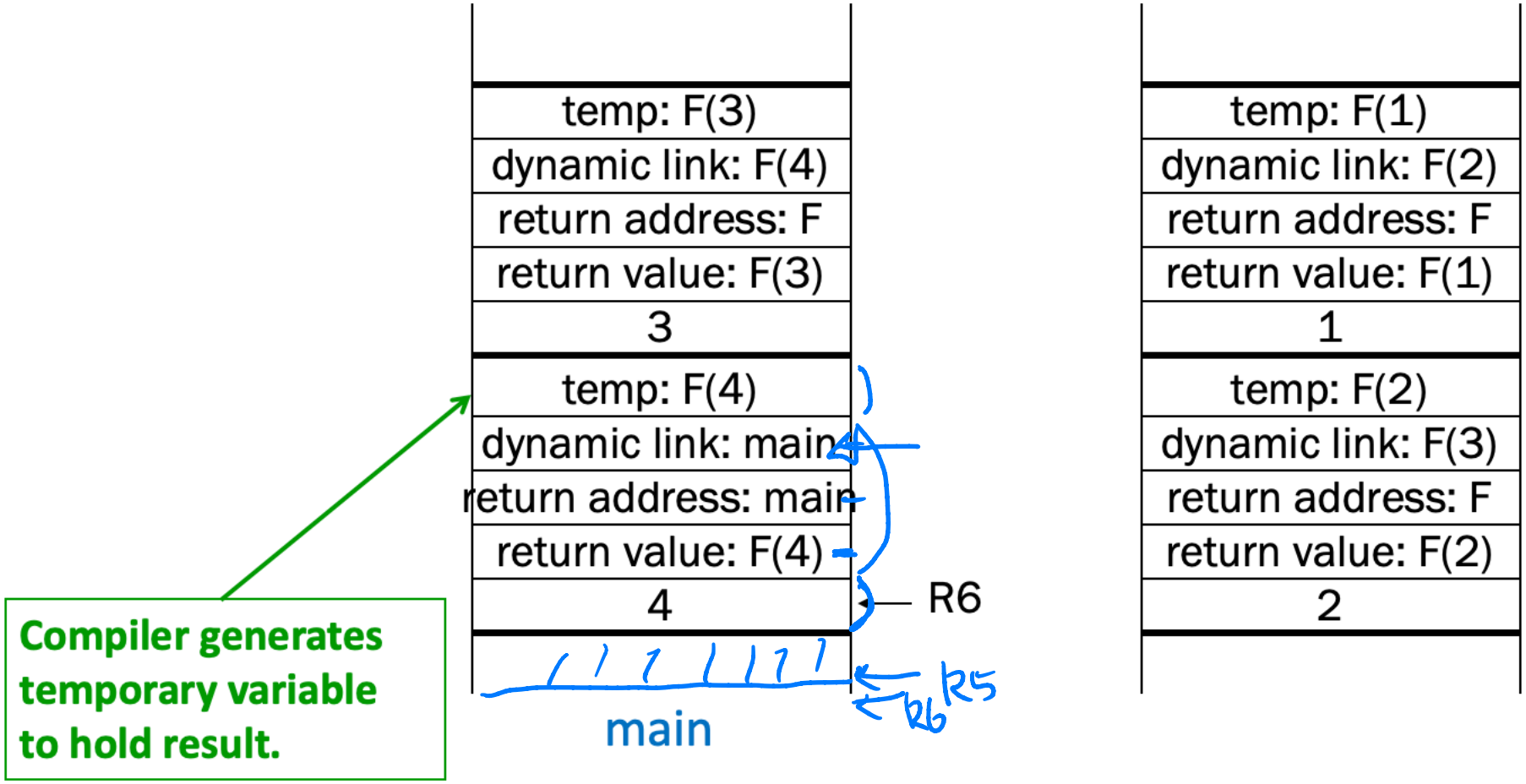
```
int foo(char *c){
    if(*c==0)
        return 0;
    else
        return foo(c+1) + 1;
}


int main(){
    char str[100] = "ABC";
    printf("%d", foo(str));
}
```

- What does the “foo” function return?
- How many function call?

Run-time Stack during Execution of Factorial

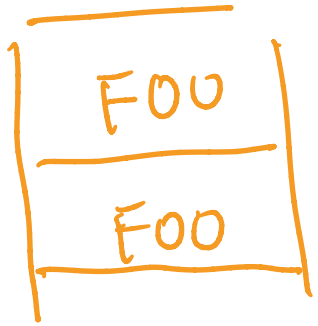


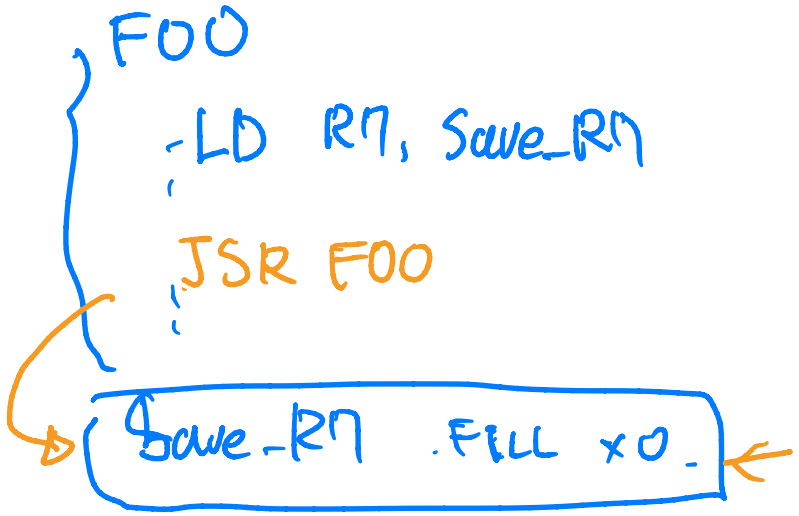
Activation Record



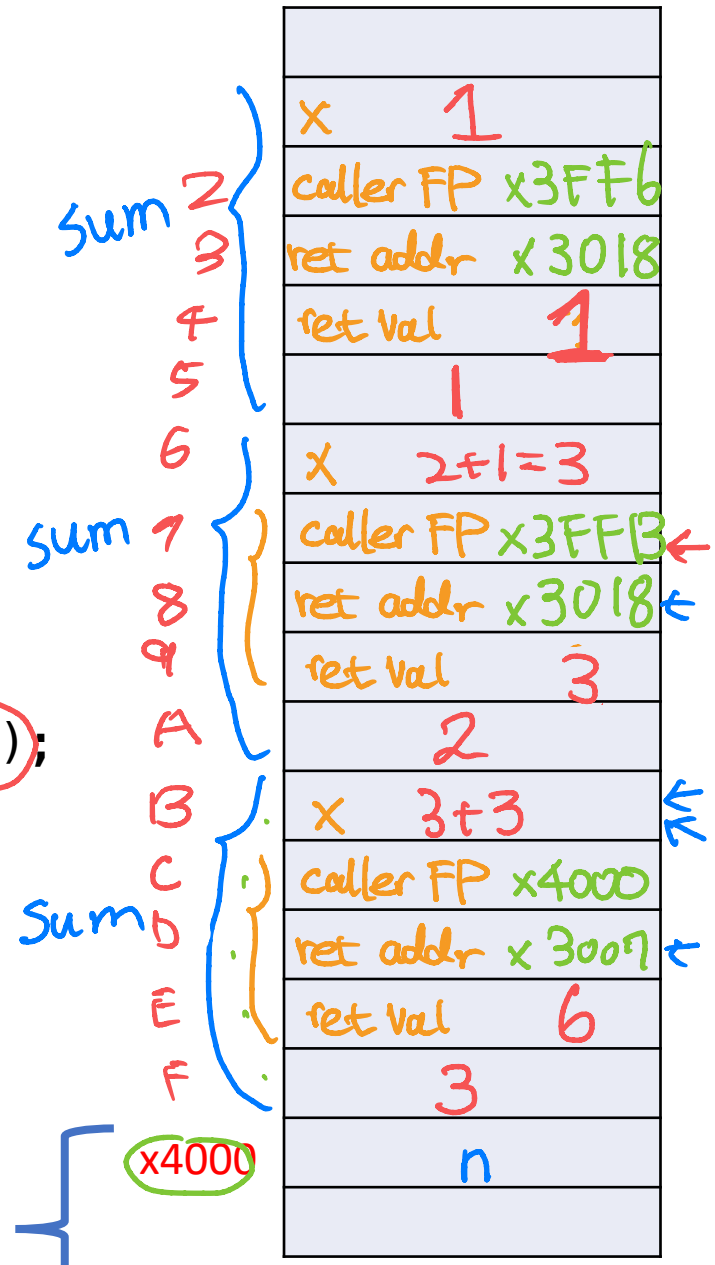
C to LC3: Recursion with Run-time Stack

```c
int main(){
    int n;
    n = sum(3);
}

int sum(int n){
    int x;
    if(n<=1){
        x = 1;
        return x;
    else{
        x = n + sum(n-1);
        return x;
    }
}
```





```c
int main(){
    int n;
    n = sum(3);
}

int sum(int n){
    int x;
    if(n<=1){
        x = 1;
        return x;
    else{
        x = n + sum(n-1);
        return x;
    }
}
```



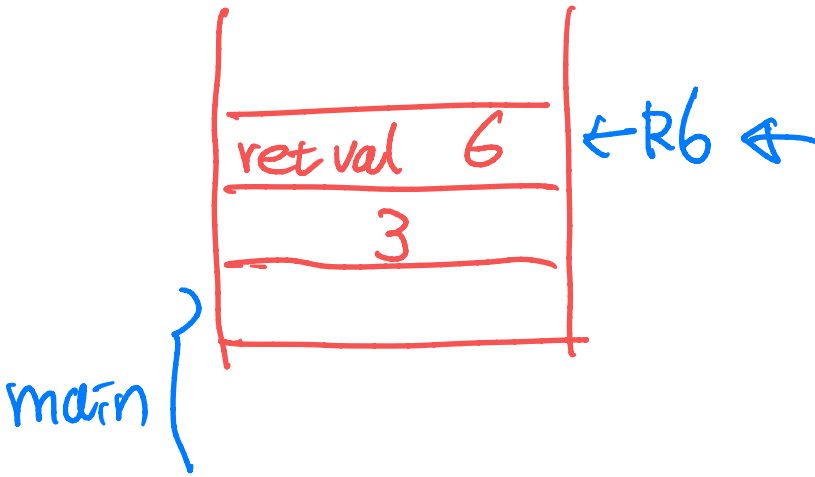
activation record of main

```
.ORIG x3000
; push argument
; ...
x3006 JSR SUM
; pop return value and tear-down
; ...
    HALT

SUM
; push bookkeeping & local
; ...
; if-else
; ...
; push n-1
; ...
; recursive call
x3017 JSR SUM
; set return value n+sum(n-1)
; and tear-down
; ...
    RET
```

push '3'

C to LC3: Recursion

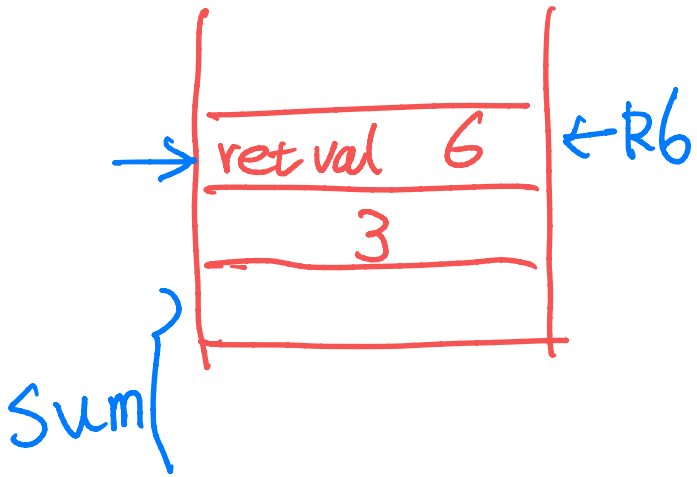
```c
int main(){
    int n;
    n = sum(3);
}
```




```
.ORIG x3000
; init R6 and R5
    LD  R6, STACK_TOP
    ADD R5, R6, #0
; push argument
    ADD R6, R6, #-1
    AND R0, R0, #0
    ADD R0, R0, #3
    STR R0, R6, #0
; call subroutine sum(n);
    JSR SUM
; pop return value from run-time stack
    LDR R0, R6, #0
    ADD R6, R6, #-2
    STR R0, R5, #0
    HALT
```

C to LC3: Recursion

```c
int sum(int n){
    int x;
    if(n<=1){
        x = 1;
        return x;
    else{
        x = n + sum(n-1);
        return x;
    }
}
```




```
SUM
; push callee's bookkeeping info onto the run-time stack
; . . .

; push n-1 onto run-time stack
    ADD R6, R6, #-1
    STR R2, R6, #0 ; R2 = n-1
; call sum subroutine
    JSR SUM
; pop return value from run-time stack (R0 = sum(n-1))
    LDR R0, R6, #0
    ADD R6, R6, #1
; pop function argument from the run-time stack
    ADD R6, R6, #1
; add n by the return value (n+sum(n-1))
    LDR R1, R5, #4
    ADD R0, R0, R1
; store result in local var
    STR R0, R5, #0
```

n-1

Full code: https://github.com/tmoon-illinois/ece220_sp24/blob/main/lec12/sum.asm