LECTURE NOTE 17

RECURSION VERSES ITERATION

Every repetitive problem can be implemented recursively or iteratively

Recursion should be used when the problem is recursive in nature. Iteration should be used when the problem is not inherently recursive

Recursive solutions incur more execution overhead than their iterative counterparts, but its advantage is that recursive code is very simple.

Recursive version of a problem is slower than iterative version since it requires PUSH and POP operations.

In both recursion and iteration, the same block of code is executed repeatedly, but in recursion repetition occurs when a function calls itself and in iteration repetition occurs when the block of code is finished or a continue statement is encountered.

For complex problems iterative algorithms are difficult to implement but it is easier to solve recursively. Recursion can be removed by using iterative version.

Tail Recursion

A recursive call is said to be tail recursive if the corresponding recursive call is the last statement to be executed inside the function.

Example: Look at the following recursive function

```
void show(int a)
{
if(a==1)
 return;
printf("%d",a);
show(a-1);
 }
```

In the above example since the recursive call is the last statement in the function so the above recursive call is Tail recursive call.

In non void functions(return type of the function is other than void) , if the recursive call appears in return statement and the call is not a part of an expression then the call is said to be Tail recursive, otherwise Non Tail recursive. Now look at the following example

```
int hcf(int p, int q)
{
if(q==0)
return p;
else

return(hcf(q,p%q));    /*Tail recursive call*/
}
int factorial(int a)
{
if(a==0)
return 1;
else

 return(a*factorial(a-1));       /*Not a Tail recursive call*/
}
```

In the above example in hcf() the recursive call is not a part of expression (i.e. the call is the expression in the return statement) in the call so the recursive call is Tail recursive. But in factorial() the recursive call is part of expression in the return statement(a*recursive call) , so the recursive call in factorial() is not a Tail excursive call.

A function is said to be Tail recursive if all the recursive calls in the function are tail recursive.

Tail recursive functions are easy to write using loops,

In tail recursive functions, the last work that a function does is a recursive call, so no operation is left pending after the recursive call returns. Therefore in tail recursive functions , there is nothing to be done in unwinding phase, so we can jump directly from the last recursive call to the place where recursive function was first called.

Tail recursion can be efficiently implemented by compilers so we always will try to make our recursive functions tail recursive whenever possible.

Functions which are not tail recursive are called augmentive recursive functions and these types of functions have to finish the pending work after the recursive call finishes.

Indirect and Direct Recursion

If a function fun1() calls another function fun2() and the function fun2() in turn calls function fun1(), then this type of recursion is said to be **indirect** recursion, because the function fun1() is calling itself indirectly.

```
fun1( )
{
……..……..     /* Some statements*/
fun2();
……..……..     /* Some statements*/
}
fun2( )
{
……..……..     /* Some statements*/
fun1();
……..……..     /* Some statements*/
}
```

The chain of functions in indirect recursion may involve any number of functions.For example suppose n number of functions are present starting from f1() to fn() and they are involved as following: f1() calls f2(), f2() calls f3(), f3() calls f4() and so on with fn() calls f1().

If a function calls itself directly i.e. function fun1() is called inside its own function body, then that recursion is called as direct recursion. For example look at the following

```
fun1()
{
...      /* Some statements*/
fun2();
...      /* Some statements*/
}
```

Indirect recursion is complex, so it is rarely used.

Exercise:

Find the output of programs from 1 to 5.

1.
```
void main()
{
 printf("%d\n",count(17243));
 }
  int count(int x)
{
 if(x==0)
       return 0;
 else
        return 1+count(x/10)
 }
```

2.
```
void main()
{
printf("%d\n",fun(4,8));
printf("%d\n",fun(3,8));
}
int fun(int x. int y)
{
 if(x==y)
    return x;
 else
     return (x+y+fun(x+1,y-1));
}
```

3.
```
void main()
{
 printf("%d\n",fun(4,9));
printf("%d\n",fun(4,0));
 printf("%d\n",fun(0,4));
}
int fun(int x, int y)
{
if(y==0)
        return 0;
 if(y==1)
       return x;
  return x+fun(x,y-1);
}
```

4.
```
void main()
{
 printf("%d\n",fun1(14837));
}
int fun1(int m)
{
return ((m)? m%10+fun1(m/10):0);
```

```
}
```

5.
```
void main()
{
printf("%d\n",fun(3,8));
}
int fun(int x, int y)
{
if(x>y)
   return 1000;
return x+fun(x+1,y);
}
```

6. What is the use of recursion in a program, Explain?
7. Explain the use of stack in recursion.
8. What do you mean by winding and unwinding phase?
9. How to write a recursive function, Explain with example?
10. What is the difference between tail and non-tail recursion, explain with example.
11. What is indirect recursion?
12. What is the difference between iteration and recursion?
13. Write a recursive function to enter a line of text and display it in reverse order, without storing the text in an array.
14. Write a recursive function to count all the prime numbers between number p and q(both inclusive).
15. Write a recursive function to find quotient when a positive integer m is divided by a positive integer n.
16. Write a program using recursive function to calculate binary equivalent of a number.
17. Write a program using recursive function to reverse number.
18. Write a program using recursive function to find remainder when a positive integer m is divided by a positive integer n.
19. Write a recursive function that displays a positive integer in words, for example if the integer is 465 then it is displayed as -four six five.
20. Write a recursive function to print the pyramids

1	abcd
1 2	abc
1 2 3	ab
1 2 3 4	a

21. Write a recursive function to find the Binomial coefficient C(n,k), which is defined as:
    C(n,0)=1
    C(n,n)=1
    C(n,k)=C(n-1,k-1)+C(n-1,k)