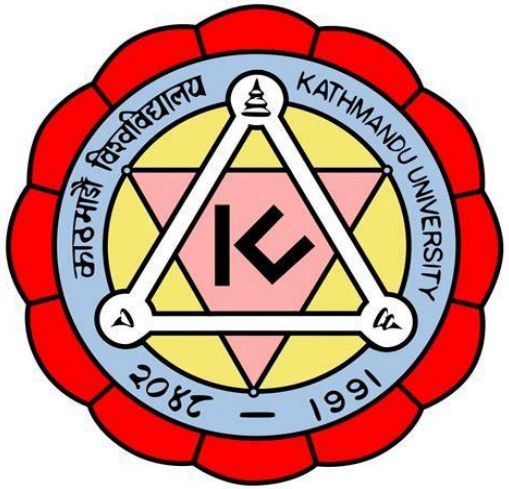
KATHMANDU UNIVERSITY SCHOOL OF MANAGEMENT

BBIS

COM 102 : 3 Credit Hours



7. Functions contd…

Functions Contd...

Call by Value and Call by Reference

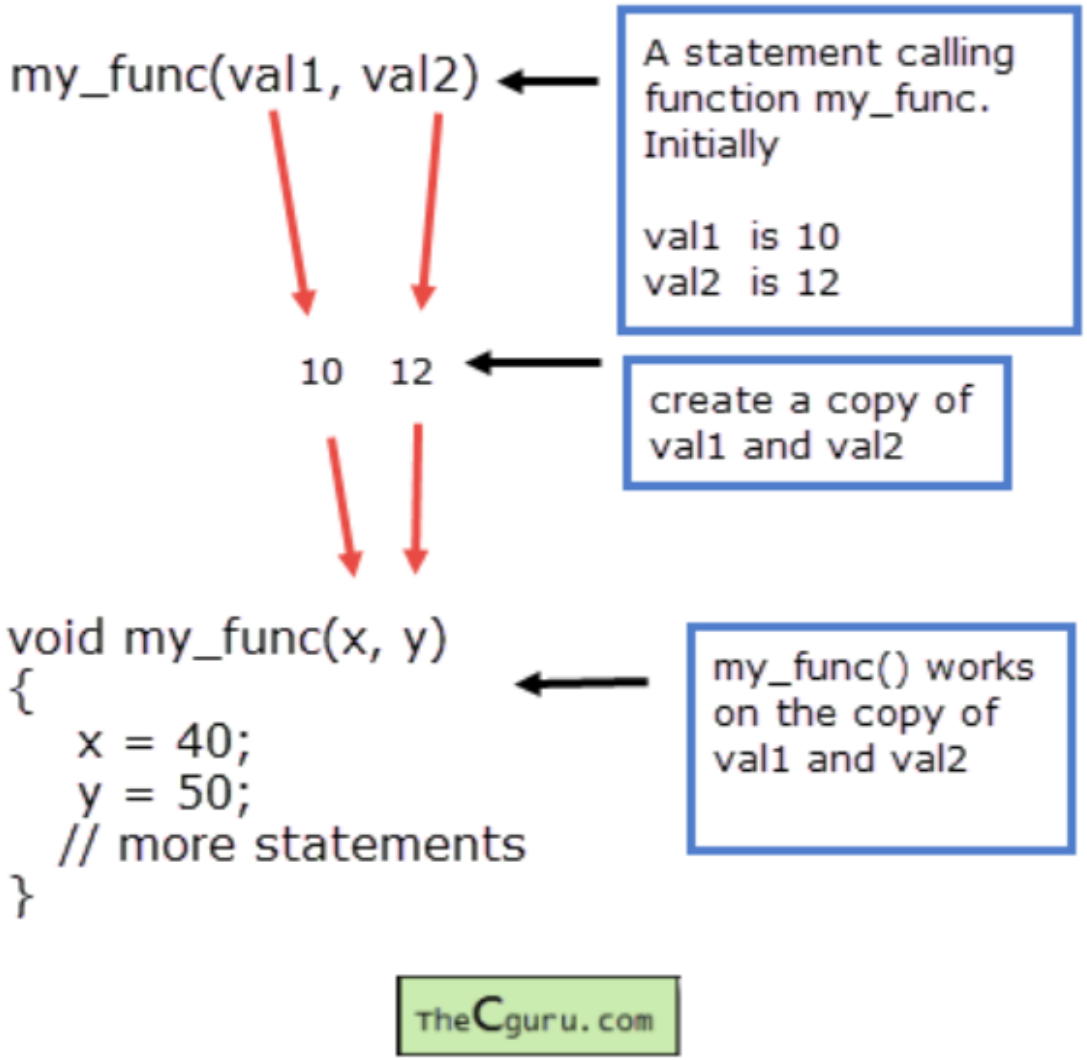
Introduction

- Function call by value is the default way of calling a function in C programming.
- terminologies that we will use while explaining this:
- **Actual parameters**: The parameters that appear in function calls.
- **Formal parameters**: The parameters that appear in function declarations or definition.

S.No	Call Type and Description
1.	This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
2.	This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

https://www.tutorialspoint.com/cprogramming/c_functions.htm

...


my_func(val1, val2)
A statement calling function my_func. Initially
val1 is 10
val2 is 12
10 12
create a copy of val1 and val2
void my_func(x, y)
{
x = 40;
y = 50;
// more statements
}
my_func() works on the copy of val1 and val2
TheCguru.com

Example ... Call by Value

```
#include <stdio.h>
int sum(int a, int b) // formal parameters
{
int c=a+b;
return c;
}
int main()
{
int var1 =10;
int var2 = 20;
int var3 = sum(var1, var2); // actual parameters
printf("%d", var3);
return 0;
}
```

- In the given example variable a and b are the
- formal parameters (or formal arguments).
- Variable var1 and var2 are the actual arguments (or actual parameters).
- The actual parameters can also be the values.
- Like sum(10, 20), here 10 and 20 are actual parameters.

Call by value in C

- The value of the actual parameters is copied into the formal parameters. In other words,
  - the value of the variable is used in the function call.
- Cannot modify the value of the actual parameter by the formal parameter.
- Different memory is allocated for actual and formal parameters
  - since the value of the actual parameter is copied into the formal parameter.

Example

```
void change(int number) {
    printf("Before adding value inside function num=%d \n",number);

    number=number+100;
    printf("After adding value inside function num=%d \n", number);
}
int main() {
    int x=100;
    printf("Before function call x=%d \n", x);
    change(x);//passing value in function
    printf("After function call x=%d \n", x);
return 0;
}
```

Call by Value Example: Swapping the values of the two variables

```
#include <stdio.h>
void swap(int , int); //prototype of the function
int main()
{
   int a = 10;
   int b = 20;
   printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the value of a and b in main
   swap(a,b);
   printf("After swapping values in main a = %d, b = %d\n",a,b); // The value of actual parameters do not change by changing the form
al parameters in call by value, a = 10, b = 20
}
void swap (int a, int b)
{
   int temp;
   temp = a;
   a=b;
   b=temp;
   printf("After swapping values in function a = %d, b = %d\n",a,b); // Formal parameters, a = 20, b = 10
}
```

Call by reference

- Here, the address of the variable is passed into the function call as the actual parameter.
- The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.
- Here, the memory allocation is similar for both formal parameters and actual parameters.
- All the operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

Example …

```
#include <stdio.h>
int increment(int var)
{
var = var+1;
return var;
}
int main()
{
int num1=20;
int num2 = increment(num1);
printf("num1 value is: %d", num1);
printf("\nnum2 value is: %d", num2);
return 0;
}
```

As mentioned in the example, in the call by value the actual arguments are copied to the formal arguments, hence any operation performed by function on arguments doesn't affect actual parameters.

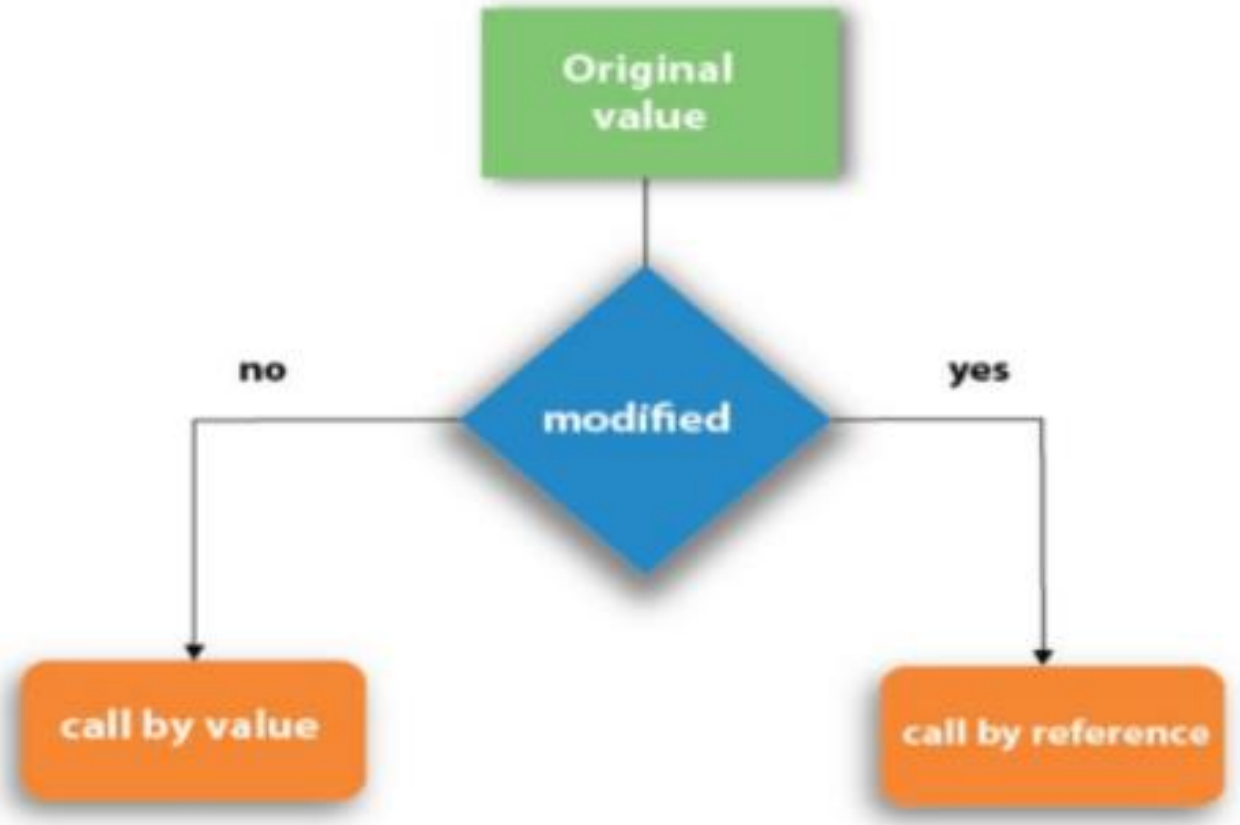
Output:
num1 value is: 20
num2 value is: 21

We passed the variable num1 while calling the method, but since we are calling the function using call by value method, only the value of num1 is copied to the formal parameter var. Thus change made to the var doesn't reflect in the num1.

Call by Reference Example: Swapping the values of the two variables

```
#include <stdio.h>
void swap(int *, int *); //prototype of the function
int main()
{
   int a = 10;
   int b = 20;
   printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the value of a and b in main
   swap(&a,&b);
   printf("After swapping values in main a = %d, b = %d\n",a,b); // The values of actual parameters do change in call by reference, a = 10, b = 20
}
void swap (int *a, int *b)
{
   int temp;
   temp = *a;
   *a=*b;
   *b=temp;
   printf("After swapping values in function a = %d, b = %d\n",*a,*b); // Formal parameters, a = 20, b = 10
}
```

Fundamental difference : Call by Value and Call by Reference



https://www.javatpoint.com/call-by-value-and-call-by-reference-in-c

Difference between Call by Value and Call by Reference

- See here:
  - https://www.geeksforgeeks.org/difference-between-call-by-value-and-call-by-reference/
  - https://www.javatpoint.com/call-by-value-and-call-by-reference-in-c

Recursive function

- The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function.
- Using recursive algorithm, certain problems can be solved quite easily. e.g. Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc.
- There must be some conditional statement to terminate the recursion otherwise the program can go through unending loops.

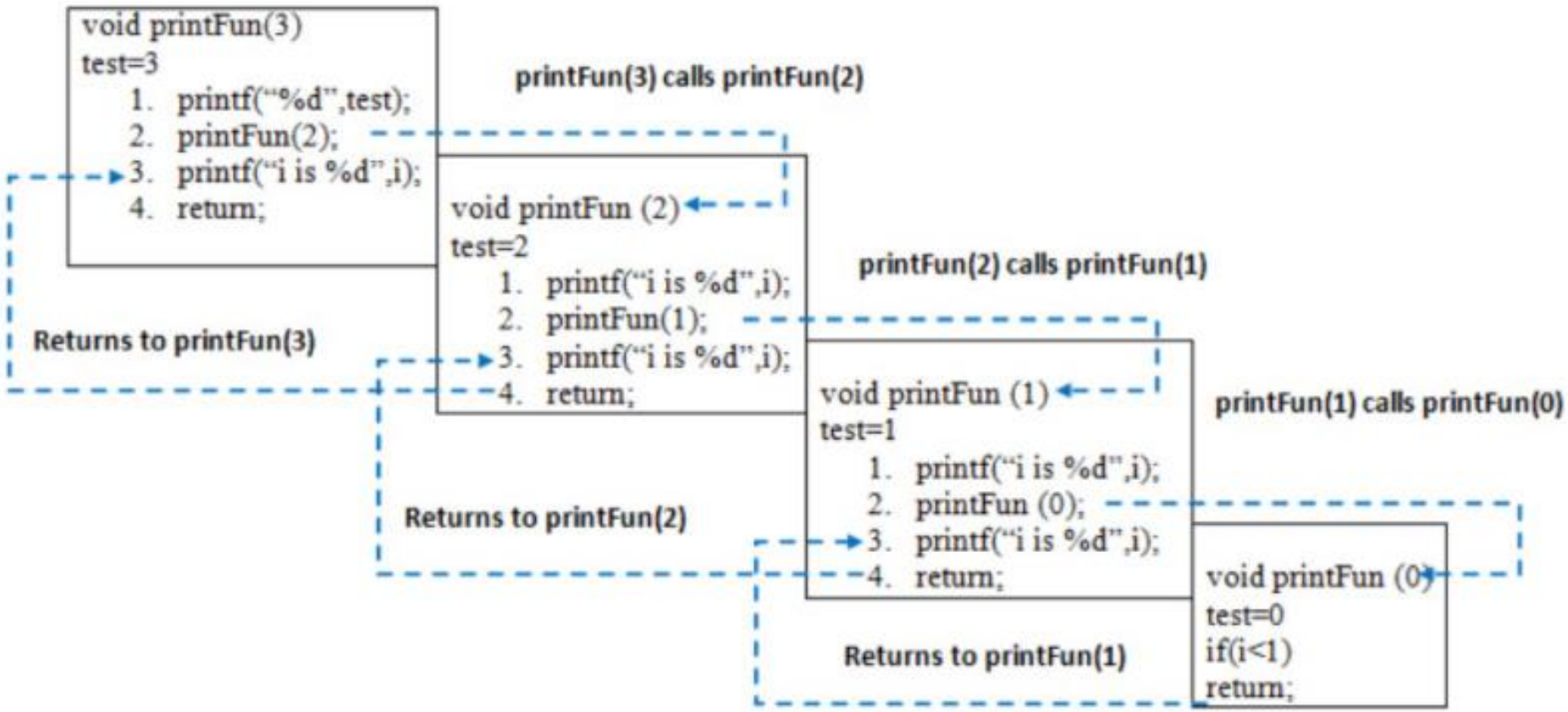
Structure: Recursive function

```
void recurse()
{
... .. ...
recurse();
... .. ...
}
int main()
{
... .. ...
recurse();
... .. ...
}
```

Example

```
void printFun(int test)
{
   if (test < 1)
      return;
   else {
      cout << test << " ";
      printFun(test - 1); // statement 2
      cout << test << " ";
      return;
   }
}

// Driver Code
int main()
{
   int test = 3;
   printFun(test);
}
```



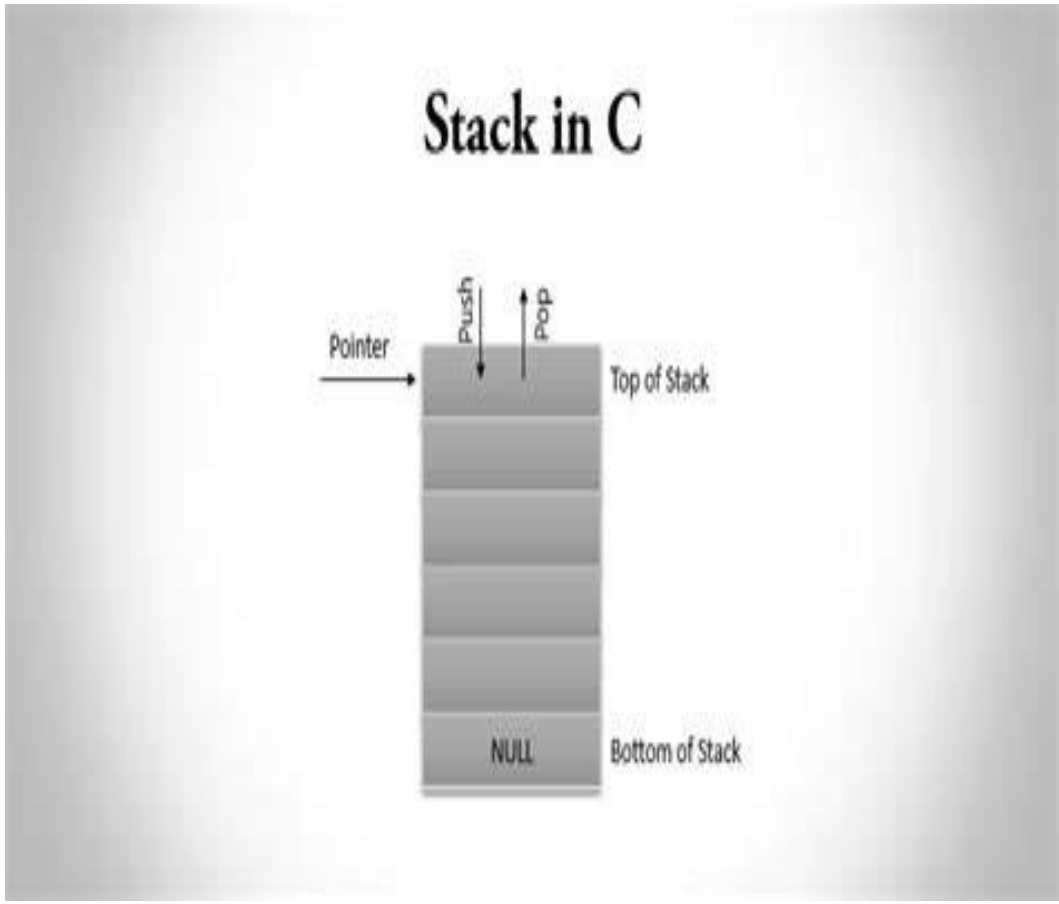
https://www.geeksforgeeks.org/recursion/

Structure: Recursive function

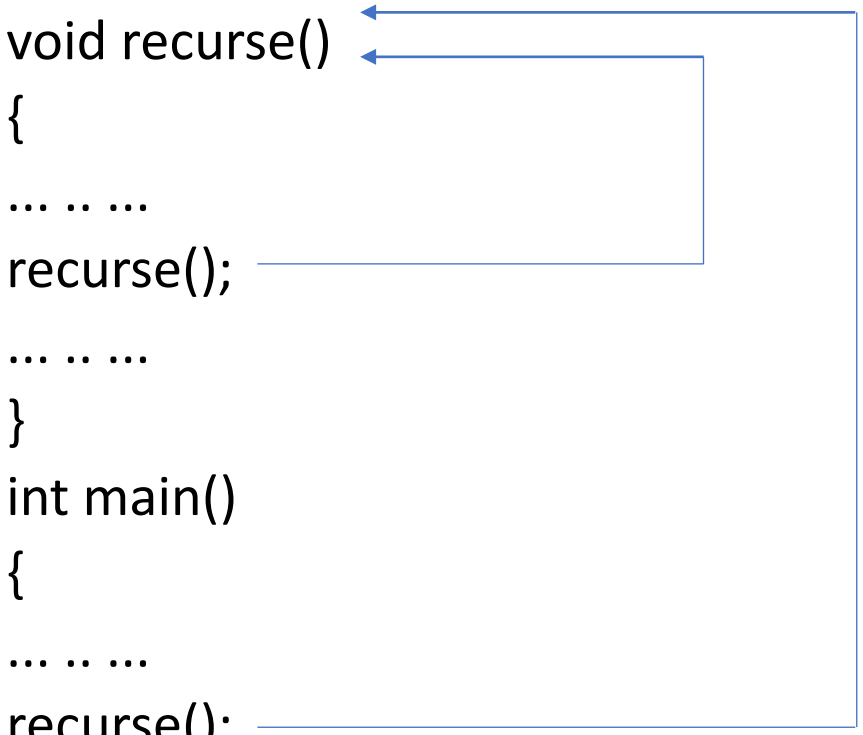
When any function is called from main(), the memory is allocated on the stack.

A recursive function calls itself, the memory for a called function is allocated on top of memory allocated to calling function and different copy of local variables is created for each function call.

When the base case is reached, the function returns its value to the function by whom it is called and memory is de-allocated and the process continues.



How does recursive function work ?



The recursion continues until some condition is met to prevent it.

To prevent infinite recursion, if...else statement (or similar approach) can be used where one branch makes the recursive call, and other doesn't.

Recursive Function: Sum of natural numbers from 1- N

```
#include <stdio.h>
int sum(int n);
int main() {
int number, result;
printf("Enter a positive integer: ");
scanf("%d", &number);
result = sum(number);
printf("sum = %d", result);
return 0;
}
```

```
int sum(int n) {
if (n != 0)
// sum() function calls itself
return n + sum(n-1);
else
return n;
}
```

Recursive function to calculate factorial of a number

```
#include <stdio.h>
int fact (int);
int main()
{
int n,f;
printf("Enter the number whose factorial you want to calculate?");
scanf("%d",&n);
f = fact(n);
printf("factorial = %d",f);
}
```

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

Iteration Vs. Recursion

Iteration	Recursion
Uses repetition structures such as for,while loops.	Uses selection structures such as if-else,switch statements.
It is counter controlled and the body of the loop terminates when the termination condition is failed.	It is terminated with a base condition.The terminal condition is gradually reached by invocation of the same function repeatedly.
They execute much faster and occupy less memory and can be designed easily.	It is expensive in terms of processor time and memory usage.

Classwork

1. WAP to count digits using recursion.
2. WAP to calculate sum of all digits using recursion
3. WAP to print Fibonacci series using recursion.