Week 7

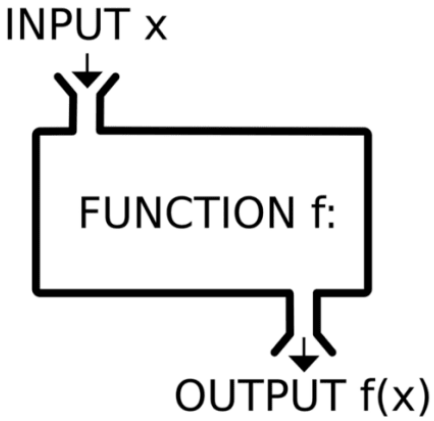
Hari Sethuraman

By the end of today

- You should have a solid understanding of
  - Functions
  - The stack
  - Recursion

What are functions

- Functions are a unit of a program or a building block that take in an input and return a processed value
- Why use functions?
  - Functions allow code to be more reusable and more organized
    - Makes debugging much easier



How to declare functions in C

- The input to a function is also called a ‘parameter’
- A function follows certain predefined operations on the parameter
  - You pass in an input value and the function substitutes the parameter with the input value

```
int function(int x, int y)
{
    …
    return …
}
```

Data type of return value

Name of function

Parameters separated by comma

Code goes between curly braces

Always declare functions before you call them in the code!

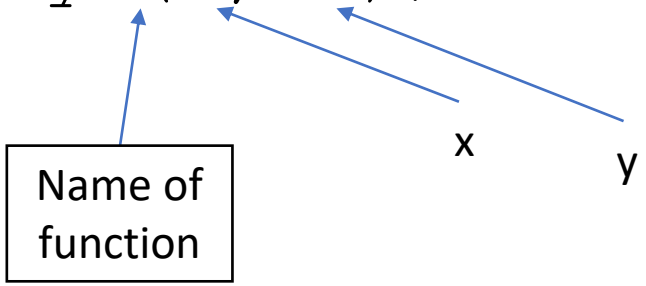
Take two numbers and multiply them

```
// take two numbers as input and return their
product
int multiply (int x, int y)
{
    int product = x * y;
    return product;
}
```

How to call a function

- `Name (parameter1, parameter 2, …);` // parameters in order of declaration!

```
int z = multiply (5, 3);
```



Program

- Get all elements of an array
  - Print all the elements of the array
- Get all elements of another array
  - Print all the elements of the array

Implementation 1

- Loop 1 to get all the input values
- Loop 2 to print the array
- Loop 3 to get all input values
- Loop 4 to print the array

Avoid repetitive code

Implementation 2

- Use functions to eliminate the using of the loops

The stack

- The memory used by a program consists of two ‘areas’: the Stack and the Heap
- Is where your computer stores the functions and variables that are called or defined when your program is running
- Works like a stack of trays in a cafeteria. Follows LIFO (Last in first out)
- Every time a new function is called, a new layer is added on the stack which stores the information of the function.
  - All functions below an executing function are frozen (have paused executing)
  - When a function finishes executing it is removed from the stack

printArray

getElements

main

5

4

Program for stack

```
int f3(int n)
{
    int ans = n * n;
} return ans;

int f2(int n)
{
    int ans = f3(n) * 2;
    return ans;
}

int f1(int n)
{
    int ans = f2(n) + 7;
    return ans;
}
int main()
{
   int n = 0;
   scanf(“%d”, &n);
   int a  = f1(n)
   printf(“%d\n”, a);

}
```

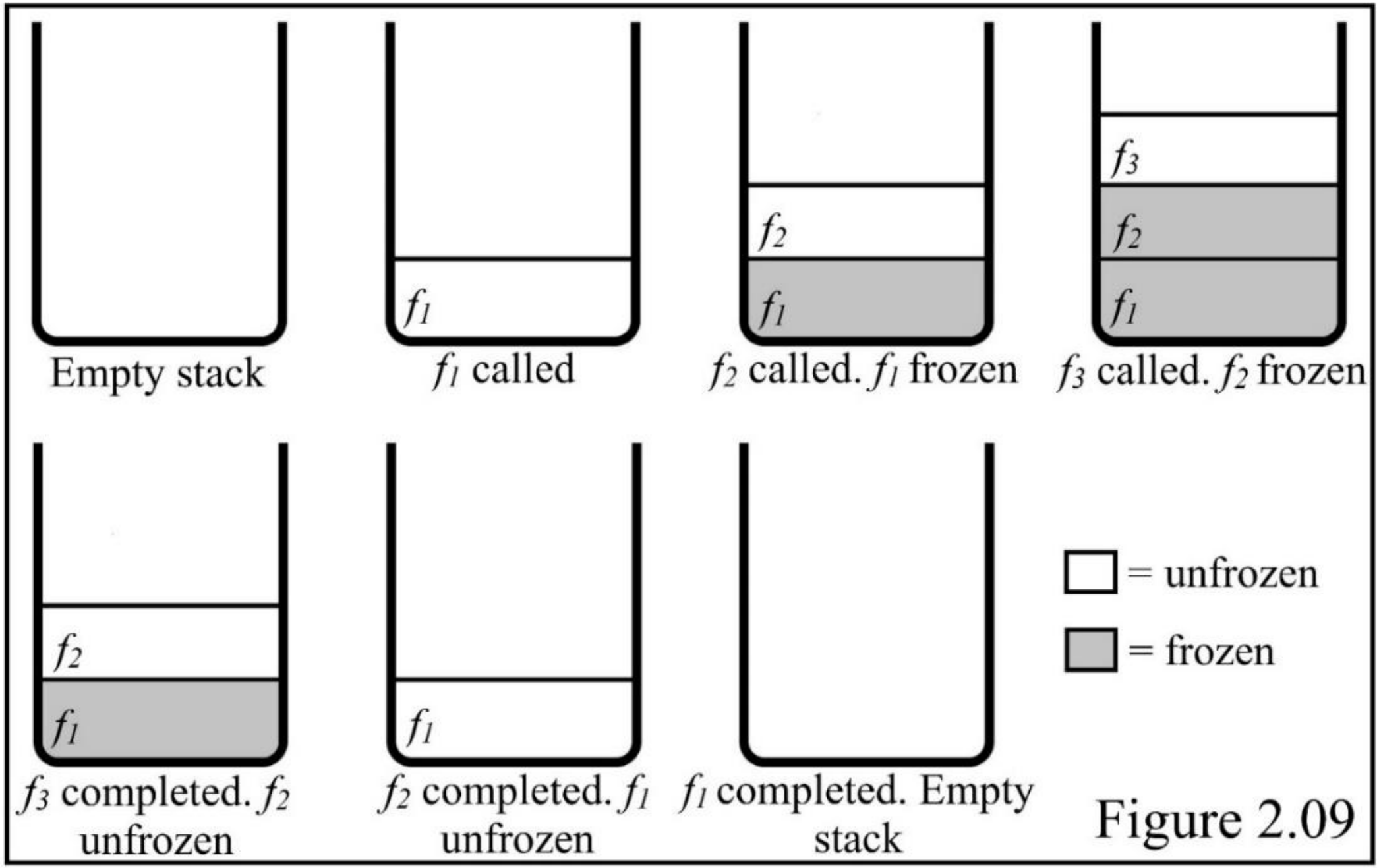

Empty stack
f_1
f_1 called
f_2
f_1
f_2 called. f_1 frozen
f_3
f_2
f_1
f_3 called. f_2 frozen
f_2
f_1
f_3 completed. f_2 unfrozen
f_1
f_2 completed. f_1 unfrozen
f_1 completed. Empty stack
= unfrozen
= frozen

Figure 2.09

Recursion

- Recursion occurs when a function calls itself
- Consists of a call and a base case
  - The call is where the function calls itself
  - The base case is a condition that prevents the function from calling itself forever

Sigma

- A function *sigma* takes in a positive integer *n* and returns the sum of all integers up till and including *n*.
- Example: *sigma(5)* = 5
- *Sigma(4) = 10*
- *Sigma(7) = ?*
- *Sigma(9) = ?*

In code

```
int sigma(int n)
{
    if (n == 1)
        return 1;
    int sum = n + sigma(n - 1);
    return sum;
}
```

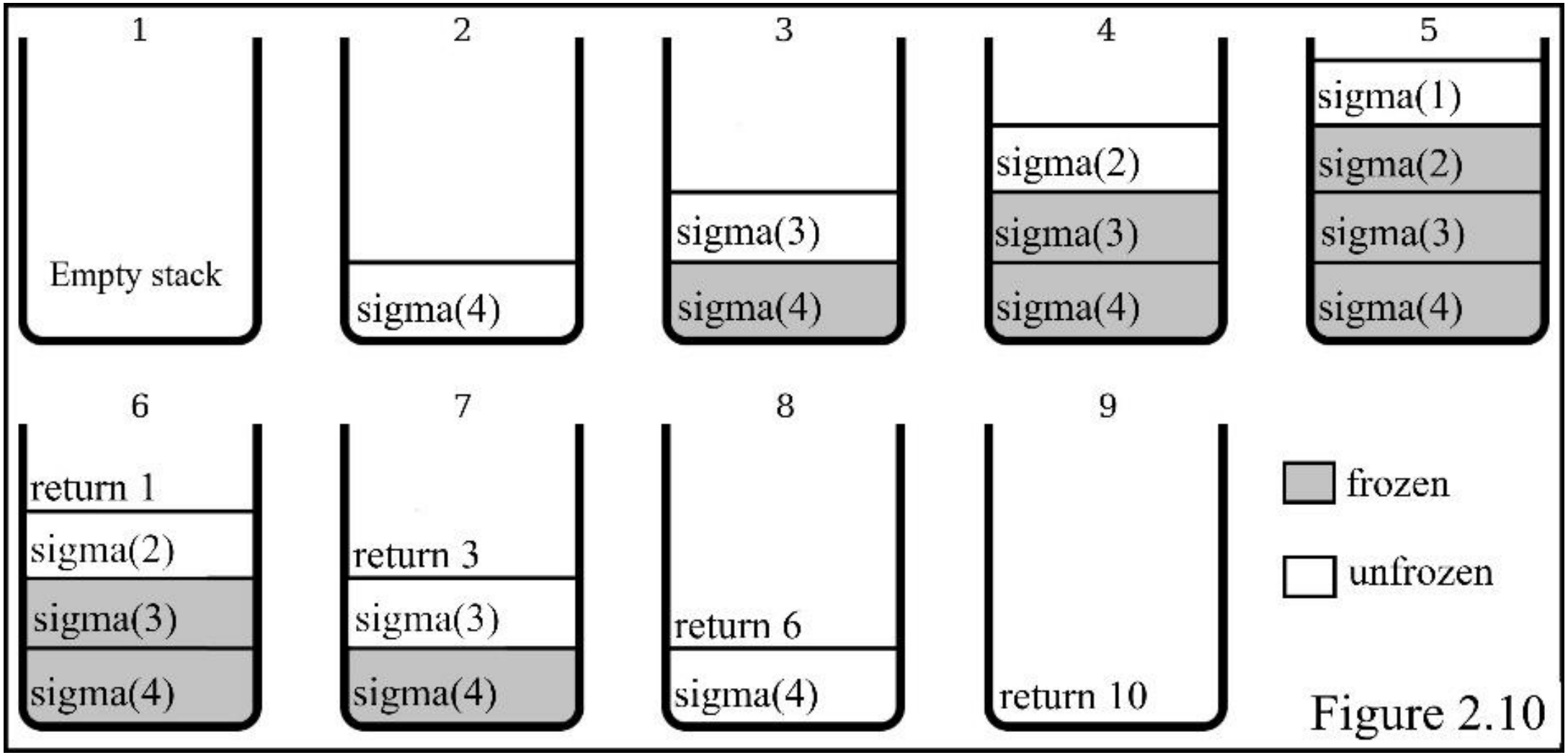

1
Empty stack
2
sigma(4)
3
sigma(3)
sigma(4)
4
sigma(2)
sigma(3)
sigma(4)
5
sigma(1)
sigma(2)
sigma(3)
sigma(4)
6
return 1
sigma(2)
sigma(3)
sigma(4)
7
return 3
sigma(3)
sigma(4)
8
return 6
sigma(4)
9
return 10
frozen
unfrozen

Figure 2.10

Homework

- Factorial
  - You are given a positive integer *n*: return the product of all positive integers up till and including *n*.