Factorial of a Number

A simple example of a recursively defined function is factorial function:

n!=1×2×3×4….(n-2)×(n-1)×n

i.e the product of the first n positive numbers (by convention, the product of

nothing is 1, so that 0!=1).

Factorial-Iteration

An iterative algorithm uses loop(for loop), but it is not recursive.

def factorial(n):

1. t = 1

2 . For i in range(1,n+1):

i. t = t * i

3. return t

Iteration Method:

```
#include<stdio.h>
int main (){
      int n, fact=1;
      printf("Factorial of : ");
      scanf("%d", &n);
      for (int i=1; i<=n; i++){
            fact=fact*i;
      }
      printf("Factorial is :  %d", fact);
}
```

Recursion Method:

```
#include<stdio.h>
int factorial(int n){
      return (n == 0) ? 1 : n * factorial(n-1);
}
int main(){
    int n;
    printf("Factorial of : ");
    scanf("%d", &n);
    printf("Factorial is: %d", factorial(n));
}
```

Tail Recursion Method:

```
#include<stdio.h>
int factorial(int n, int fact){
    return (n==1) ? fact : factorial(n-1, n*fact);
}
int main(){
    int n;
    printf("Factorial oF : ");
    scanf("%d", &n);
    printf("The factorial is: %d",factorial(n,1));
}
```

Fibonacci series

A Fibonacci sequence is an integer series in which each member is the sum of the two elements before it. The Fibonacci series begins with the two numbers F0 and F1. F0 and F1's initial values can be 0, 1, or 1 respectively.

Fibonacci Sequence – Recursion

Function fibonacci(n)

1. If n == 0 or n == 1

    i. fibonacci = n

Else

    fibonacci = fibonacci(n-1) + fibonacci(n-2)

Iteration Method:

```
#include<stdio.h>

int main(){

    int a=0, b=1, c;

    printf("Fibonacci series:\n");

    for(int i=1; i<=10; i++){

        printf("%d ",a);

        c = a + b;

        a = b;

        b = c;

    }

}
```

Recursion Method:

```
#include<stdio.h>
int fibo(int a){
    if (a==0 || a==1){
        return a;
    }
    else{
        return fibo(a-1)+fibo(a-2);
    }
}
int main (){
    int n, i, f1=0;
    printf("Enter a number of terms:");
    scanf("%d", &n);
    printf("Fibonacci series:");
    for(i=1; i<=n; i++){
        printf ("\t%d", fibo(f1));
        f1++;
    }
}
```

Tail Recursion Method:

```
#include<stdio.h>
int fibo(int n, int a=0, int b=1){
    if (n==0 || n==1){
        return a;
    }
    return (fibo(n-1, b, a+b));
}
int main(){
int n, i, f1=1;
printf("Enter a number of terms:");
scanf("%d", &n);
printf("Fibonacci series:");
    for(i=1; i<=n; i++){
        printf ("\t%d", fibo(f1));
        f1++;
    }
}
```

HCF of a number.

The HCF of two positive integers a and b is the greatest number that divides both a and b.

ALGORITHM

function GCD(a, b)

Input: integers a > 0, b ≥ 0

Output: GCD (a, b)

1. If b = 0 then return (a)

2. else return GCD(b, a mod b)

Iteration Method:

```
#include<stdio.h>
int main() {
    int a, b, i, hcf;
    printf("Enter two numbers:");
    scanf("%d %d", &a, &b);
    for(i = 1; i <= a || i <= b; i++) {
        if( a%i == 0 && b%i == 0 ){
            hcf = i;
        }
    }
    printf("HCF of %d, %d = %d",a, b ,hcf);
}
```

Tail Recursion Method:

```
#include<stdio.h>
int hcf(int n1, int n2) {
    if (n2 != 0) {
        return hcf(n2, n1 % n2);
    } else {
        return n1;
    }
}
int main() {

    int a, b, i;
```

```
    printf("Enter two numbers:");
    scanf("%d %d", & a, & b);
    printf("HCF of %d, %d = %d", a, b, hcf(a, b));
    return 0;
}
```

TOH(Tower of hanoi).

TOH is a mathematical puzzle which consists of three pegs named as origin, intermediate and destination and more than one disks. These disks are of different sizes and the smaller one sits over the darger one.

ALGORITHM

function TOH(disk, Source, Dest, Aux):

    if disk == 1:

        move disk from Source to Dest

    else:

        TOH(disk - 1, Source, Aux, Dest)

        move disk from Source to Dest

        TOH(disk - 1, Aux, Dest, Source)

Iteration Method:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <limits.h>
struct Stack {
```

```
    unsigned capacity;
    int top;
    int * array;
};
struct Stack * createStack(unsigned capacity) {
    struct Stack * stack = (struct Stack * ) malloc(sizeof(struct Stack));
    stack -> capacity = capacity;
    stack -> top = -1;
    stack -> array = (int * ) malloc(stack -> capacity * sizeof(int));
    return stack;
}
int isFull(struct Stack * stack) {
    return (stack -> top == stack -> capacity - 1);
}
int isEmpty(struct Stack * stack) {
    return (stack -> top == -1);

}
void push(struct Stack * stack, int item) {
    if (isFull(stack))
        return;
    stack -> array[++stack -> top] = item;
}
int pop(struct Stack * stack) {
    if (isEmpty(stack))
        return INT_MIN;
```

```
    return stack -> array[stack -> top--];
}
void moveDisk(char fromPeg, char toPeg, int disk) {
    printf("Move the disk %d from \'%c\' to \'%c\'\n",
        disk, fromPeg, toPeg);
}
void moveDisksBetweenTwoPoles(struct Stack * src,
    struct Stack * dest, char s, char d) {
    int pole1TopDisk = pop(src);
    int pole2TopDisk = pop(dest);
    if (pole1TopDisk == INT_MIN) {

        push(src, pole2TopDisk);
        moveDisk(d, s, pole2TopDisk);

    } else if (pole2TopDisk == INT_MIN) {
        push(dest, pole1TopDisk);
        moveDisk(s, d, pole1TopDisk);
    } else if (pole1TopDisk > pole2TopDisk) {
        push(src, pole1TopDisk);
        push(src, pole2TopDisk);
        moveDisk(d, s, pole2TopDisk);
    } else {
        push(dest, pole2TopDisk);
        push(dest, pole1TopDisk);
        moveDisk(s, d, pole1TopDisk);
```

```
    }
}
void tohIterative(int num_of_disks, struct Stack *
    src, struct Stack * aux,
    struct Stack * dest) {

    int i, total_num_of_moves;
    char s = 'J', d = 'K', a = 'L';
    if (num_of_disks % 2 == 0) {
        char temp = d;
        d = a;
        a = temp;
    }
    total_num_of_moves = pow(2, num_of_disks) - 1;
    for (i = num_of_disks; i >= 1; i--)
        push(src, i);
    for (i = 1; i <= total_num_of_moves; i++) {
        if (i % 3 == 1)
            moveDisksBetweenTwoPoles(src, dest, s, d);
        else if (i % 3 == 2)
            moveDisksBetweenTwoPoles(src, aux, s, a);
        else if (i % 3 == 0)
            moveDisksBetweenTwoPoles(aux, dest, a, d);
    }
}
int main() {
```

```
    unsigned num_of_disks = 3;
    struct Stack * src, * dest, * aux;
    src = createStack(num_of_disks);
    aux = createStack(num_of_disks);

    dest = createStack(num_of_disks);
    tohIterative(num_of_disks, src, aux, dest);
}
```

Tail Recursion Method:

```
#include<stdio.h>
void toh(int ndisk, char source, char temp, char dest) {
    if (ndisk > 0) {
        toh(ndisk - 1, source, dest, temp);
        printf("Move Disk %d %c-->%c\n", ndisk, source, dest);
        toh(ndisk - 1, temp, source, dest);
    }
}
int main() {
    char source = 'S', temp = 'T', dest = 'D';
    int ndisk;
    printf("Enter the number of disks : ");
    scanf("%d", & ndisk);

    printf("Sequence is :\n");
    toh(ndisk, source, temp, dest);
    return 0;
}
```