Recursion

Unit 4

Recursion

- Recursion is a process by which a **function calls itself repeatedly**, until some specified condition has been satisfied.
- Most powerful programming tool.
- The process is **used for repetitive computations** in which each action is stated in terms of a previous result.
- In order to solve a problem recursively, two conditions must be satisfied.
    - The problem must be written in a recursive form
    - the problem statement must include a stopping condition

factorial of an integer number using recursive function

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int n;
        long int facto;
        long int factorial(int n);
        printf("Enter value of n:");
        scanf("%d",&n);
        facto=factorial(n);
        printf("%d! = %ld",n,facto);
        getch();
}
```

```
long int factorial(int n)
{
        if(n == 0)
                return 1;
        else
                return n * factorial(n-1);
}
```

Recursive example for 5!

5! = 5*4!

4! = 4*3!

3! = 3*2!

2! = 2*1!

1! = 1*0!

0! = 1

0! = 1
1! = 1*0! = 1*1 = 1
2! = 2* 1! = 2*1 = 2
3! = 3* 2! = 3*2 = 6
4! = 4* 3! = 4*6 = 24
5! = 5* 4! = 5*24 = 120

Calculation of the factorial of an integer number without using recursive function

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int n;
   long int facto;
   long int factorial(int n);
   printf("Enter value of n:");
   scanf("%d",&n);
   facto=factorial(n);
   printf("%d! = %ld",n,facto);
   getch();
}
```

```
long int factorial(int n)
{
// Complete this function




}
```

calculation of the factorial of an integer number without using recursive function

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int n;
   long int facto;
   long int factorial(int n);
   printf("Enter value of n:");
   scanf("%d",&n);
   facto=factorial(n);
   printf("%d! = %ld",n,facto);
   getch();
}
```

```
long int factorial(int n)
{
      long int facto=1;
      int i;
      if(n==0)
         return 1;
   else {
      for(i=1;i<=n;i++)
         facto=facto*i;
      return facto;
   }
}
```

Key Idea Behind using recursive function,

- Break the problem to small and solve it through
- Example:
    - Calculating factorial… let us take 3! So to get three factorial
        - Multiply 3 with 2!
        - To get 2! Multiply 2 with 1!

        - That is n factorial can be calculated as n * factorial(n-1)
- Example
    - Calculating sum of first n natural numbers if n = 5 sum will be 5+4+3+2+1
        - That will be given by n+ sum(n-1)

Types of Recursion

1. Direct Recursion: a function calls itself from within itself

```
int abc(){
          ………;
          abc();
}
```

2. Indirect Recursion: Two functions call one another mutually

```
int abc(){
          ………;
          xyz();
}
int xyz(){
          ………;
          abc();
}
```

Program to generate Fibonacci series up to n terms using recursive function

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int n,i;
   int fibo(int);
   printf("Enter n:");
   scanf("%d",&n);
   printf("Fibonacci numbers up to %d terms:\n",n);
   for(i=1; i<=n; i++)
      printf("%d\n",fibo(i));
   getch();
}
```

```
int fibo(int k)
{
   if(k == 1 || k == 2)
      return 1;
   else
      return fibo(k-1)+fibo(k-2);
}
```

Program to find sum of first n natural numbers using recursion

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int n;
   int sum_natural(int );
   printf("n = ");
   scanf("%d",&n);
   printf("Sum of first %d natural numbers = %d", n, sum_natural(n));
   getch();
}
```

```
int sum_natural(int n)
{
   if(n == 1)
      return 1;
   else
      return n + sum_natural(n-1);
}
```

Program to find multiplication of first n natural numbers using recursion

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int n;
        int mul_natural(int );
        printf("n = ");
        scanf("%d",&n);
        printf("Product of first %d natural numbers = %d", n, mul_natural(n));
        getch();
}
```

```
int mul_natural(int n)
{
if(n == 1)
return 1;
else
return (n * mul_natural(n-1));
}
```

Tower of Hanoi

- **Initial state:**
- There are three poles named as origin, intermediate and destination.
- N number of different-sized disks having hole at the center is stacked around the
- origin pole in decreasing order.
- The disks are numbered as 1, 2, 3, 4, ...,n

Tower of Hanoi Problem

Objective:

- Transfer all disks from origin pole to destination pole using intermediate pole for temporary storage.

Conditions:

- Move only one disk at a time.
- Each disk must always be placed around one of the pole.
- Never place larger disk on top of smaller disk.

Algorithm: To move a tower of n disks from source to dest (where N is positive integer):

- 1. If n ==1:
  - 1.1. Move a single disk from source to dest.
- 2. If n > 1:
  - 2.1. Let temp be the remaining pole other than source and dest
  - 2.2. Move a tower of (n– 1) disks form Source to temp
  - 2.3. Move a single disk from Source to dest.
  - 2.4. Move a tower of (n– 1) disks form temp to dest.
- 3. Terminate.

Recursive solution of tower of Hanoi:

```
#include <stdio.h>
#include <conio.h>
void TOH(int, char, char, char);    //Function prototype
void main()
{
    int n;
    clrscr();
    printf("Enter number of disks ");
    scanf(" %d", &n);
    TOH(n,'S','D','T');
    getch();
}
```

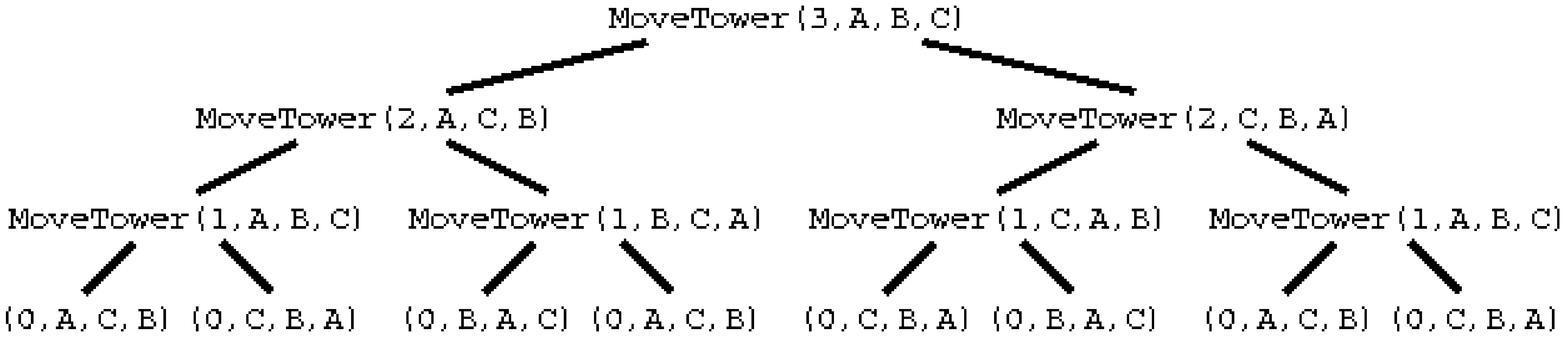
```
void TOH(int n, char A, char B, char C)
{
    if(n>0)
    {
            TOH(n-1, A, C, B);
            printf("Move disk %d from %c to %c \n", n, A, B);
            TOH(n-1, C, B, A);
    }
}
```

Advantages/ Disadvantages of Recursion:

- **Advantages of Recursion:**
- The code may be much easier to write.
- To solve some problems which are naturally recursive such as tower of Hanoi.

- **Disadvantages of Recursion:**
- It is not more efficient in terms of speed and execution time.
- May require a lot of memory to hold intermediate results on the system stack.
- If proper precautions are not taken, recursion may result in non-terminating iterations.
- It is difficult to think recursively so one must be very careful when writing recursive functions.

Recursion tree of TOH



MoveTower(3, A, B, C)

MoveTower(2, A, C, B) MoveTower(2, C, B, A)

MoveTower(1, A, B, C) MoveTower(1, B, C, A) MoveTower(1, C, A, B) MoveTower(1, A, B, C)

(0, A, C, B) (0, C, B, A) (0, B, A, C) (0, A, C, B) (0, C, B, A) (0, B, A, C) (0, A, C, B) (0, C, B, A)