CS1002 – Programming Fundamentals

Lecture # 26
Tuesday, November 29, 2022
FALL 2022
FAST – NUCES, Faisalabad Campus

Rizwan Ul Haq

Recursion

- **Background**
  - We have been using iterations to solve repetitive tasks
  - We have another alternate to iteration known as **Recursion**
- **Definition**
  - Process of solving a problem by reducing it to smaller versions of itself

Recursion in mathematics

- In mathematics recursion is defined as

$$0! = 1 \qquad 1.0$$

$$n! = n * (n-1)!\, if\ n > 0 \qquad 2.0$$

- Above 0! is defined to be 1, and if n is an integer greater than 0, we first calculate $(n-1)!$ and the multiply it with n.
- To find $(n-1)!$ We apply the definition again. If $(n-1)!$ is greater than 0 then equation 2.0 otherwise 1.0

Recursion in mathematics

$$0! = 1 \qquad 1.0$$
$$n! = n * (n-1)! \; if \; n > 0 \qquad 2.0$$

Example: 3!

Here $n = 3$ and hence $n > 0$ we use equation 2.0

$$3! = 3 * 2!$$

Here $n = 2$ now we find 2!

$$2! = 2 * 1!$$

Here $n = 1$ now we find 1!

$$1! = 1 * 0!$$

Now as $n = 0$ we use equation 1.0 to find 0! Which is 1. Now substituting 0! into 1! gives $1! = 1$.

This gives $2! = 2 * 1! = 2 * 1 = 2$, which in turn, gives

$3! = 3 * 2! = 3 * 2 = 6$

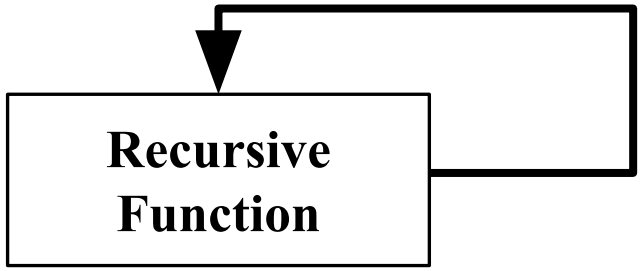
Recursion

$$0! = 1 \qquad 1.0$$
$$n! = n * (n-1)! \; if \; n > 0 \qquad 2.0$$

- The definition of factorial given in 1.0 and 2.0 are called **recursive definition**
- Equation 1.0 is called **Base case**
  - Every recursive definition must have one or more base cases
  - The base case stops the recursion
- Equation 2.0 is called **general case**
  - The general case must eventually be reduced to base case

Recursion in computer science

- An algorithm that finds the solution to a problem by reducing it to smaller versions of the problem itself is called **Recursive Algorithm**
  - Must have at-least one base case
  - General case must reduce to a base case
- A function that calls itself is called a **Recursive Function**.
  - The body of the function contains a statement that calls itself before completing the current call

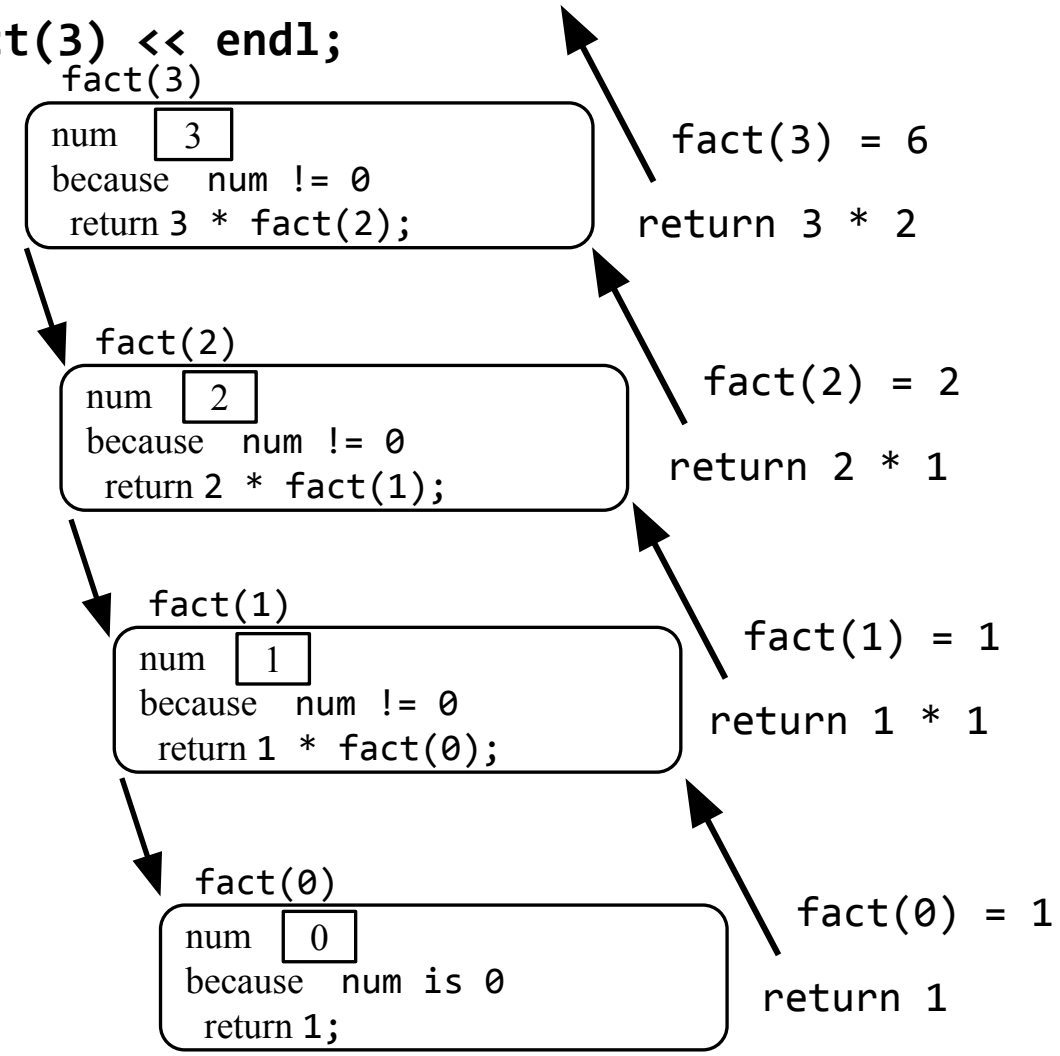


Factorial using recursion

```
int fact(int num)
{
    if (num == 0)
        return 1;
    else
        return num * fact(num - 1);
}
```

Execution Trace for fact(3)

```
cout << fact(3) << endl;
```



Recursive Function

- Logically, you can think of a recursive function as having an unlimited number of copies of itself
- Every recursive call, has its own code and its own set of parameters and local variables
- After completing a particular recursive call, control goes back to the calling environment, which is the previous call
- The current (recursive) call must execute completely before control goes back to the previous call
- The execution in the previous call begins from the point immediately following the recursive call
- Recursive function in which the last statement executed is the recursive call is called a **tail recursive function.** E.g. fact()

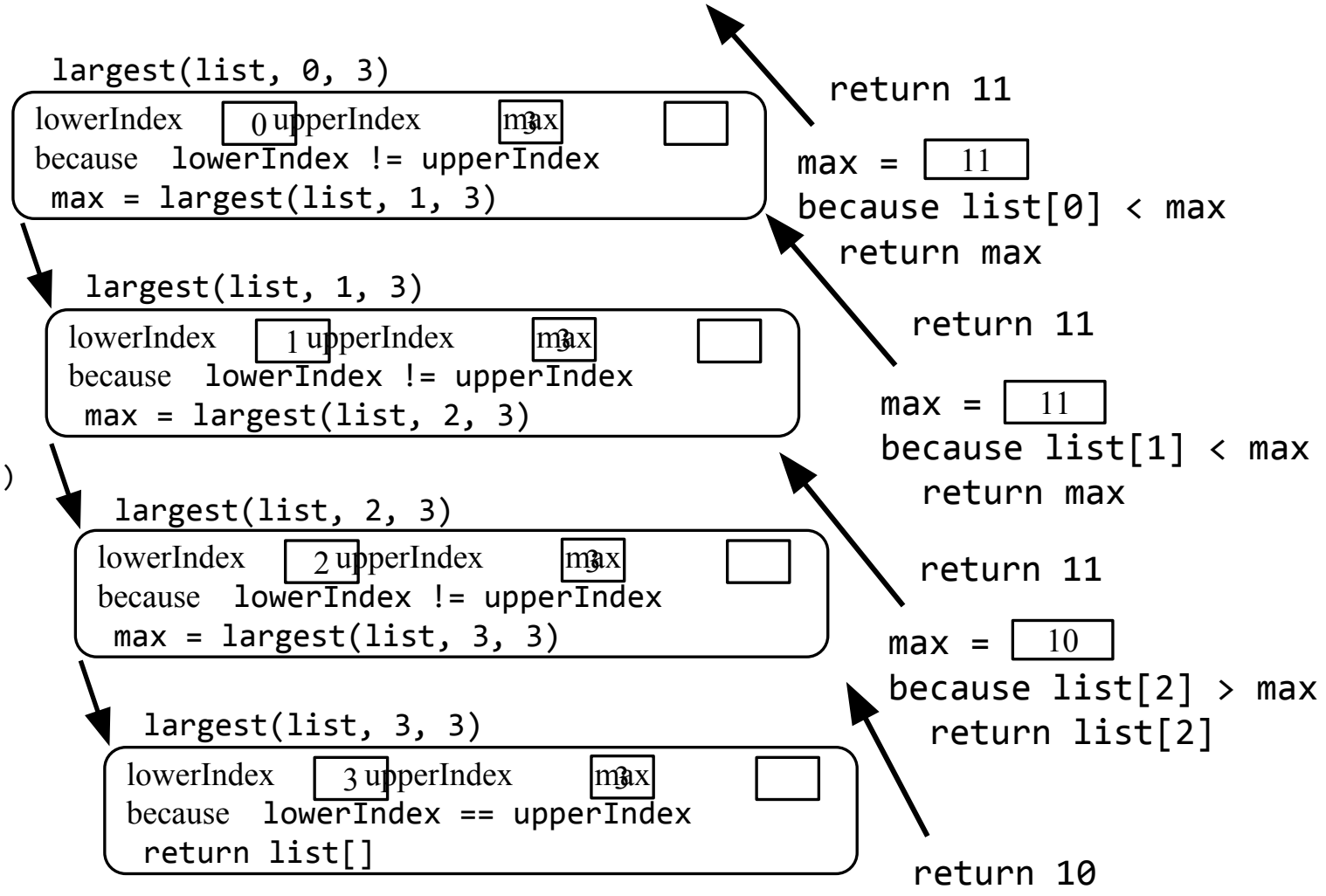
Example Recursion with arrays

```
int largest(const int list[], int lowerIndex, int upperIndex)
{
    int max;
    if (lowerIndex == upperIndex)
        return list[lowerIndex];
    else
    {
        max = largest(list, lowerIndex + 1, upperIndex);
        if (list[lowerIndex] >= max)
            return list[lowerIndex];
        else
            return max;
    }
}
```

Execution Trace for fact(3)

```
Int list[4] = {6,9,11,10};
cout << largest(list,0,3) << endl;
```

```
int largest(const int list[], int lowerIndex, int
                                                upperIndex)
{
   int max;
   if (lowerIndex == upperIndex)
     return list[lowerIndex];
   else
   {
     max = largest(list, lowerIndex + 1, upperIndex);
     if (list[lowerIndex] >= max)
        return list[lowerIndex];
     else
        return max;
   }
}
```



Pros and Cons of Recursion

- Pros
  - Program logic looks much cleaner
  - Reduces lines of code
  - Good to solve recursive problems
  - Good to implement many data structures
  - Remembers previous state automatically
- Cons
  - Uses too much of memory
  - Recursion is slower than iteration normally

Questions