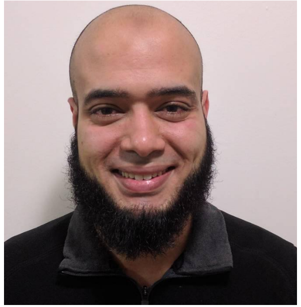
Programming4kids

Recursive Functions

Mostafa Saad Ibrahim
Computer Vision Researcher @ Huawei Canada
PhD - Simon Fraser University
Bachelor / Msc - FCI Cairo University

Ex-(Software Engineer / Teaching Assistant)



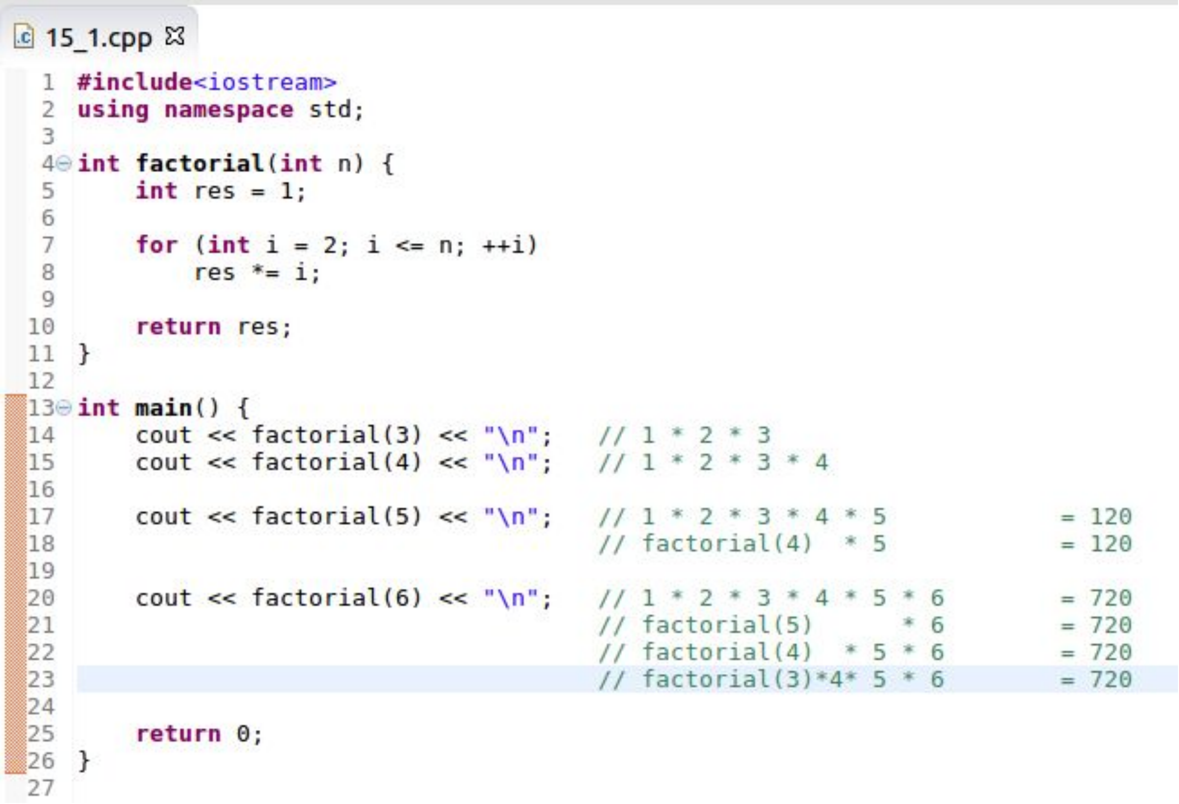
Problem and subproblems

- Sometimes we can decompose a problem to set of sub-problems
- E.g. Print all prime numbers that are palindrome and < 1000000
- We have 2 sub-problems
    - bool is_prime(int n)
    - bool is_palindrome(int n)
- Now we iterate from 1 to 1000000
    - If number satisfy the 2 conditions: count it
- What if the sub-problem is same type as the problem? Recursion!

Recall the factorial

- factorial(6) = **1 * 2 * 3 * 4 * 5** * 6
- factorial(5) = **1 * 2 * 3 * 4 * 5**
- factorial(4) = 1 * 2 * 3 * 4
- factorial(3) = 1 * 2 * 3
- factorial(2) = 1 * 2
- factorial(1) = 1
- Think for a few minutes:
    - What is relation between factorial(6) and factorial(5)?
    - Can you know factorial(6) if you know factorial(5)?

Factorial

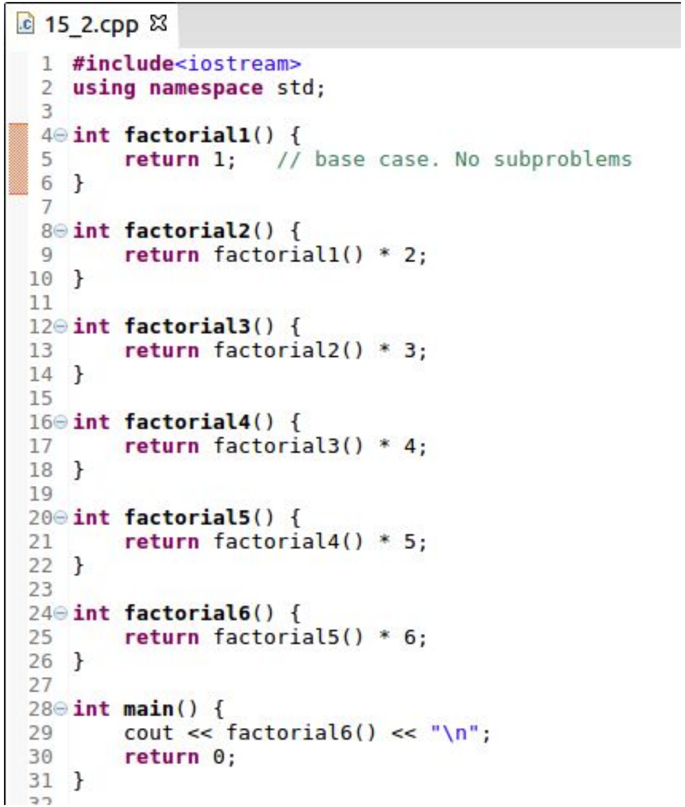


```
#include<iostream>
using namespace std;

int factorial(int n) {
    int res = 1;

    for (int i = 2; i <= n; ++i)
        res *= i;

    return res;
}

int main() {
    cout << factorial(3) << "\n";  // 1 * 2 * 3
    cout << factorial(4) << "\n";  // 1 * 2 * 3 * 4

    cout << factorial(5) << "\n";  // 1 * 2 * 3 * 4 * 5            = 120
                                    // factorial(4)  * 5            = 120

    cout << factorial(6) << "\n";  // 1 * 2 * 3 * 4 * 5 * 6        = 720
                                    // factorial(5)      * 6        = 720
                                    // factorial(4)  * 5 * 6        = 720
                                    // factorial(3)*4* 5 * 6        = 720

    return 0;
}
```

Factorial: Problem and subproblem

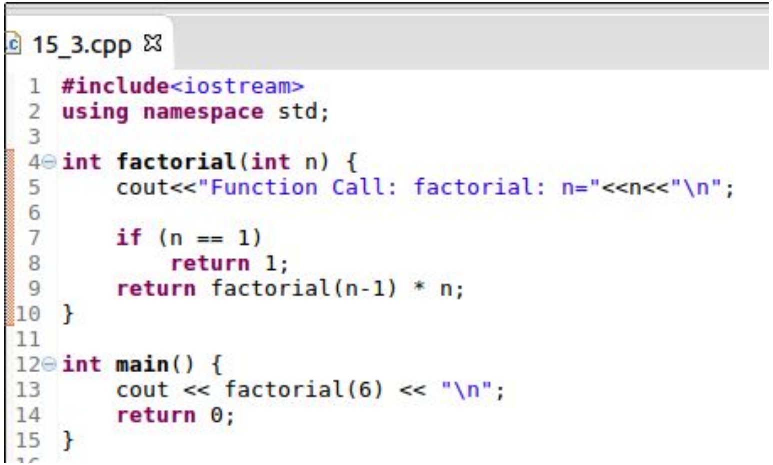
- Let say we want to solve factorial(6)
  - This is our problem
  - We can solve it directly with 1*2*3*4*5*6
- Another thinking is: can we think of it is
  - What is factorial(5)? A simpler subproblem
  - Would it help if u know its answer? Yes: 6 * factorial(5) = factorial 6
  - Same logic for factorial(5). It is 5 * factorial(4).
- Going for ever in smaller sub-problems? No
  - There must be a case where no more subproblems. We call it basecase
  - Factorial 1 = 1

Factorial: Problem and subproblem



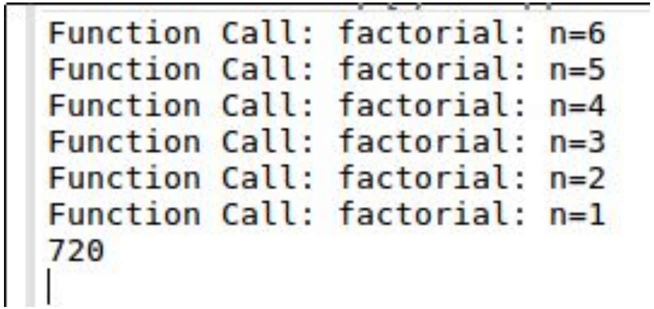
```cpp
#include<iostream>
using namespace std;

int factorial1() {
	return 1;	// base case. No subproblems
}

int factorial2() {
	return factorial1() * 2;
}

int factorial3() {
	return factorial2() * 3;
}

int factorial4() {
	return factorial3() * 4;
}

int factorial5() {
	return factorial4() * 5;
}

int factorial6() {
	return factorial5() * 6;
}

int main() {
	cout << factorial6() << "\n";
	return 0;
}
```

Factorial: A recursive function



```cpp
#include<iostream>
using namespace std;

int factorial(int n) {
    cout<<"Function Call: factorial: n="<<n<<"\n";

    if (n == 1)
        return 1;
    return factorial(n-1) * n;
}

int main() {
    cout << factorial(6) << "\n";
    return 0;
}
```

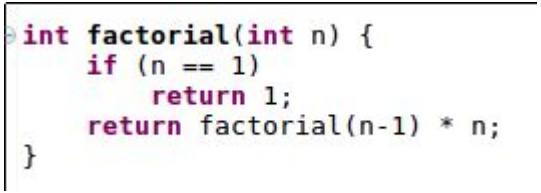
- A recursive function: Function that calls itself with smaller input (supproblem) till reaches baseline



```
Function Call: factorial: n=6
Function Call: factorial: n=5
Function Call: factorial: n=4
Function Call: factorial: n=3
Function Call: factorial: n=2
Function Call: factorial: n=1
720
```

Let’s trace it

- Call **Factorial**(6)
  - If 6 == 1? False
  - Call **Factorial** (5) and multiply results with 6
    - If 5 == 1? False
    - Call **Factorial** (4) and multiply results with 5
      - If 4 == 1? False
      - Call **Factorial** (3) and multiply results with 4
        - If 3 == 1? False
        - Call **Factorial** (2) and multiply results with 3
          - If 2 == 1? False
          - Call **Factorial** (1) and multiply results with 2
            - If 1 == 1? True
              - Return 1

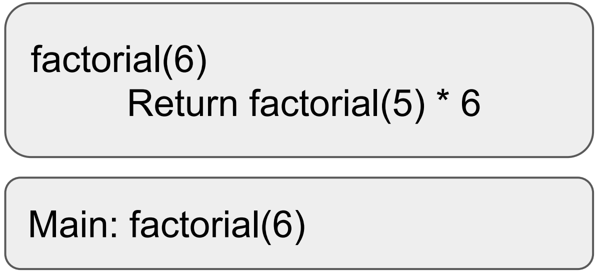


```
int factorial(int n) {
    if (n == 1)
        return 1;
    return factorial(n-1) * n;
}
```

Let’s trace it

Main: factorial(6)

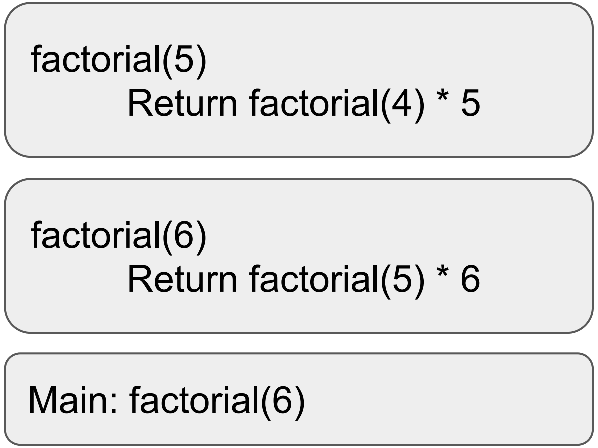
Let’s trace it



factorial(6)
    Return factorial(5) * 6

Main: factorial(6)

Let’s trace it

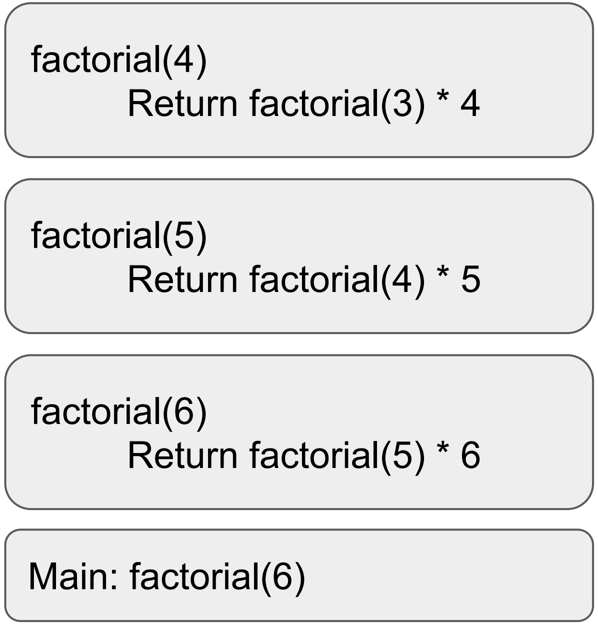


```
factorial(5)
        Return factorial(4) * 5
```

```
factorial(6)
        Return factorial(5) * 6
```

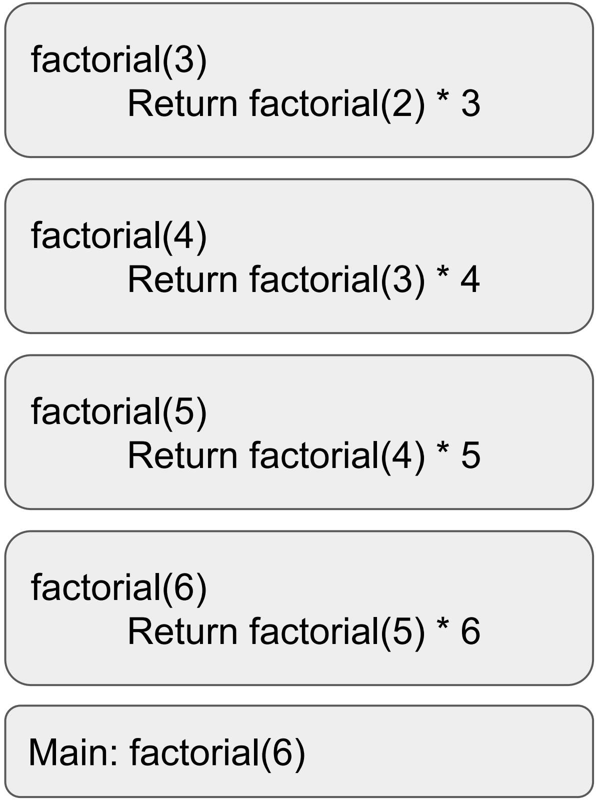
```
Main: factorial(6)
```

Let’s trace it



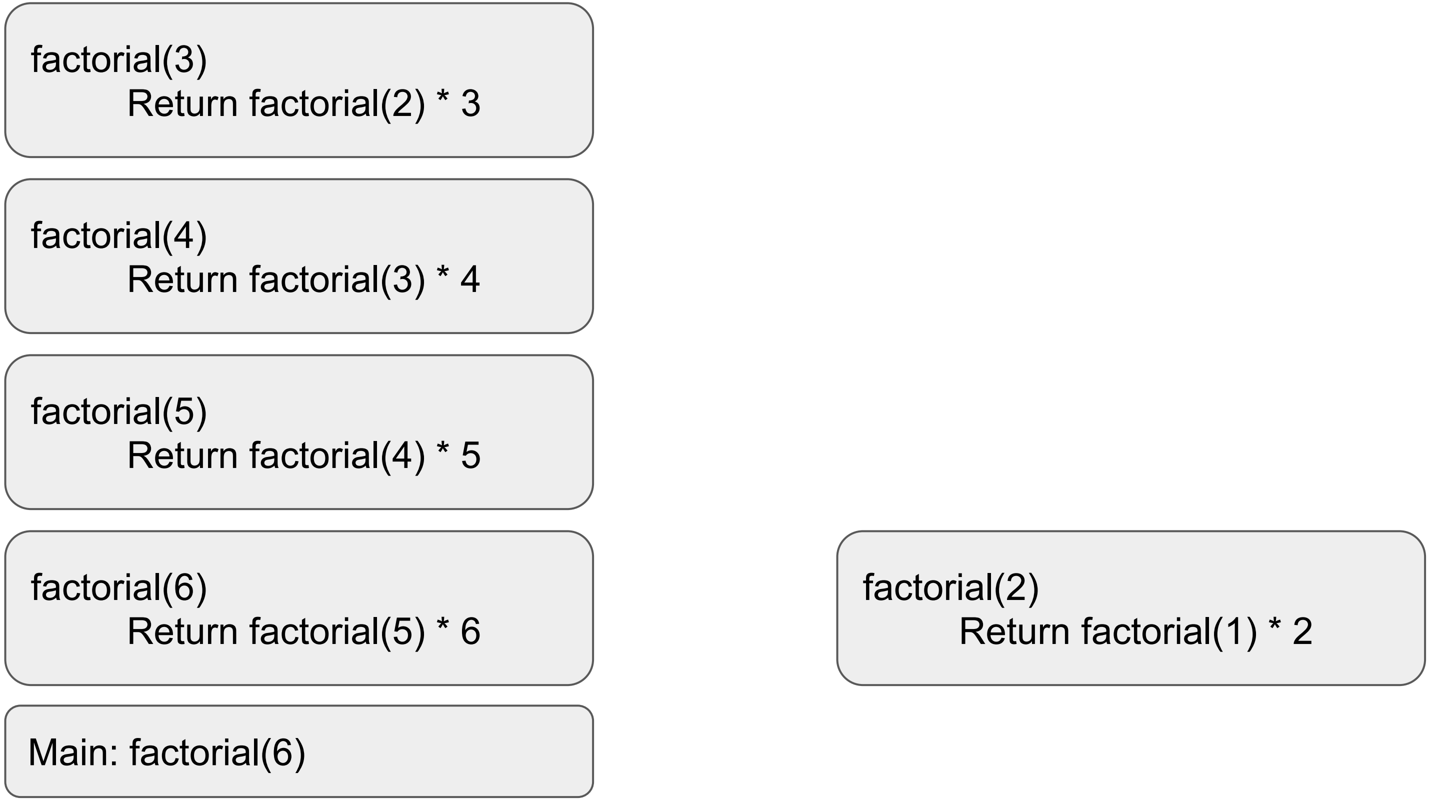
```
factorial(4)
        Return factorial(3) * 4

factorial(5)
        Return factorial(4) * 5

factorial(6)
        Return factorial(5) * 6

Main: factorial(6)
```

Let’s trace it



```
factorial(3)
        Return factorial(2) * 3

factorial(4)
        Return factorial(3) * 4

factorial(5)
        Return factorial(4) * 5

factorial(6)
        Return factorial(5) * 6

Main: factorial(6)
```

Let’s trace it



factorial(3)
Return factorial(2) * 3

factorial(4)
Return factorial(3) * 4

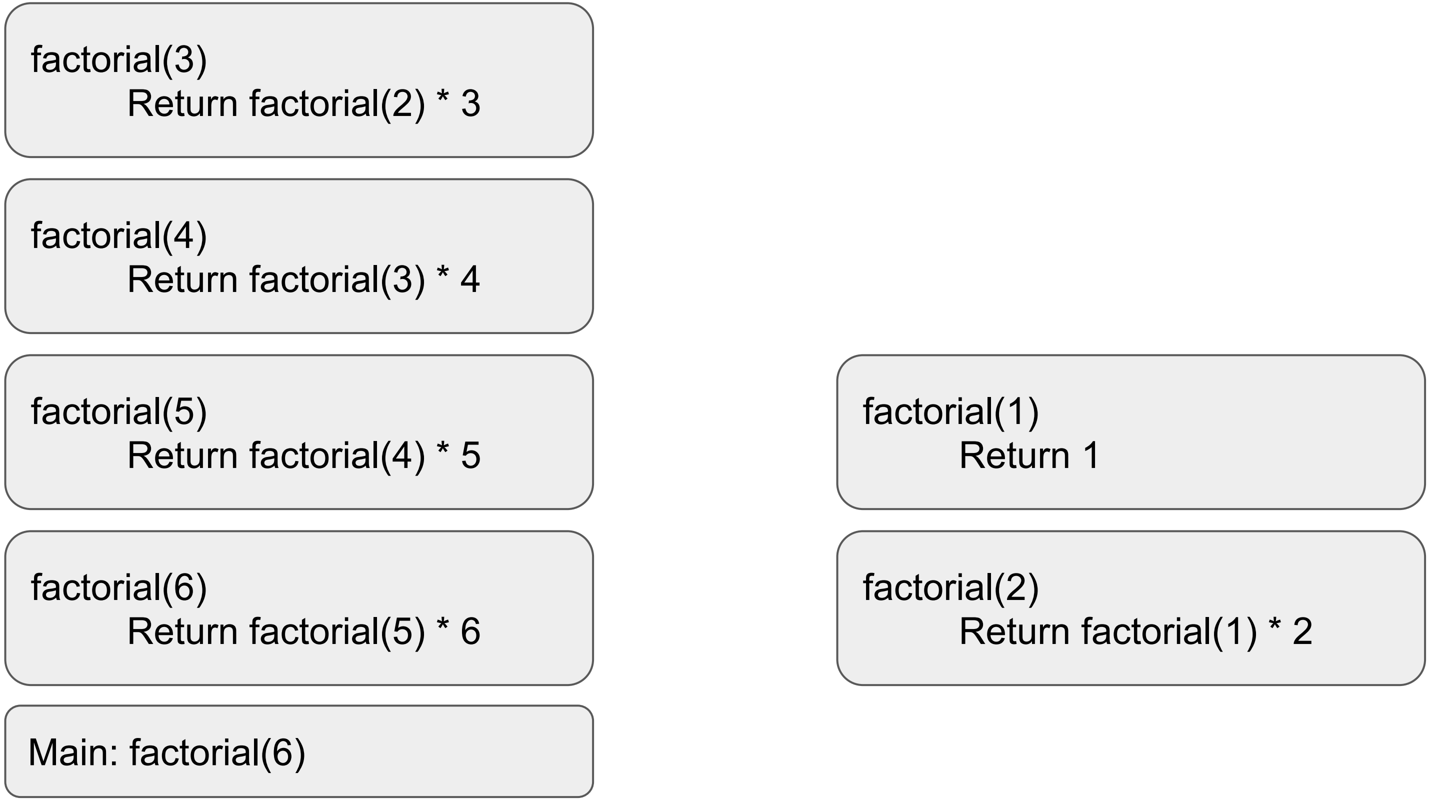
factorial(5)
Return factorial(4) * 5

factorial(6)
Return factorial(5) * 6

Main: factorial(6)

factorial(2)
Return factorial(1) * 2

Let’s trace it



factorial(3)
    Return factorial(2) * 3

factorial(4)
    Return factorial(3) * 4

factorial(5)
    Return factorial(4) * 5

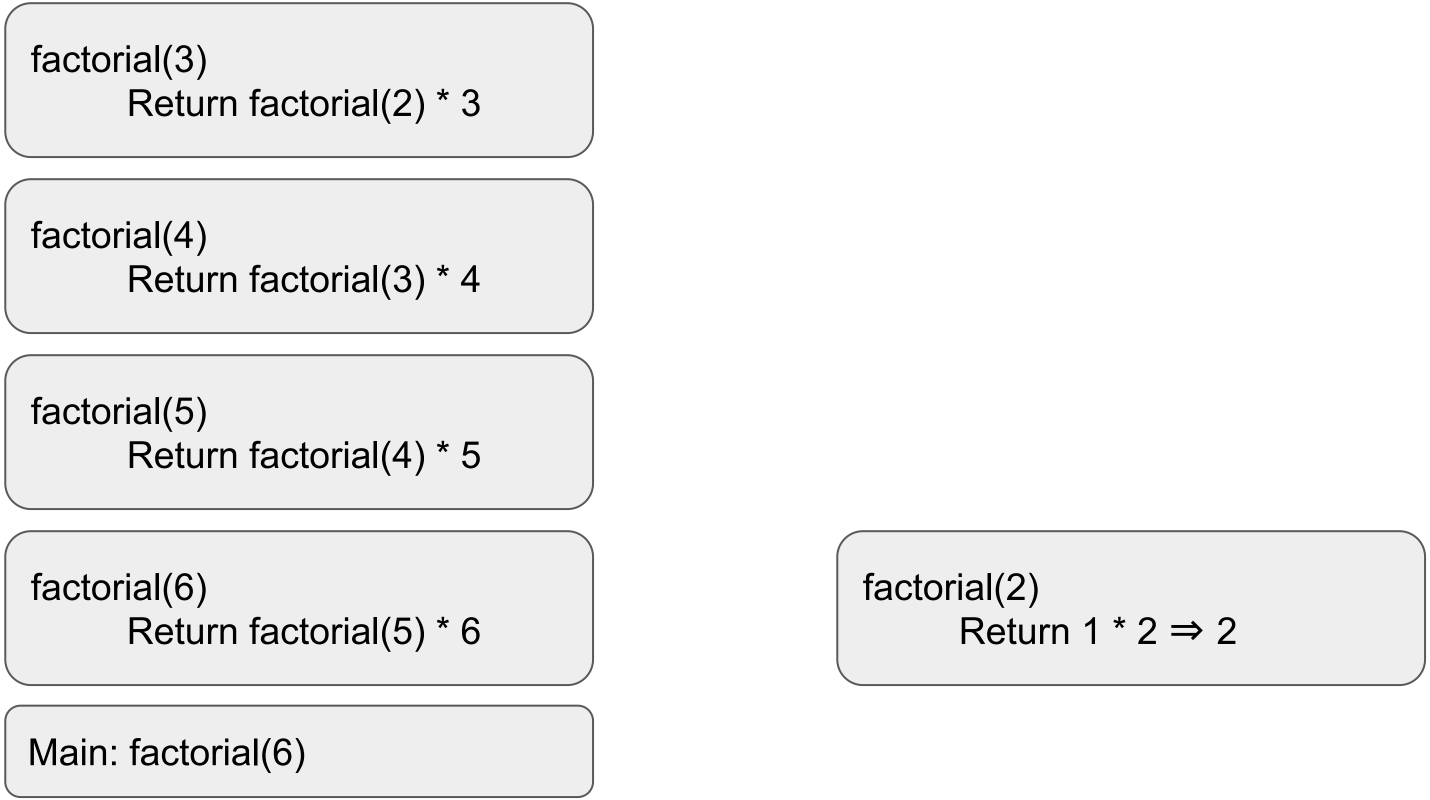
factorial(6)
    Return factorial(5) * 6

Main: factorial(6)

factorial(1)
    Return 1

factorial(2)
    Return factorial(1) * 2

Let’s trace it



factorial(3)
Return factorial(2) * 3

factorial(4)
Return factorial(3) * 4

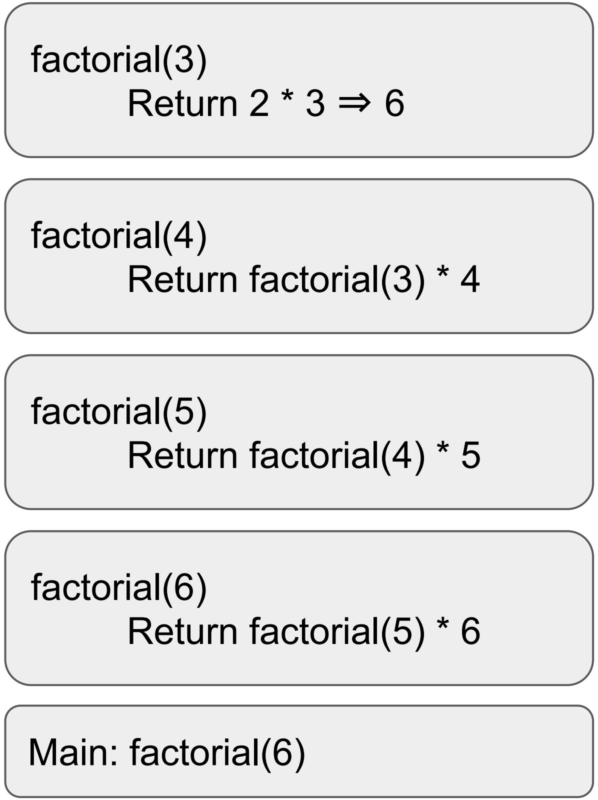
factorial(5)
Return factorial(4) * 5

factorial(6)
Return factorial(5) * 6

Main: factorial(6)

factorial(2)
Return 1 * 2 ⇒ 2

Let’s trace it



factorial(3)
    Return 2 * 3 ⇒ 6

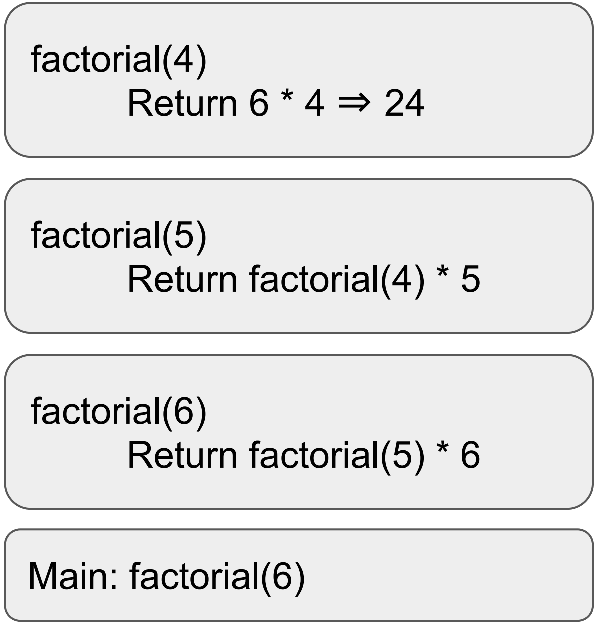
factorial(4)
    Return factorial(3) * 4

factorial(5)
    Return factorial(4) * 5

factorial(6)
    Return factorial(5) * 6

Main: factorial(6)

Let’s trace it



factorial(4)

Return 6 * 4 ⇒ 24

factorial(5)

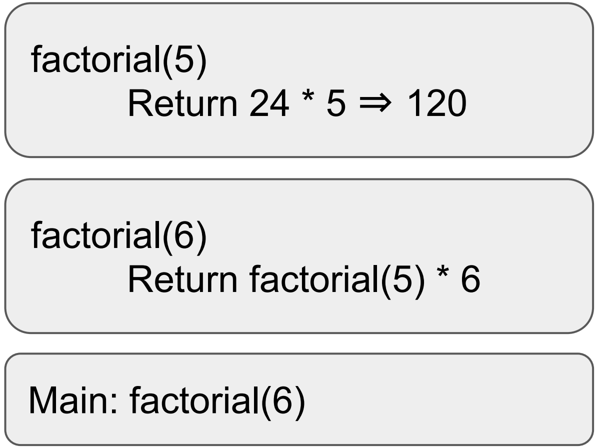
Return factorial(4) * 5

factorial(6)

Return factorial(5) * 6

Main: factorial(6)

Let’s trace it

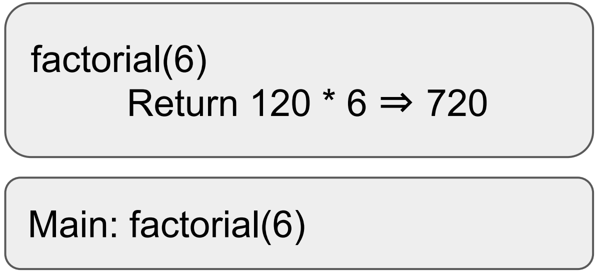


factorial(5)
Return 24 * 5 ⇒ 120

factorial(6)
Return factorial(5) * 6

Main: factorial(6)

Let’s trace it



factorial(6)

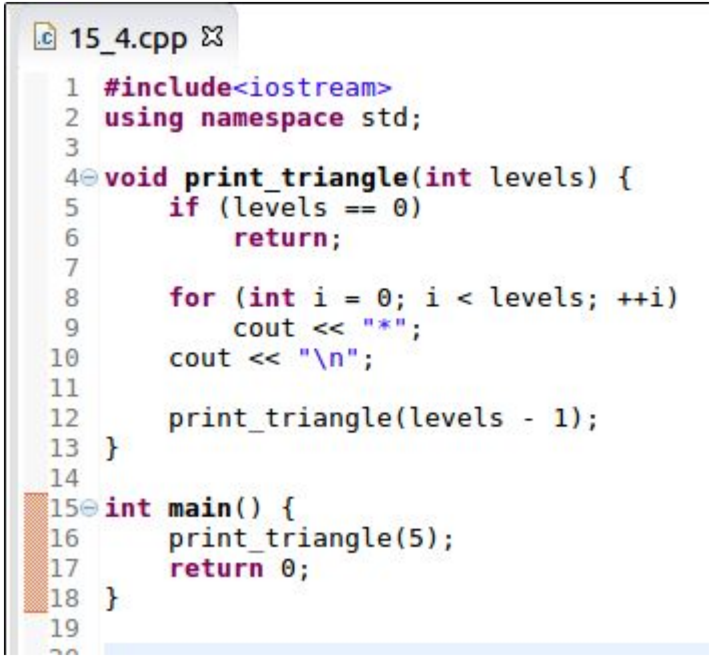
Return 120 * 6 ⇒ 720

Main: factorial(6)

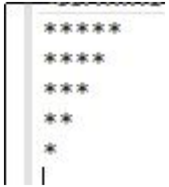
Let’s trace it

Main: factorial(6) ⇒ 720

Print a Triangle (v1)



```
#include<iostream>
using namespace std;

void print_triangle(int levels) {
    if (levels == 0)
        return;

    for (int i = 0; i < levels; ++i)
        cout << "*";
    cout << "\n";

    print_triangle(levels - 1);
}

int main() {
    print_triangle(5);
    return 0;
}

```



```
*****
****
***
**
*
```

Let’s trace it

```
*****
```

```
print_triangle(5)
        Print 5 stars
        print_triangle(4)
```

Let’s trace it

```
*****
****
```

```
print_triangle(4)
        Print 4 stars
        print_triangle(3)
```

```
print_triangle(5)
        Print 5 stars
        print_triangle(4)
```

Let’s trace it

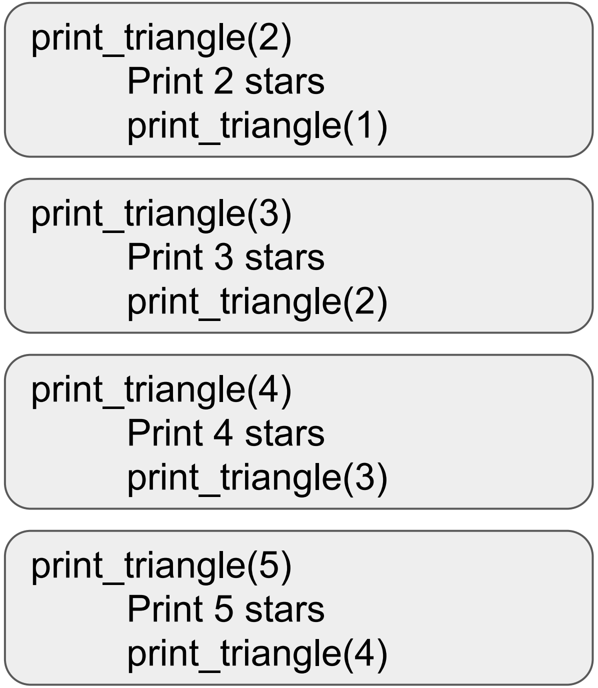
```
*****
****
***
```

```
print_triangle(3)
        Print 3 stars
        print_triangle(2)
```

```
print_triangle(4)
        Print 4 stars
        print_triangle(3)
```

```
print_triangle(5)
        Print 5 stars
        print_triangle(4)
```

Let’s trace it



```
print_triangle(2)
        Print 2 stars
        print_triangle(1)
```

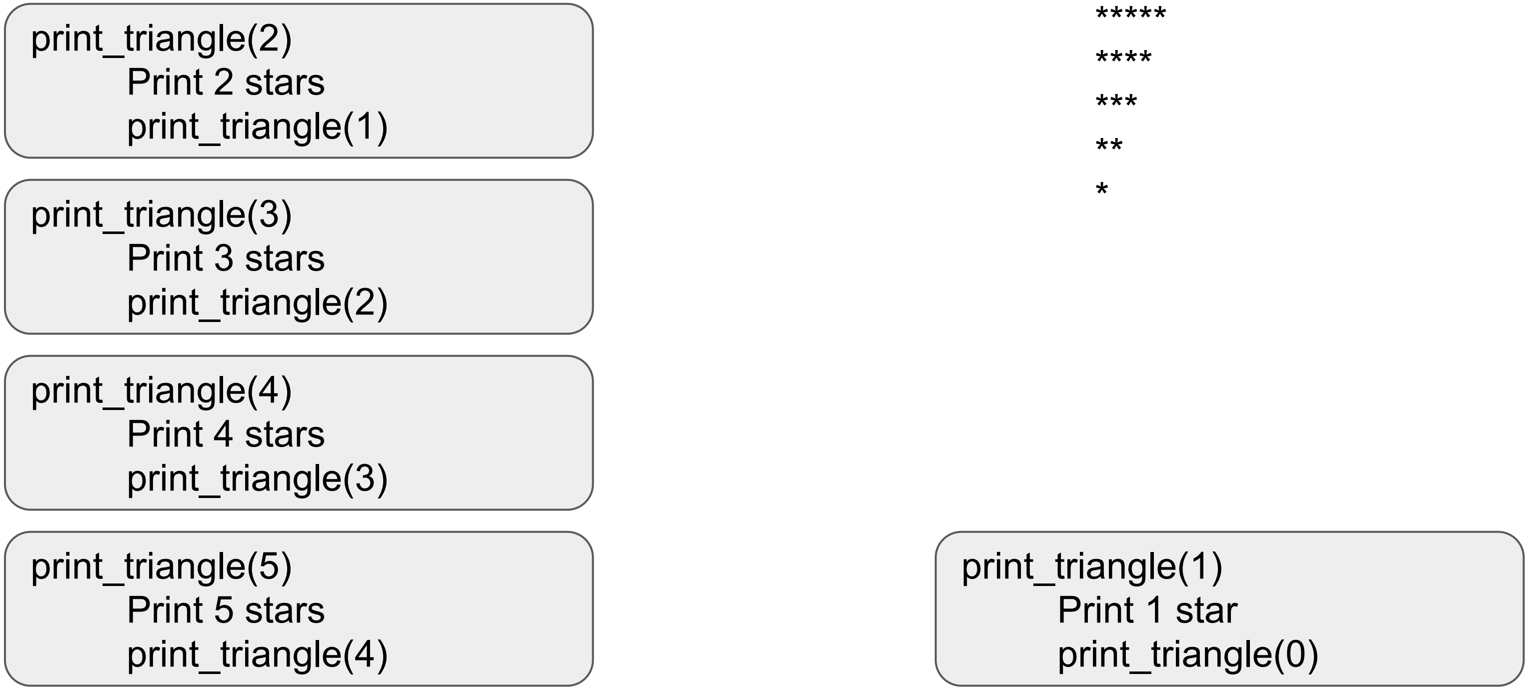
```
print_triangle(3)
        Print 3 stars
        print_triangle(2)
```

```
print_triangle(4)
        Print 4 stars
        print_triangle(3)
```

```
print_triangle(5)
        Print 5 stars
        print_triangle(4)
```

```
*****
****
***
**
```

Let’s trace it



```
print_triangle(2)
        Print 2 stars
        print_triangle(1)
```

```
print_triangle(3)
        Print 3 stars
        print_triangle(2)
```

```
print_triangle(4)
        Print 4 stars
        print_triangle(3)
```

```
print_triangle(5)
        Print 5 stars
        print_triangle(4)
```

```
*****
****
***
**
*
```

```
print_triangle(1)
        Print 1 star
        print_triangle(0)
```

Let’s trace it

```
print_triangle(2)
        Print 2 stars
        print_triangle(1)
```

```
print_triangle(3)
        Print 3 stars
        print_triangle(2)
```

```
print_triangle(4)
        Print 4 stars
        print_triangle(3)
```

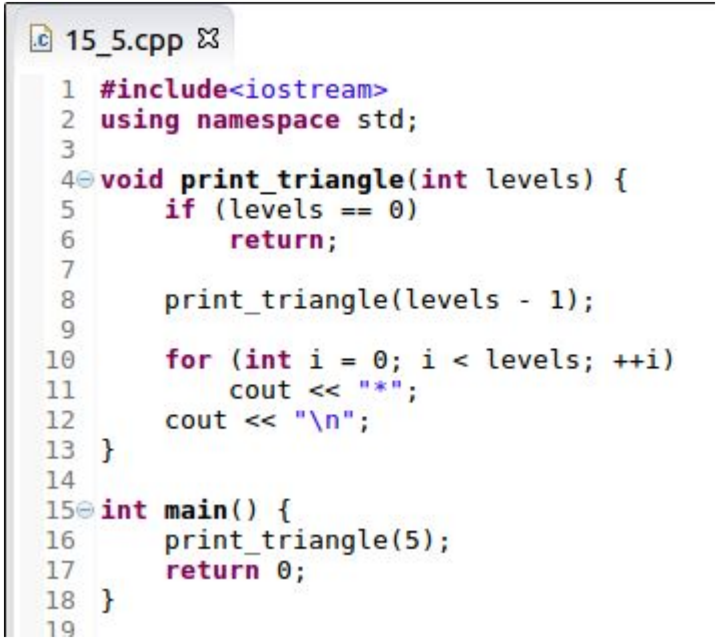
```
print_triangle(5)
        Print 5 stars
        print_triangle(4)
```

```
*****
****
***
**
*
```

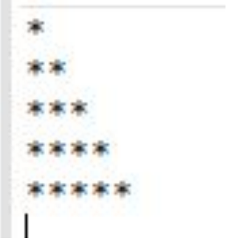
```
print_triangle(0)
        Return
```

```
print_triangle(1)
        Print 1 star
        print_triangle(0)
```

Print a Triangle (v2)



```cpp
#include<iostream>
using namespace std;

void print_triangle(int levels) {
    if (levels == 0)
        return;

    print_triangle(levels - 1);

    for (int i = 0; i < levels; ++i)
        cout << "*";
    cout << "\n";
}

int main() {
    print_triangle(5);
    return 0;
}

```

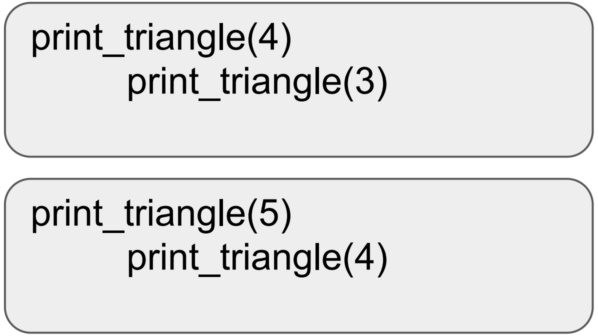


```
*
**
***
****
*****
```

Let’s trace it

```
print_triangle(5)
        print_triangle(4)
```

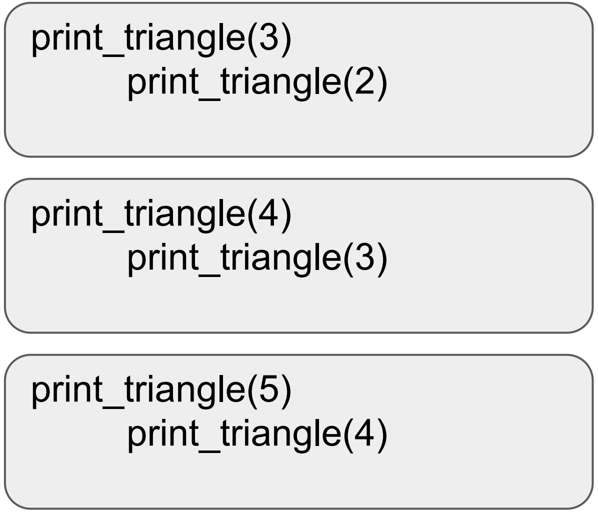
Let’s trace it



```
print_triangle(4)
        print_triangle(3)
```

```
print_triangle(5)
        print_triangle(4)
```

Let’s trace it

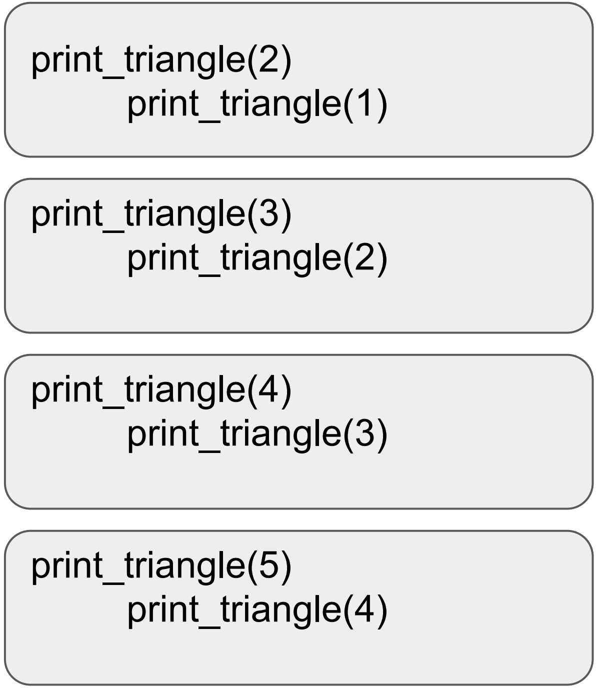


```
print_triangle(3)
        print_triangle(2)
```

```
print_triangle(4)
        print_triangle(3)
```

```
print_triangle(5)
        print_triangle(4)
```

Let’s trace it



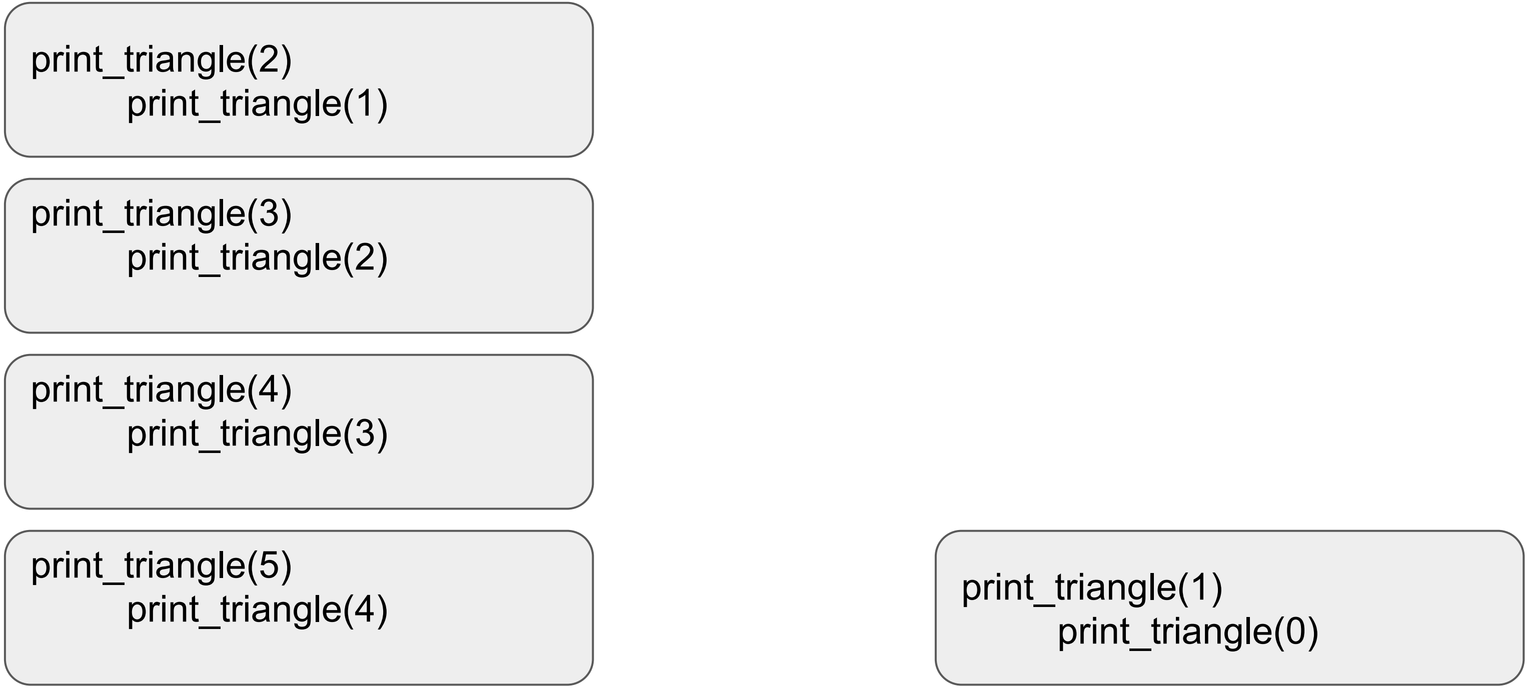
```
print_triangle(2)
        print_triangle(1)
```

```
print_triangle(3)
        print_triangle(2)
```

```
print_triangle(4)
        print_triangle(3)
```

```
print_triangle(5)
        print_triangle(4)
```

Let’s trace it



```
print_triangle(2)
        print_triangle(1)
```

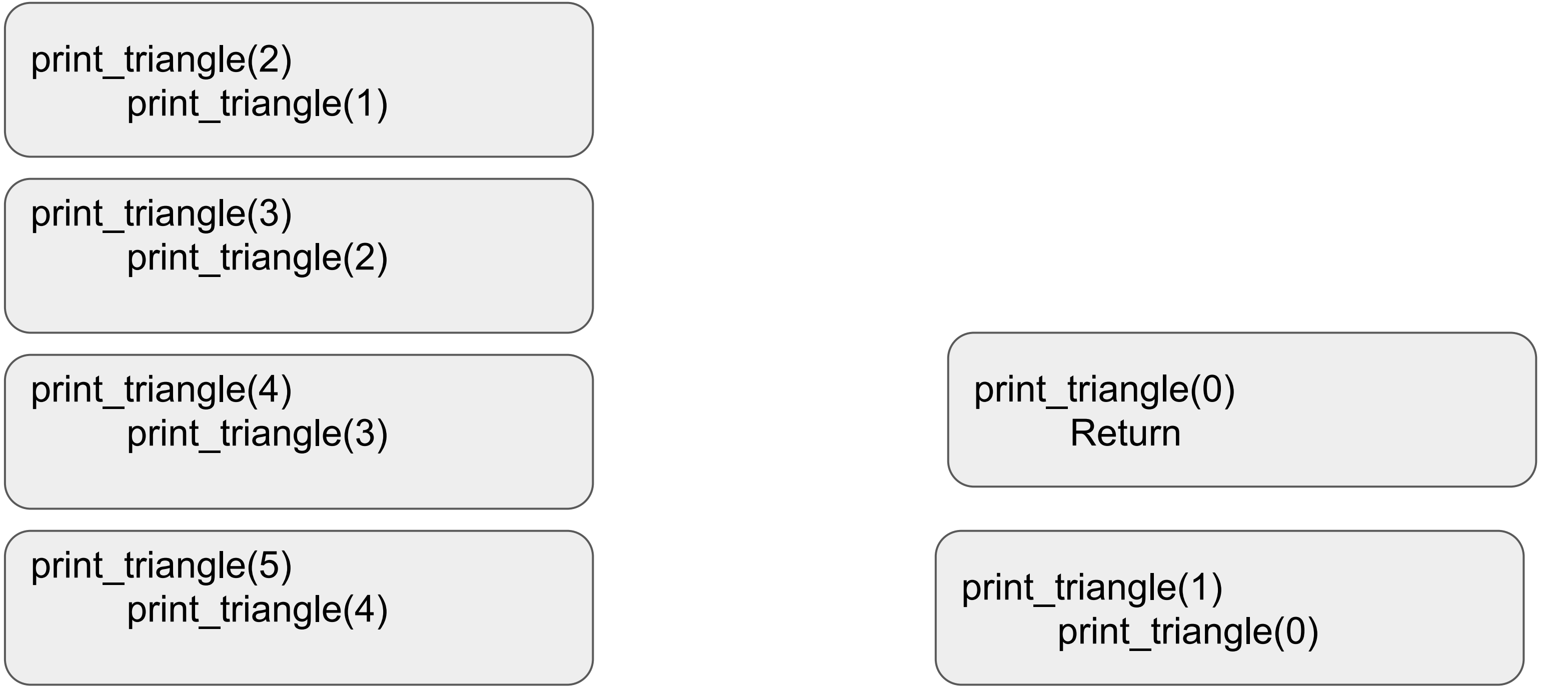
```
print_triangle(3)
        print_triangle(2)
```

```
print_triangle(4)
        print_triangle(3)
```

```
print_triangle(5)
        print_triangle(4)
```

```
print_triangle(1)
        print_triangle(0)
```

Let’s trace it



```
print_triangle(2)
        print_triangle(1)

print_triangle(3)
        print_triangle(2)

print_triangle(4)
        print_triangle(3)

print_triangle(5)
        print_triangle(4)

print_triangle(0)
        Return

print_triangle(1)
        print_triangle(0)
```

Let’s trace it

```
print_triangle(2)
        print_triangle(1)
```

```
print_triangle(3)
        print_triangle(2)
```

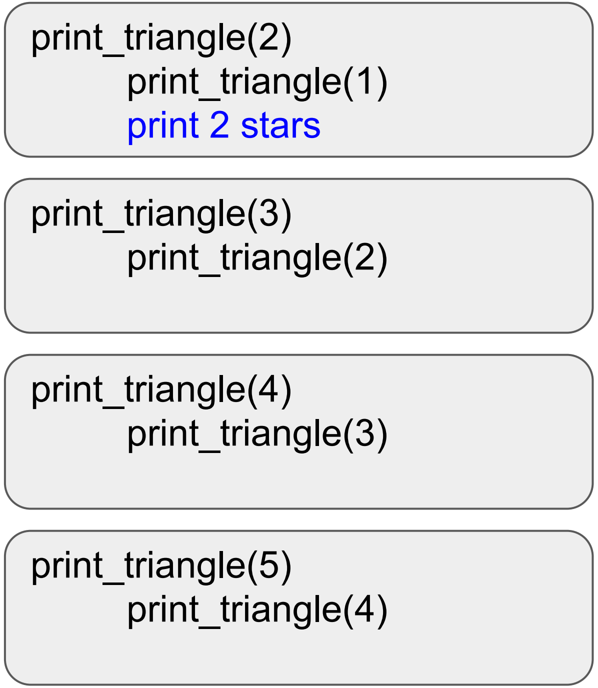
```
print_triangle(4)
        print_triangle(3)
```

```
print_triangle(5)
        print_triangle(4)
```

```
*
```

```
print_triangle(1)
        print_triangle(0)
        print 1 star
```

Let’s trace it



```
print_triangle(2)
        print_triangle(1)
        print 2 stars

print_triangle(3)
        print_triangle(2)

print_triangle(4)
        print_triangle(3)

print_triangle(5)
        print_triangle(4)
```

```
*
**
```

Let’s trace it

```
print_triangle(3)
        print_triangle(2)
        print 3 stars
```

```
print_triangle(4)
        print_triangle(3)
```

```
print_triangle(5)
        print_triangle(4)
```

```
*
**
***
```

Let’s trace it

```
*
**
***
****
```

```
print_triangle(4)
        print_triangle(3)
        print 4 stars
```

```
print_triangle(5)
        print_triangle(4)
```

Let’s trace it

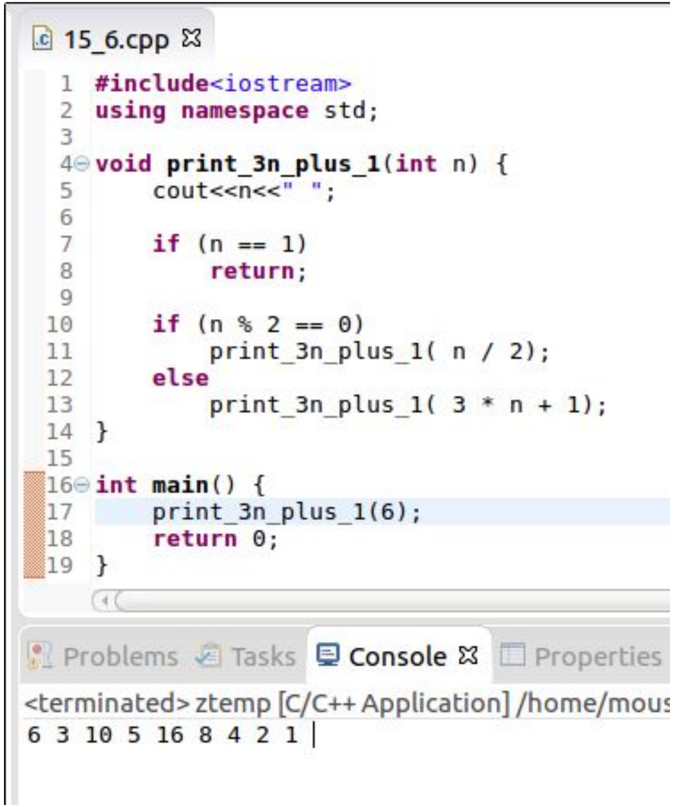
```
*
**
***
****
*****
```

```
print_triangle(5)
    print_triangle(4)
    print 5 stars
```

Print 3n+1 Sequence

- A 3n+1 goes as following
- Start from a number n
- If this number is even, next number in sequence is n / 2
- If this number is odd, next number in sequence is 3 * n + 1
- If this number is 1 = end of sequence
- E.g. Start from 5: **5 16 8 4 2 1**
- E.g. Start from 6: 6 3 10 **5 16 8 4 2 1**
- E.g. Start from 9: 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
- Write a recursive function to print it
    - Stop the video and try

Print 3n+1 Sequence



```
#include<iostream>
using namespace std;

void print_3n_plus_1(int n) {
    cout<<n<<" ";

    if (n == 1)
        return;

    if (n % 2 == 0)
        print_3n_plus_1( n / 2);
    else
        print_3n_plus_1( 3 * n + 1);
}

int main() {
    print_3n_plus_1(6);
    return 0;
}
```

```
<terminated> ztemp [C/C++ Application] /home/mous
6 3 10 5 16 8 4 2 1
```

Let’s trace it

```
print_3n_plus_1(5)
        print 5
        Print_3n_plus_1 (3*5+1)
```

```
print_3n_plus_1(10)
        print 10
        Print_3n_plus_1 (10 / 2)
```

```
print_3n_plus_1(3)
        print 3
        Print_3n_plus_1 (3*3+1)
```

```
print_3n_plus_1(6)
        print 6
        Print_3n_plus_1 (6/2)
```

Let’s trace it

```
print_3n_plus_1(5)
        print 5
        Print_3n_plus_1 (3*5+1)
```

```
print_3n_plus_1(10)
        print 10
        Print_3n_plus_1 (10 / 2)
```

```
print_3n_plus_1(3)
        print 3
        Print_3n_plus_1 (3*3+1)
```

```
print_3n_plus_1(6)
        print 6
        Print_3n_plus_1 (6/2)
```

```
print_3n_plus_1(2)
        print 2
        Print_3n_plus_1 (2/2)
```

```
print_3n_plus_1(4)
        print 4
        Print_3n_plus_1 (4/2)
```

```
print_3n_plus_1(8)
        print 8
        Print_3n_plus_1 (8/2)
```

```
print_3n_plus_1(16)
        print 16
        Print_3n_plus_1 (16/2)
```

```
print_3n_plus_1(1)
        print 1
```

Homework 0:

- Revise & Trace by hand & code all the methods

Homework 1: Length of 3n+1

- Implement 3n+1 function to compute the length of the sequence
- **int** length_3n_plus_1(int n)
- E.g. length_3n_plus_1(6) ⇒ 9

Homework 2: Power function

- int my_pow(int value, int p = 2)
- Return value * value ….. * value p times
- E.g. my_pow(7, 3) = 7 * 7 * 7 = 343
- Note: if p = 0, answer is 1

Homework 3: Array maximum

- int arr_max(int arr[], int len);
- Write a function that computes array maximum
- Input 1, 8, 2, 10, 3 ⇒ 10

Homework 4: Array sum

- Int sum(int arr[], int len);
- Write a function that computes array sum
- Input 1, 8, 2, 10, 3 ⇒ 24

Homework 5: Array average

- double average(int arr[], int len);
- Write a function that computes array average
  - Don’t divide by length in the main
- Input 1, 8, 2, 10, 3 ⇒ 4.8

Homework 6: Array Increment

- void array_increment(int arr[], int len)
- The function increments each arr[i] with i
- E.g. for input
  - [1, 2, 5, 9] it be [1+0, 2+1, 5+2, 9+3]
  - 1 8 2 10 3 ⇒ 1 9 4 13 7

Homework 7: Array Accumulation

- Given an array we want to accumulate it as following:
  - Input 1 2 3 4 5 6
  - Output array
    - 1, 1+2, 1+2+3, 1+2+3+4, 1+2+3+4+5, 1+2+3+4+5+6
    - 1, 3, 6, 10, 15, 21
  - That is return arr[i] arr[i] = sum of all numbers from 0 to i
- void accumulate_arr(int arr[], int len);
  - Input 1 8 2 10 3 ⇒ 1 9 11 21 24

Homework 8: Left-Max

- Given array, change each element at position i to be the maximum of numbers from 0 to index i
- E.g. input 1 3 5 7 4 2 ⇒ [1, 3, 5, 7, 7, 7]
- Void left_max(int arr[], int len);

Homework 9: Right-Max

- Given array, change each element at position i to be the maximum of numbers from index i to end of array
- E.g. input 1 3 5 7 4 2 ⇒ [7, 7, 7, 7, 4, 2]
- Void left_max(int arr[], int len, int start_position = 0);

Homework 10: Suffix Sum

- Write a function that sums only the last N elements in an array.
- Define its signature
- Input [1, 3, 4, 6, 7], 3 ⇒ 17 (4+6+7)

Homework 11: Prefix Sum

- Write a function that sums only the first N elements in an array.
- Define its signature
- Input [1, 3, 4, 6, 7], 3 ⇒ 8 (1+3+4)

Homework 12: Is Palindrome

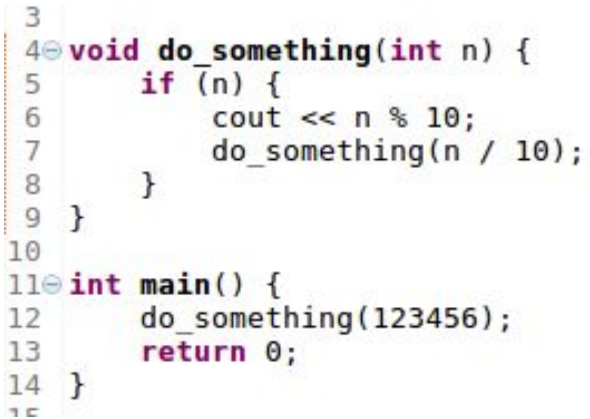
- Implement a function that decides if array is palindrome or not
- Define its signature

Homework 13: Is prefix

- bool is_prefix(string main, string prefix, int start_pos = 0)
- E.g. is_prefix(“abcdefgh”, “abcd”) ⇒ true
- E.g. is_prefix(“abcdefgh”, “”) ⇒ true
- E.g. is_prefix(“abcdefgh”, “abd”) ⇒ false

Homework 14: ??? Number

- Without running code on the right
    - Trace by hand: What does this method do?
    - What happens if we swapped lines 6 & 7?



```
 3
 4 void do_something(int n) {
 5     if (n) {
 6         cout << n % 10;
 7         do_something(n / 10);
 8     }
 9 }
10
11 int main() {
12     do_something(123456);
13     return 0;
14 }
```

Homework 15: Count primes

- Int count_primes(int start, int end);
  - Compute how many primes between start & end, inclusive indices
- Don’t use loops at all
- Input
  - 10 20 ⇒ 4
  - 10 200 ⇒ 42
- Can u compute answer for [10, 5000000]?

Homework 16: Grid Sum

- Given a 2D array of numbers, all of them are positive distinct. Robot start from (0, 0). It can move to the right or left or diagonal. It will select one direction: the maximum. Print the total path sum of this robot
  - int path_sum(int grid[100][100], int row, int col, int ROWS, int COLS)
- Input
  - 3 3
  - 1 7 8
  - 2 10 11
  - 20 5 9
- Output: 31 (from 1 + 10 + 11 + 9)
  - Robot start at (0, 0). 3 possible values (2, 7, 10). Max 10, so go to this cell
  - Then 3 possible values (5, 9, 11). Go to 11. Then only 9 available

Homework 17: Fibonacci

- Implement fibonacci: Int fibonacci(int n)
  - Recall fibonacci sequence: 1 1 2 3 **5 8 13** 21 35
  - E.g. fibonacci(6) = 13
  - Recall that: fibonacci(n) = fibonacci(n-1) + fibonacci(n-2). E.g. fib(6) = fib(5)+fib(4) =13
    - So it calls 2 subproblems of its type
- Can u compute fibonacci(40)? fibonacci(50)? Why? Any work around? Hint: Array

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً