DATA STRUCTURE AND ALGORITHM

Lecturer: Er. Saroj Ghimire

Qualification: Msc.CSIT, BE(COMPUTER)

Lincoln University College

Overview

- Principles of recursion
- Comparison between recursion and iteration
- Factorial, Fibonacci, GCD & Tower of Hanoi

Objectives

- To learn how to formulate programs recursively.
- To understand and apply the three laws of recursion.

Principles of recursion

- The recursion is a process by which a function calls itself.
- We use recursion to solve bigger problem into smaller sub-problems.
- Recursion is a programming technique using function or algorithm that calls itself one or more times until a specified condition is met at which time the rest of each repetition is processed from the last one called to the first.
- For example, let's look at a recursive definition of a person's ancestors:

  One's parents are one's ancestors. The parents of any ancestor are also ancestors of the person under consideration

  We can write pseudo code to determine whether somebody is someone's ancestor.

```
FUNCTION is Ancestor (Person x, Person y):
    IF x is y's parent, THEN:
        return true
    ELSE
        return is Ancestor (x, y's mom) OR is Ancestor (x, y's dad)
}
```

- Thus, a recursive function usually has a certain structure:
  1. **base case**, which does not call the function itself;
  2. **recursive step**, which calls the function itself and moves closer to the base case.

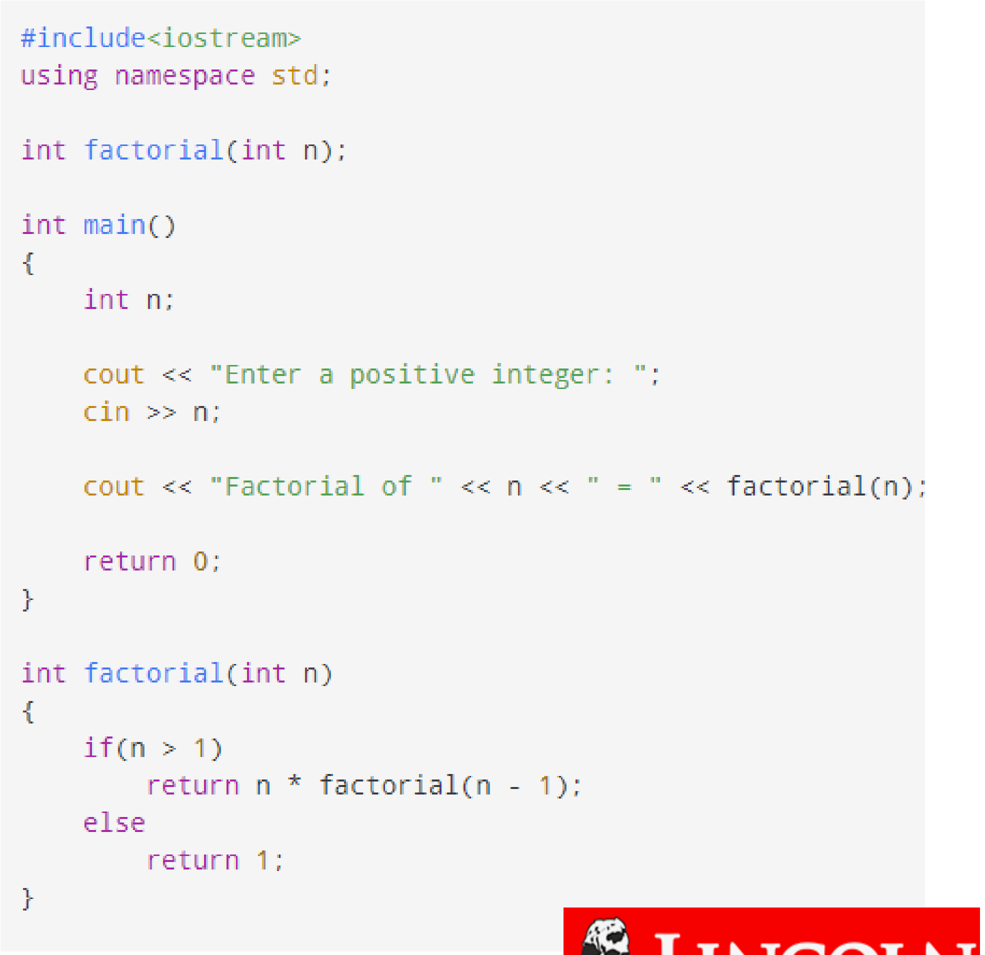
Comparison between recursion and iteration

Recursion	Iteration
1. Recursion is like piling all of those steps on top of each other and then quashing the mall into the solution.	1. In iteration, a problem is converted into a train of steps that are finished one at a time, one after another.
2. In recursion, each step replicates itself at a smaller scale, so that all of them combined together eventually solve the problem.	2. With iteration, each step clearly leads on to the next, like stepping stones across a river.
3. Not all recursive problem can solved by iteration	3. Any iterative problem is solved recursively
4. It uses Stack	4. It does not use Stack

5. int fib(**int** n)
{ **if**(n <= 1)
{ **return** n }
return fib(n-1) + fib(n-2)
}
	5. int fib(**int** n)
	{ **if**(n <= 1)
	return n
	a = 0, b = 1 **for**(i = 2 to n)
	{ c = a + b
	a = b
	b = c }
	return c
	}

Factorial, Fibonacci, GCD & Tower of Hanoi

- Factorial
  - The factorial of a number is the product of all the integers from 1 to that number.
  - For example, the factorial of 6 is 1*2*3*4*5*6 = 720.
  - Factorial is not defined for negative numbers and the factorial of zero is one, 0! = 1



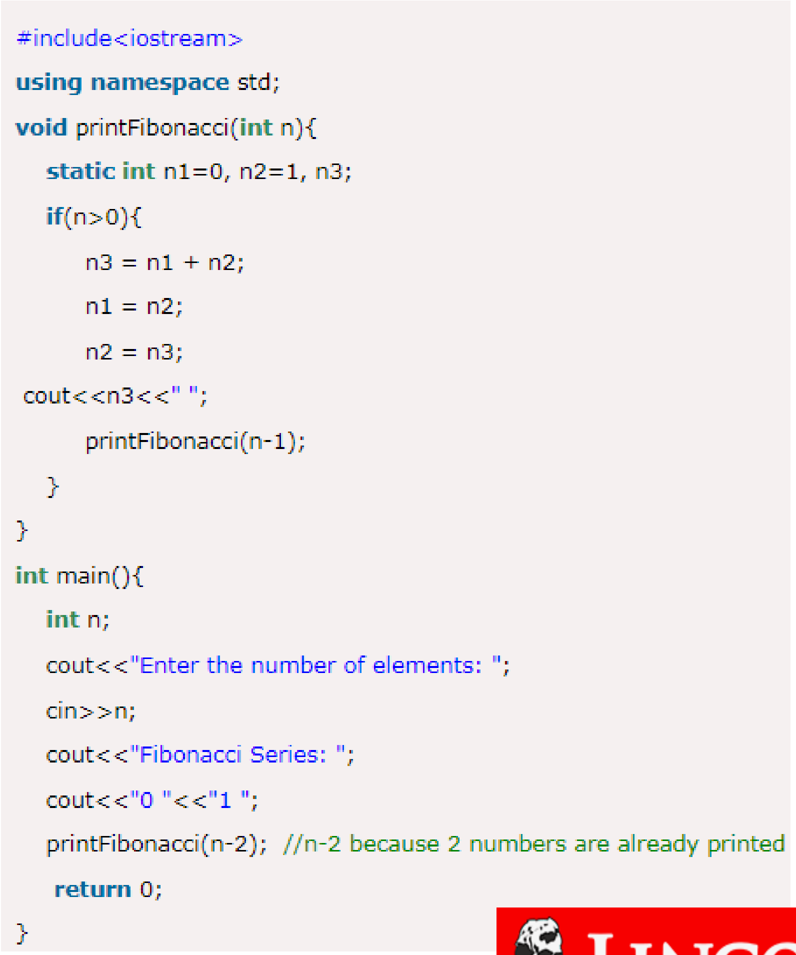
```
#include<iostream>
using namespace std;

int factorial(int n);

int main()
{
    int n;

    cout << "Enter a positive integer: ";
    cin >> n;

    cout << "Factorial of " << n << " = " << factorial(n);

    return 0;
}

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

Factorial, Fibonacci, GCD & Tower of Hanoi

➢ Fibonacci

– The Fibonacci numbers are the numbers in the following integer sequence.0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..

– In mathematical terms, the sequence F_n of Fibonacci numbers is defined by the recurrence relation $F_n = F_{n-1} + F_{n-2}$ with seed values $F_0 = 0$ and $F_1 = 1$



```
#include<iostream>
using namespace std;
void printFibonacci(int n){
    static int n1=0, n2=1, n3;
    if(n>0){
         n3 = n1 + n2;
         n1 = n2;
         n2 = n3;
 cout<<n3<<" ";
         printFibonacci(n-1);
    }
}
int main(){
    int n;
    cout<<"Enter the number of elements: ";
    cin>>n;
    cout<<"Fibonacci Series: ";
    cout<<"0 "<<"1 ";
    printFibonacci(n-2);  //n-2 because 2 numbers are already printed
     return 0;
}
```

Factorial, Fibonacci, GCD & Tower of Hanoi

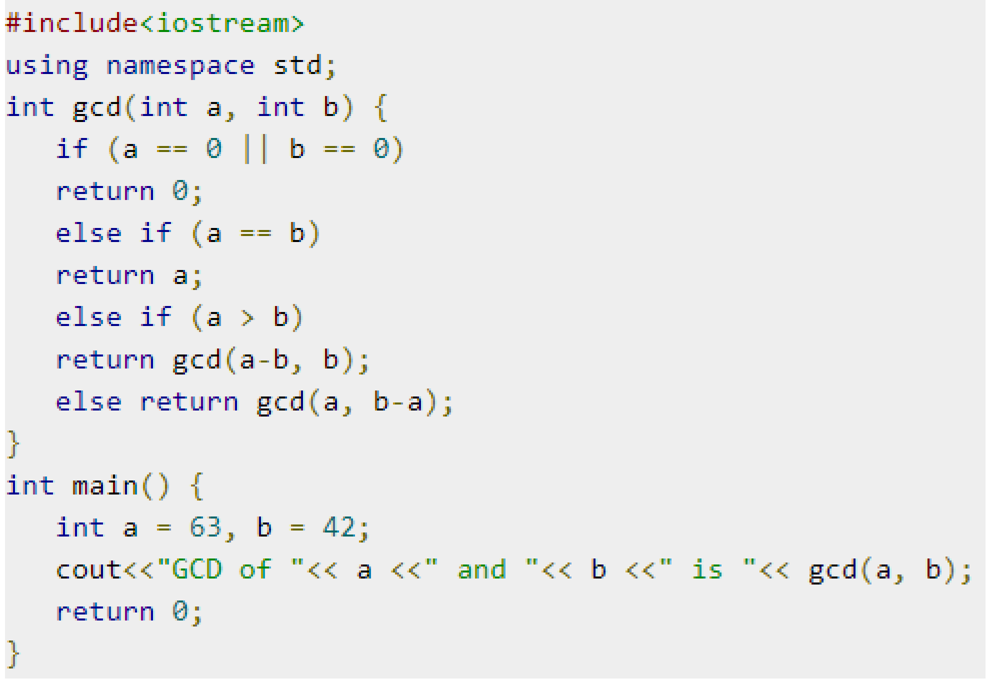
➢ GCD

- GCD (Greatest Common Divisor) or HCF (Highest Common Factor) of two numbers is the largest number that divides both of them.
- For example: Let’s say we have following two numbers: 63 and 42

63 = 7 * 3 * 3

7 * 3 * 2

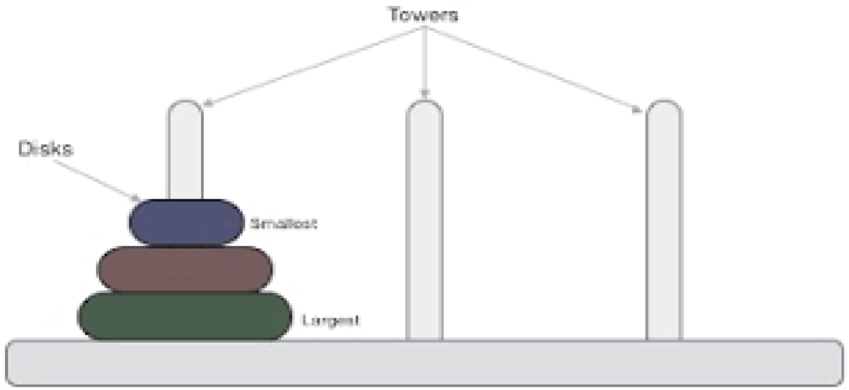
So, the GCD of 63 and 42 is 21

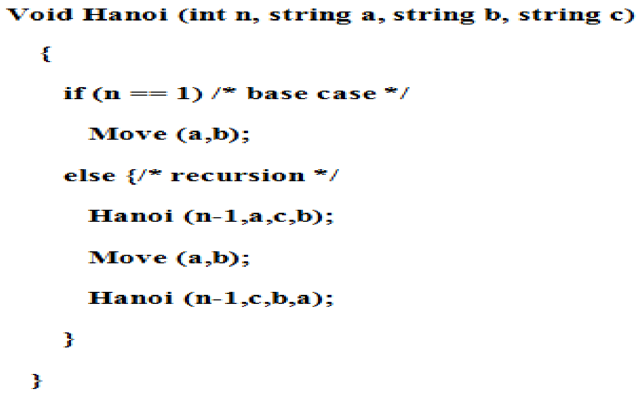


```
#include<iostream>
using namespace std;
int gcd(int a, int b) {
   if (a == 0 || b == 0)
   return 0;
   else if (a == b)
   return a;
   else if (a > b)
   return gcd(a-b, b);
   else return gcd(a, b-a);
}
int main() {
   int a = 63, b = 42;
   cout<<"GCD of "<< a <<" and "<< b <<" is "<< gcd(a, b);
   return 0;
}
```

Tower of Hanoi

- Tower of Hanoi, is a mathematical puzzle which consists of three tower pegs and more than one rings; as depicted below –





```
Void Hanoi (int n, string a, string b, string c)
  {
    if (n == 1) /* base case */
      Move (a,b);
    else {/* recursion */
      Hanoi (n-1,a,c,b);
      Move (a,b);
      Hanoi (n-1,c,b,a);
    }
  }
```

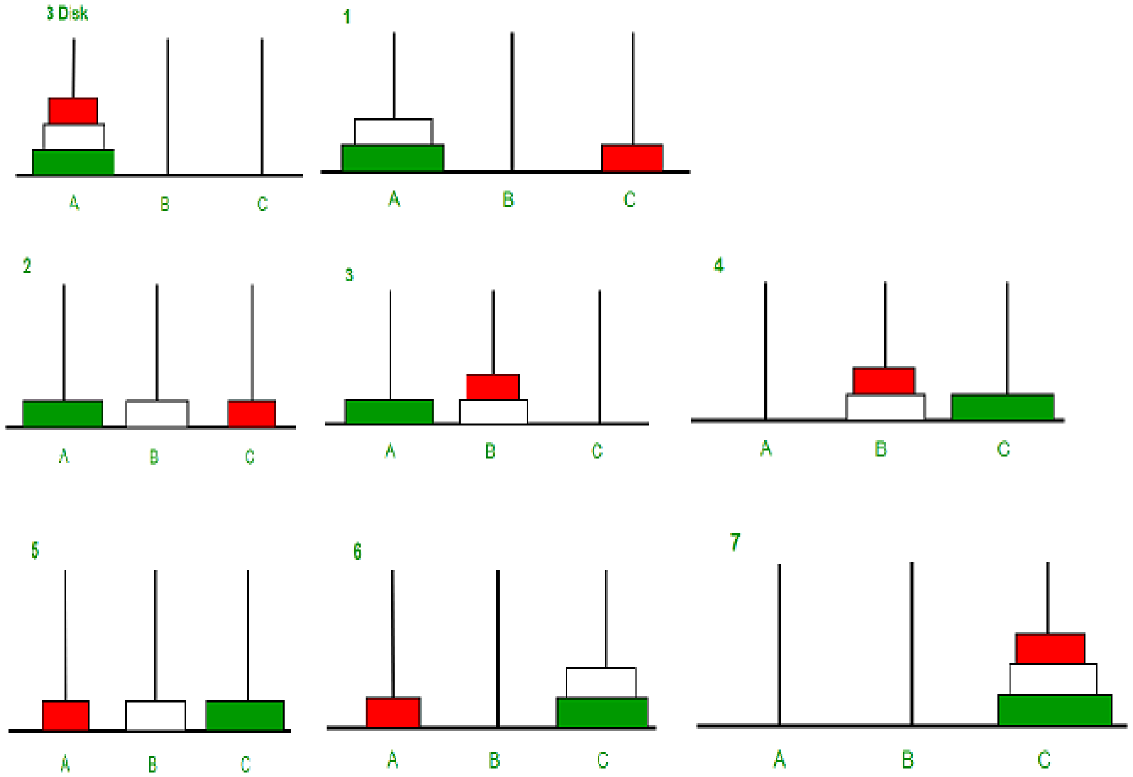
- It consists of three rods and a number of disks of different sizes, which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape. The objective of the puzzle is to move the entire stack to last rod, obeying the following simple rules:
  - Only one disk can be moved at a time.
  - Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
  - No larger disk may be placed on top of a smaller disk.

Tower of Hanoi

- For an even number of disks:
  - make the legal move between pegs A and B (in either direction),
  - make the legal move between pegs A and C (in either direction),
  - make the legal move between pegs B and C (in either direction),
  - repeat until complete.
- For an odd number of disks:
  - make the legal move between pegs A and C (in either direction),
  - make the legal move between pegs A and B (in either direction),
  - make the legal move between pegs B and C (in either direction),
  - repeat until complete.
- In each case, a total of $2^n - 1$ moves are made.

Tower of Hanoi

```
Void Hanoi (int n, string a, string b, string c)
   {
      if (n == 1) /* base case */
         Move ( a, b);
      else {/* recursion */
         Hanoi (n-1,a,c,b);
         Move ( a, b);
         Hanoi (n-1,c,b,a);
      }
   }
```



Further Readings

- Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Prentice Hall, 1988.
- Data Structures and Algorithms; Shi-Kuo Chang; World Scientific.
- Data Structures and Efficient Algorithms, Burkhard Monien, Thomas Ottmann, Springer.
- Kruse Data Structure & Program Design, Prentice Hall of India, New Delhi
- Mark Allen Weles: Data Structure & Algorithm Analysis in C Second Adition. Addison-Wesley publishing
- RG Dromey, How to Solve it by Computer, Cambridge University Press.
- Shi-kuo Chang, Data Structures and Algorithms, World Scientific
- Sorenson and Tremblay: An Introduction to Data Structure with Algorithms.
- Thomas H. Cormen, Charles E, Leiserson & Ronald L. Rivest: Introduction to Algorithms.
- Prentice-Hall of India Pvt. Limited, New Delhi Timothy A. Budd, Classic Data Structures in C++, Addison Wesley