Data Structures Lab 6

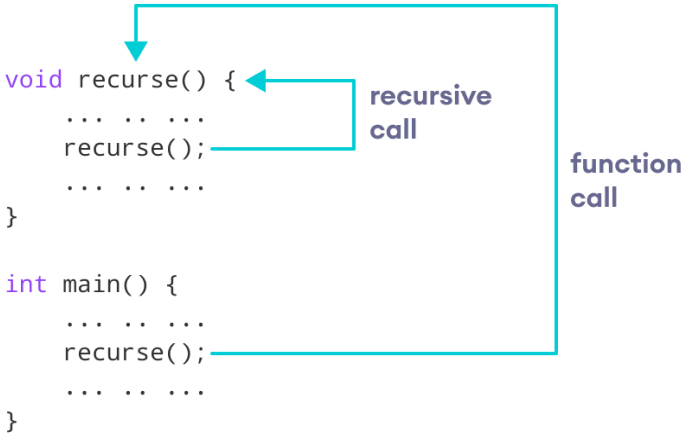
Course: Data Structures (CL2001) **Semester:** Spring 2024
Instructor: Bushra Sattar **T.A:** M. Anas

Note:

- Lab manual cover following below recursion topics
  {Recursion in detail and Backtracking}
- Maintain discipline during the lab.
- Just raise hand if you have any problem.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session.

What is Recursion?

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH).



Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.

Recursion may be a bit difficult to understand. The best way to figure out how it works is to experiment with it.

A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.

Recursion Example

Adding two numbers together is easy to do, but adding a range of numbers is more complicated. In the following example, recursion is used to add a range of numbers together by breaking it down into the simple task of adding two numbers:

```
public class Main {
  public static void main(String[] args) {
    int result = sum(10);
    System.out.println(result);
  }
  public static int sum(int k) {
    if (k > 0) {
      return k + sum(k - 1);
    } else {
      return 0;
    }
  }
}
```

Example Explained

When the sum() function is called, it adds parameter k to the sum of all numbers smaller than k and returns the result. When k becomes 0, the function just returns 0. When running, the program follows these steps:

10 + sum(9)

10 + (9 + sum(8))

10 + (9 + (8 + sum(7)))

...

10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + sum(0)

10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0
Since the function does not call itself when k is 0, the program stops there and returns the result.

Halting Condition

Just as loops can run into the problem of infinite looping, recursive functions can run into the problem of infinite recursion. Infinite recursion is when the function never stops calling itself. Every recursive function should have a halting condition, which is the condition where the function stops calling itself. In the previous example, the halting condition is when the parameter k becomes 0.

It is helpful to see a variety of different examples to better understand the concept.

Java Recursion Example 2: Factorial Number

```
public class RecursionExample3 {
   static int factorial(int n){
        if (n == 1)
          return 1;
        else
          return(n * factorial(n-1));
   }

public static void main(String[] args) {
System.out.println("Factorial of 5 is: "+factorial(5));
}
}
```

Output:

```
Factorial of 5 is: 120
```

Working of above program:

factorial(5)

factorial(4)

factorial(3)

factorial(2)

factorial(1)

return 1

return 2*1 = 2

return 3*2 = 6

return 4*6 = 24

return 5*24 = 120

In this example, the function is calculating the factorial of five. The halting condition for this recursive function is when n is equal 1:

Example 3: Factorial of 4

```
class Factorial {

    static int factorial( int n ) {
        if (n != 0)  // termination condition
            return n * factorial(n-1); // recursive call
        else
            return 1;
    }

    public static void main(String[] args) {
        int number = 4, result;
        result = factorial(number);
        System.out.println(number + " factorial = " + result);
    }
}
```

```
4 factorial = 24
```

In the above example, we have a method named factorial(). The factorial() is called from the main() method. with the number variable passed as an argument.

Here, notice the statement,

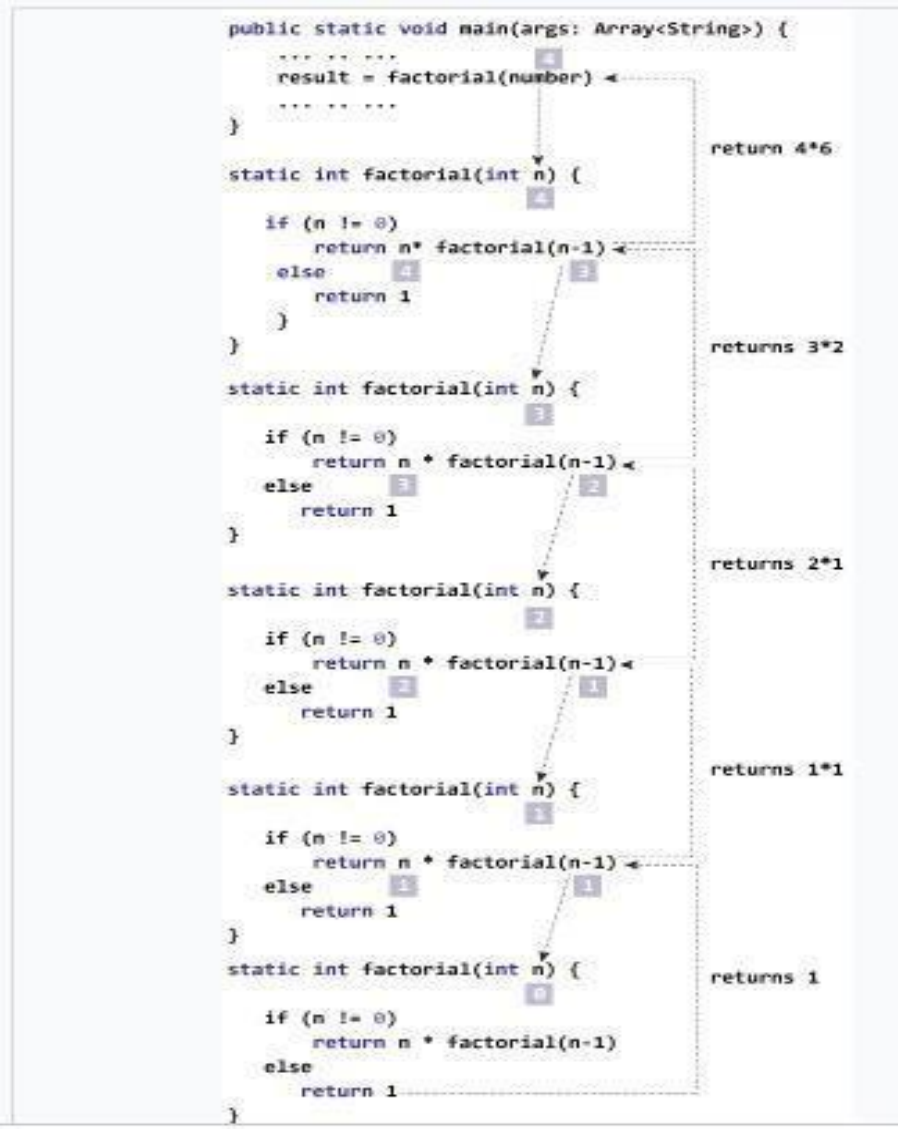
return n * factorial(n-1);

The factorial() method is calling itself. Initially, the value of n is 4 inside factorial(). During the next recursive call, 3 is passed to the factorial() method. This process continues until n is equal to 0.

When n is equal to 0, the if statement returns false hence 1 is returned. Finally, the accumulated result is passed to the main() method.

Working of Factorial Program

The image below will give you a better idea of how the factorial program is executed using recursion.



The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

Backtracking

Backtracking is an algorithm technique for recursive problem by trying to build every possible solution incrementally and removing those solutions that fails to satisfy the constraints of a problem at any point of time

Backtracking Algorithm

A backtracking algorithm is a problem-solving algorithm that uses a brute force approach for finding the desired output.

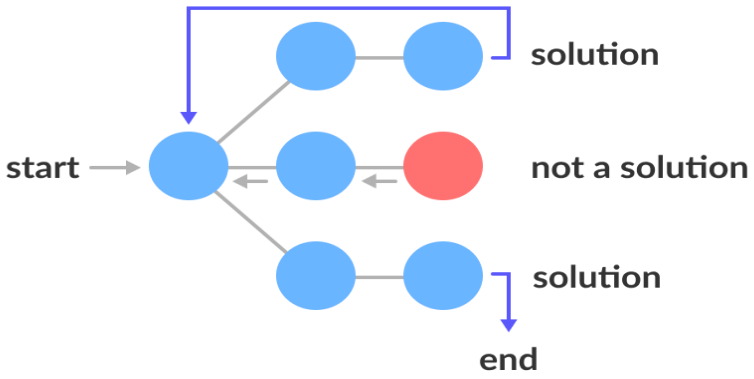
The Brute force approach tries out all the possible solutions and chooses the desired/best solutions.

The term backtracking suggests that if the current solution is not suitable, then backtrack and try other solutions. Thus, recursion is used in this approach.

This approach is used to solve problems that have multiple solutions. If you want an optimal solution, you must go for dynamic programming.

State Space Tree

A space state tree is a tree representing all the possible states (solution or nonsolution) of the problem from the root as an initial state to the leaf as a terminal state.



Backtracking Algorithm

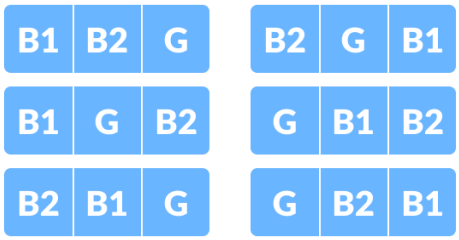
```
Backtrack(x)

    if x is not a solution

        return false

    if x is a new solution

        add to list of solutions

    backtrack(expand x)
```

Example Backtracking Approach

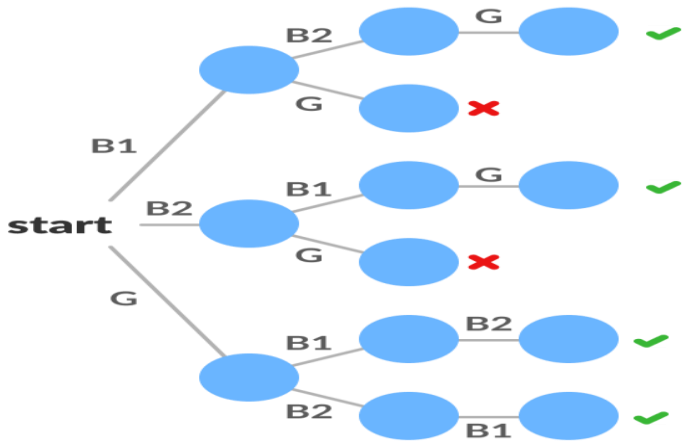
Problem: You want to find all the possible ways of arranging 2 boys and 1 girl on 3 benches.
Constraint: Girl should not be on the middle bench.

Solution: There are a total of 3! = 6 possibilities.

We will try all the possibilities and get the possible solutions.
We recursively try all the possibilities.
All the possibilities are:



The following state space tree shows the possible solutions.



Algorithm for given example

Initialize Variables:
Set boysCount to 2 (number of boys).
Set girlsCount to 1 (number of girls).
Set totalBenches to 3 (total number of benches).
Create an array arrangement of size totalBenches to store the current arrangement.

Recursive Function arrangeBenches:
Base Case:
If arrangement is complete (length is equal to totalBenches), check and print if the girl is not on the middle bench (index 1).
Return.
Iterate over all benches (i) from 0 to totalBenches - 1:
If the bench at index i is not occupied:
Set arrangement[position] to 'B' if i is not 1 (middle bench), otherwise set it to 'G'.
Recursively call arrangeBenches with updated parameters (position + 1).
Backtrack by clearing the occupation of the current bench.

Main Function:
Call the recursive function arrangeBenches with initial parameters (boysCount, girlsCount, totalBenches, arrangement, 0).

Lab Task

Task-1:
Given an integer N, the task is to print the first N terms of the Fibonacci series in reverse order using Recursion.

Task-2:
Given a number, we need to find sum of its digits using recursion.
Example 1
Input: 12345
Output: 15

Example 2
Input: 45632
Output: 20

Task-3:
Print sum of first 30 natural numbers

Task 4:
Write a recursive method in java that check that given array is sorted or not.

Task 5:
Given a string S, the task is to write a program to print all permutations of a given string.
A permutation also called an "arrangement number" or "order," is a rearrangement of the elements of an ordered list S into a one-to-one correspondence with S itself. A string of length N has N! permutations.
Examples:
Input: S = "ABC"
Output: "ABC", "ACB", "BAC", "BCA", "CBA", "CAB"
Input: S = "XY"
Output: "XY", "YX"

Task 6:
implement a recursive function to solve the Tower of Hanoi problem. The Tower of Hanoi is a classic problem that involves moving a stack of disks from one rod to another, subject to the constraint that only one disk can be moved at a time, and no disk can be placed on top of a smaller disk.

Task 7:
Implement a C++/Java program to solve a Sudoku puzzle using backtracking. The program should be able to take an incomplete Sudoku board as input, where 0 represents empty cells, and fill in the missing numbers such that the completed board satisfies the rules of Sudoku - each row, each column, and each of the nine 3×3 sub-grids that compose the grid should contain all of the digits from 1 to 9.
Key points
1. Each row should have numbers 1-9, no repeats
2. Each column should have numbers 1-9, no repeats
3. Each 3x3 quadrant should have numbers 1-9, no repeats