FUNDAMENTALS OF STRUCTURED PROGRAMMING

Lecture 11

Function IV: Recursion

Course Coordinator: Prof. Zaki Taha Fayed

Presented by: Dr. Sally Saad

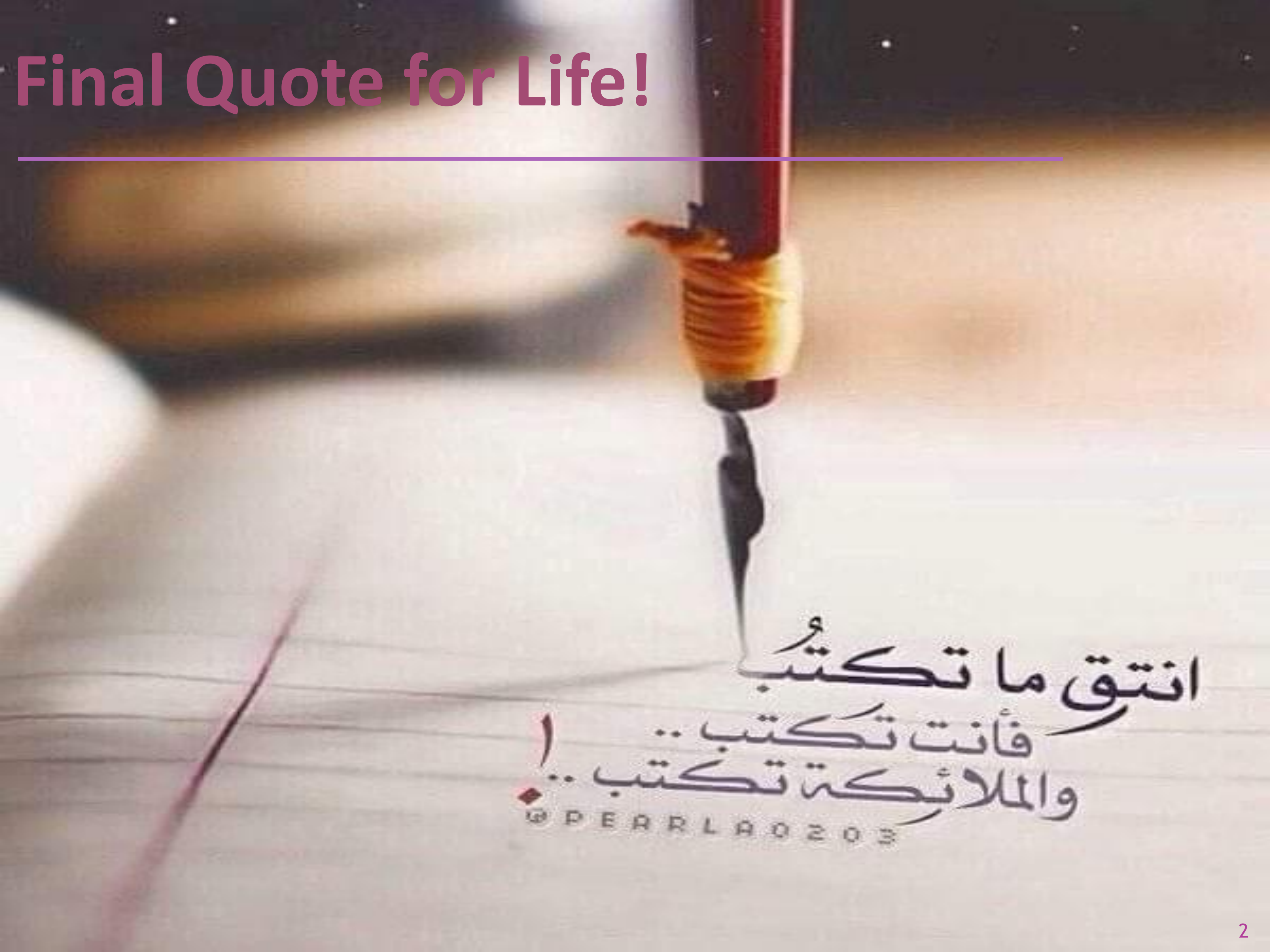
SallySaad@gmail.com

DropBox folder link

https://www.dropbox.com/sh/85vnrgkfqgrzhwn/AABdwKLJZqZs26a7u-y0AFwia?dl=0

Credits to Dr.Salma Hamdy for content preparation

Final Quote for Life!



Contents

1. Programmer-defined Functions – (cont.)

(12) Recursion

- In C++, you can call a function from within another function.
- The main function may call other functions.
- A sorting function may call a swap function.

1. Programmer-defined Functions – (cont.)

(12) Recursion

- Functions are used to do subtasks, and sometimes it turns out that one subtask is a smaller example of another.
- Search array for a value
  - Subtask 1: search 1st half of array
  - Subtask 2: search 2nd half of array
- The subtasks of searching the array’s halves are smaller versions of the original task.

1. Programmer-defined Functions – (cont.)

(12) Recursion

- **Whenever one subtask is a smaller version of the original task, you can solve the original task using a recursive function.**
- ***A recursive function is a function that calls itself.***
- **Recursion is a useful programming techniques and most high-level languages allow it.**
- **But it has limitations as you'll see.**

1. Programmer-defined Functions – (cont.)

(12) Recursion-Most Known Example

- To find the factorial of a number N.
  - If N is 1, the factorial is 1.
  - Otherwise, find the factorial of N-1 and multiply it by N.
  - fact(4) = 4 x fact(3)

    = 4 x 3 x fact(2)

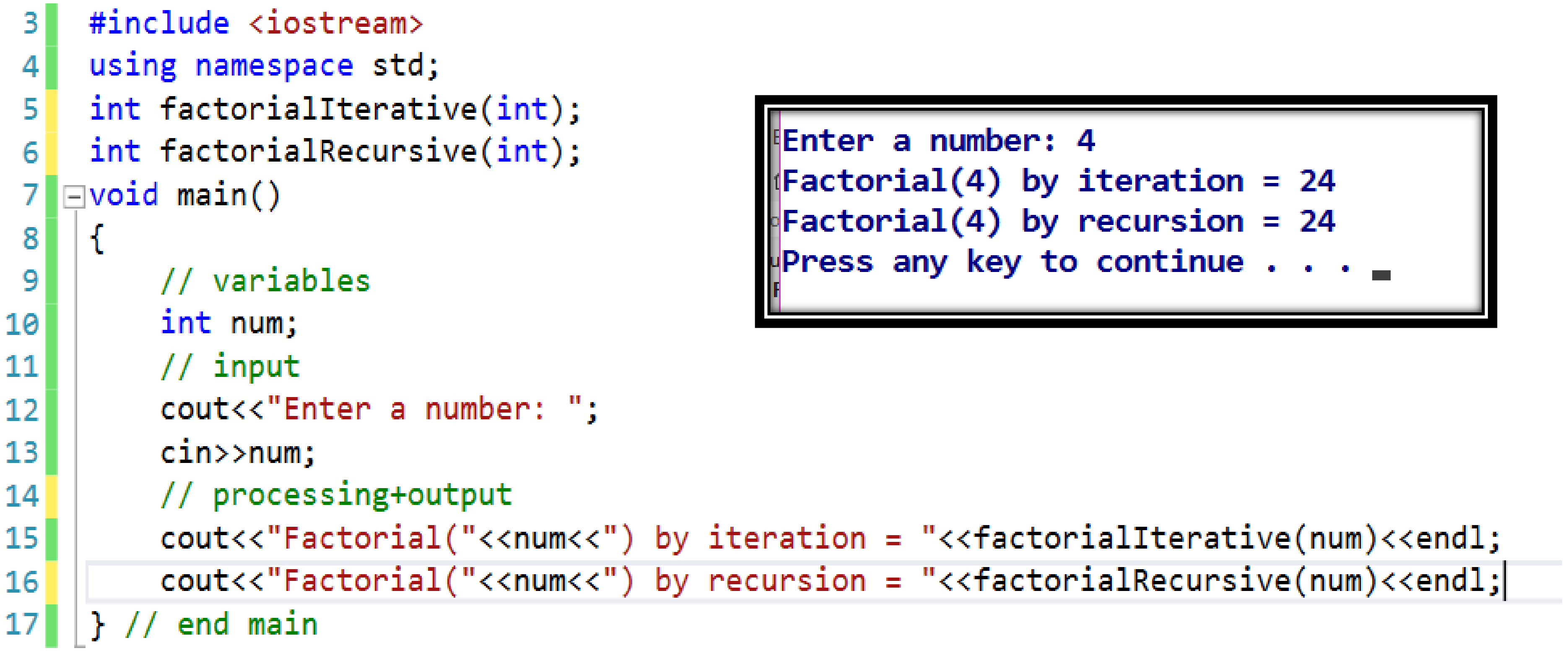
    = 4 x 3 x 2x fact(1)

    = 4 x 3 x 2 x 1

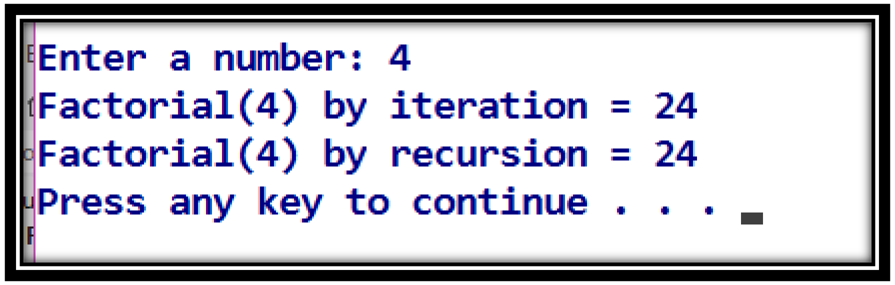
    =24

Lec11Ex1.cpp

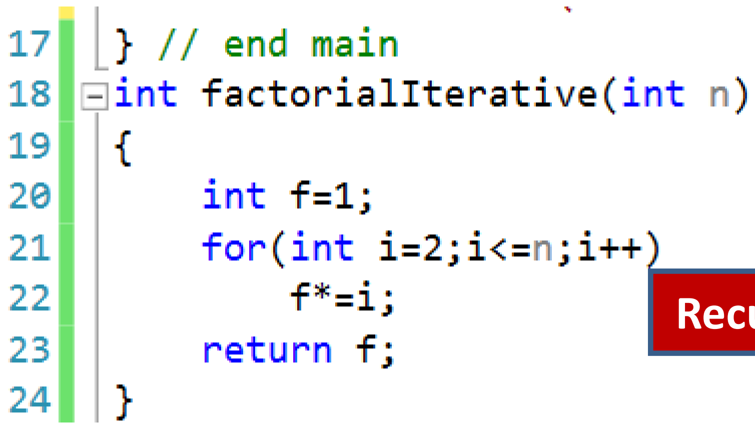
Example 1: Factorial of a number using iterative and recursive methods



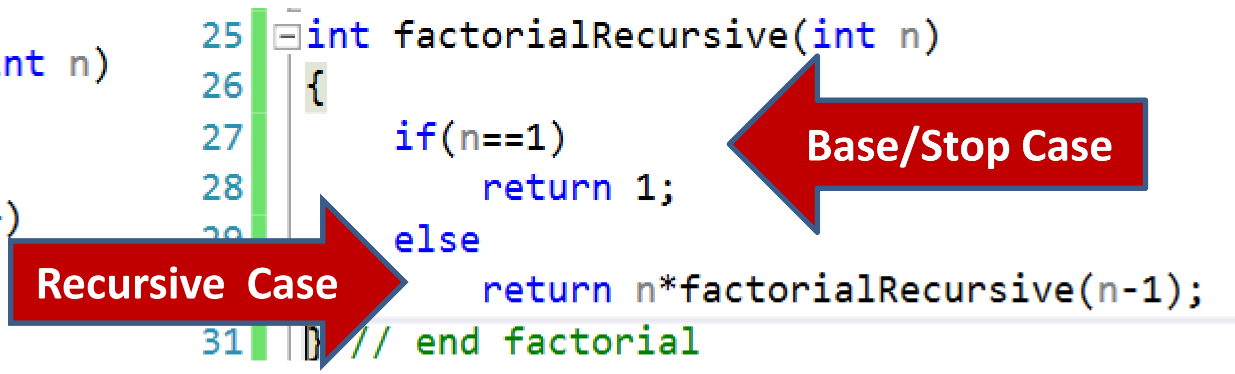
```cpp
#include <iostream>
using namespace std;
int factorialIterative(int);
int factorialRecursive(int);
void main()
{
    // variables
    int num;
    // input
    cout<<"Enter a number: ";
    cin>>num;
    // processing+output
    cout<<"Factorial("<<num<<") by iteration = "<<factorialIterative(num)<<endl;
    cout<<"Factorial("<<num<<") by recursion = "<<factorialRecursive(num)<<endl;
} // end main
```



```
Enter a number: 4
Factorial(4) by iteration = 24
Factorial(4) by recursion = 24
Press any key to continue . . .
```



```cpp
} // end main
int factorialIterative(int n)
{
    int f=1;
    for(int i=2;i<=n;i++)
        f*=i;
    return f;
}
```



```cpp
int factorialRecursive(int n)
{
    if(n==1)
        return 1;
    else
        return n*factorialRecursive(n-1);
} // end factorial
```

1. Programmer-defined Functions – (cont.)

(12) Recursion-Tracing a Recursive Call

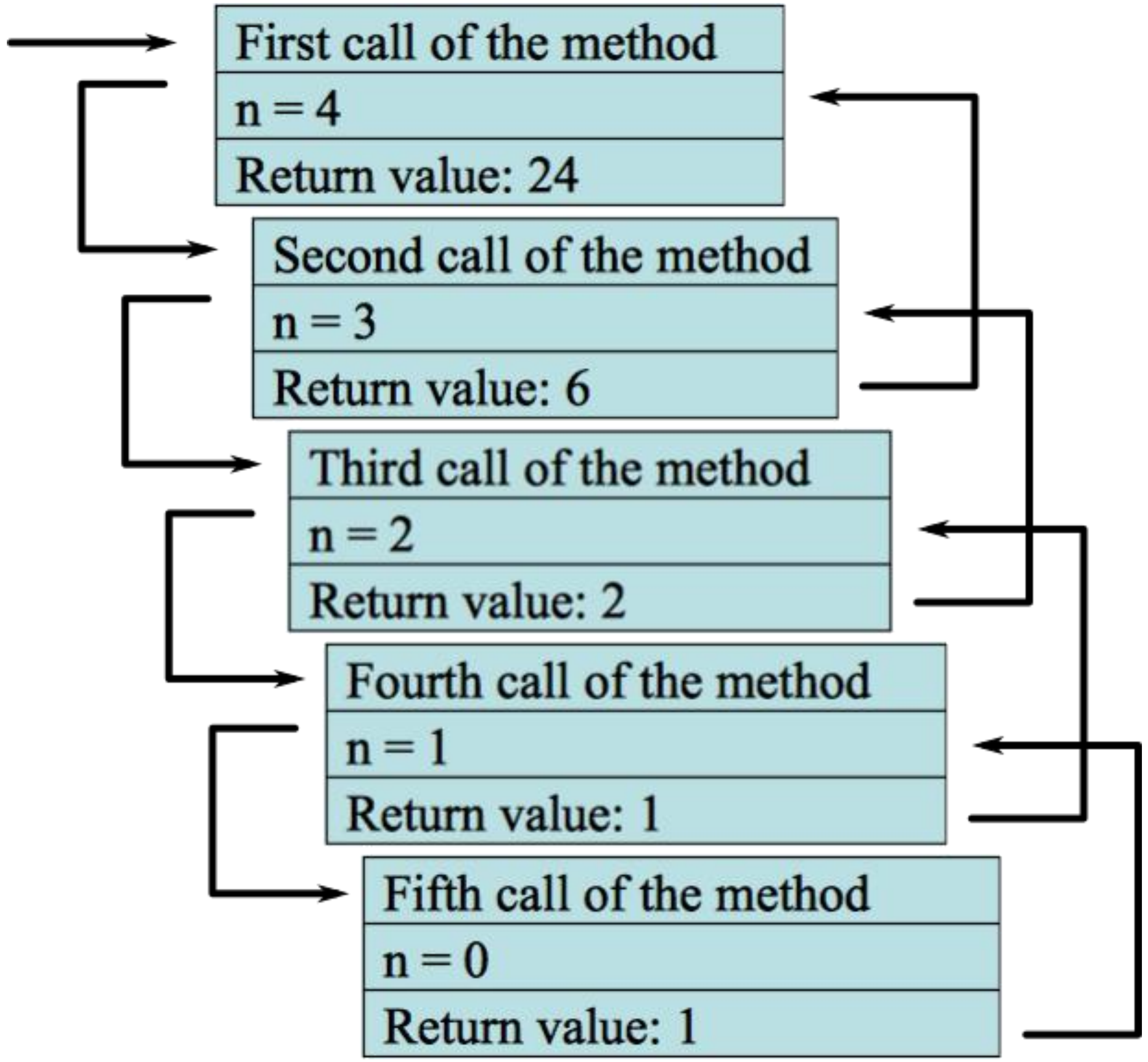
- Stops current function.
- Must know results of new recursive call before proceeding.
- Saves all information needed for current call to be used later.
- Proceeds with evaluation of new recursive call.
- When THAT call completes, returns to "outer“ computation

1. Programmer-defined Functions – (cont.)

(12) Recursion

Tracing a Recursive Call

- **Suspended/resumed computations.**
- **Note that there must be one piece of code that doesn't depend on recursion to terminate the call(s) → <u>base case/stopping case.</u>**
- **Infinite recursion… very bad!** ✗


First call of the method
n = 4
Return value: 24
Second call of the method
n = 3
Return value: 6
Third call of the method
n = 2
Return value: 2
Fourth call of the method
n = 1
Return value: 1
Fifth call of the method
n = 0
Return value: 1

1. Programmer-defined Functions – (cont.)

(12) Recursion

Outline of successful recursive function:

- **One or more cases where function accomplishes its task by making one or more recursive calls to solve smaller versions of original task → Called *“recursive case(s)”.***
- **One or more cases where function accomplishes its task without recursive calls → *Called "base case(s)" or stopping case(s).***

1. Programmer-defined Functions – (cont.)

(12) Recursion

Recursion Versus Iteration

- **Recursion is not always "necessary“.**
- **Not even allowed in some languages.**
- **Any task accomplished with recursion can also be done without it.**
  - **Non-recursive: iterative, using loops.**
- **Recursive:**
  - **Runs slower, uses more storage.**
  - **Elegant solution; less coding.**

1. Programmer-defined Functions – (cont.)

(12) Recursion

Example 2 – Lec11Ex2.cpp

- **Power function with two methods.**

1. Programmer-defined Functions – (cont.)

(12) Recursion

Example 2

Power function

```
double result = 1;
for(int i=0; i<expo; i++)
    result = result*base;
return result;
```

```
if(expo==0)
    return 1;
else if(expo==1)
    return base;
else
    return base*my_pow(base, expo-1);
```

Exercise: optimize the code

1. Programmer-defined Functions – (cont.)

(12) Recursion

Example 3 – Lec6Ex3.cpp

- To display the digits of a number separately.

```
1234
1
2
3
4
```

Three methods

1. Programmer-defined Functions – (cont.)

(12) Recursion-Example 3 – Lec11Ex3.cpp

- **To display the digits of a number separately.**
  - If N is less than 10, display it. **(BASE CASE)**
  - Otherwise, display the digits separately without the last one, then display the last one. **(RECURSIVE CASE)**

```
1234
1
2
3
4
```

TIPS ON EXAMS AND PROJECT DELIVERY

4. Practical Exam Tips *(Sat 28/4-Thurs 3/5)*

- Come on time as the schedule that will be announced soon, late students will be punished.
- Bring your FCIS card/ID with photo.
- Laptops are allowed but will be interchanged with your colleague.
- All Questions should be implemented using FUNCTIONS.
- All studied practical content of lecture and lab is required in Practical Exam except File Streams and Recursion.
- Cheaters will be given both BIG ZEROES.

4. Project Delivery Tips *(Sun 6/5- Mon 7/5)*

- Bring your **ID**.
- Come on Scheduled **time**.
- Understand **all the project code**.
- **All the team members** must attend discussion. *(Around 20 mints)*
- **Individual Grading.**
- Get the **source code** on many resources (flash memory, cd, email, laptop..etc)
- IF you have any **documentation**, like design or user manual, bring it with you for extra bonus.
- You will be discussed by your **mentor**.
- **He/She might take the code** for later reference.
- **Best Projects** will be **rewarded**.
- **Copied Projects** will take **ZERO**.
- **InActive** Members will take **ZERO** too.

GOOD LUCK ☺

4. Final Exam Tips *(31/5/18)*

Required Content:

- All lectures and labs including built-in functions and recursive functions except file stream.

Questions may include:

1. MCQ – T/F.
2. Code Tracing.
3. Write a C++ program from scratch.
4. Write a piece of code (function or so).
5. Pseudocode trace and/or conversion to C++ code.
6. Top Down design.
7. Error Types and Code Correction.

4. Final Exam Tips *(31/5/18)*

Remember:

- **Local vs. global data.**
- **Parameters vs. arguments.**
- **Calling by value vs. by reference.**
- **Recursion vs. iteration.**
- **Dynamic vs. static arrays.**
- **Dangling pointers.**
- **Lecture 1 theoretical part.**

Assess our Course before it Assesses You ☺

<u>For a couple of minutes</u>

- In a tip of paper kindly write down any recommendation to enhance our course.

It will be nice to hear about the positives you learned too .

- Any Positive or negative **<u>POLITE</u>** **<u>comment</u>** about myself or the TAs is accepted.
- No names required.

Thank
you.