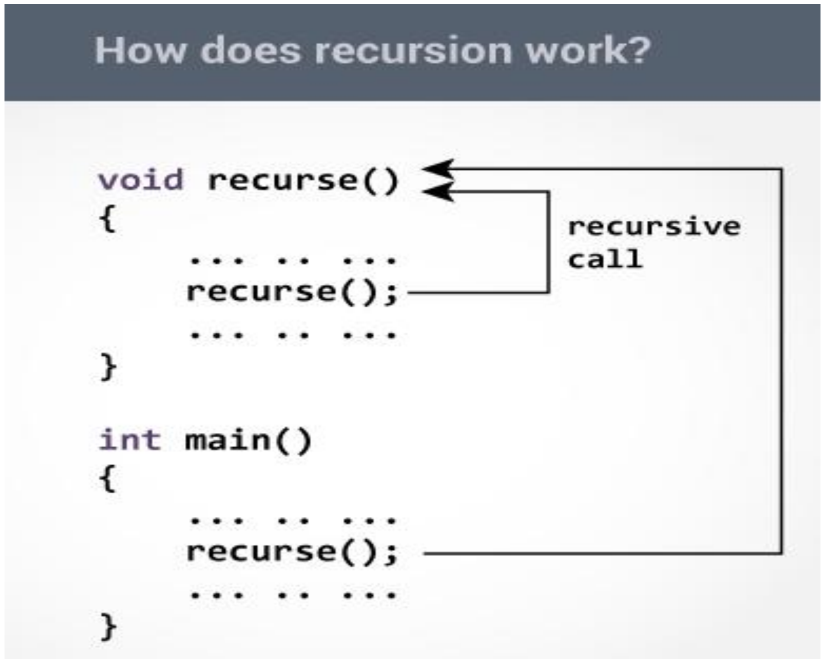
Recursion

A function that calls itself is known as recursive function. And, this technique is known as recursion.



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

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

Recursion Slower than Iteration

Recursion can be slower than iteration because, in addition to processing the loop content, it has to deal with the recursive call stack frame, which will mean more code is run, which means it will be slower.

Why we use Recursion?

1. People use recursion only when it is very complex to write iterative code.
2. For example, tree traversal techniques like preorder, postorder can be made both iterative and recursive.
3. But usually we use recursive because of its simplicity.

Tail Recursion

1. A tail recursion is a recursive function where the function calls itself at the end ("tail") of the function in which no computation is done after the return of recursive call.
2. Many compilers optimize to change a recursive call to a tail recursive.
3. Tail recursion is important because it can be implemented more efficiently than general recursion.

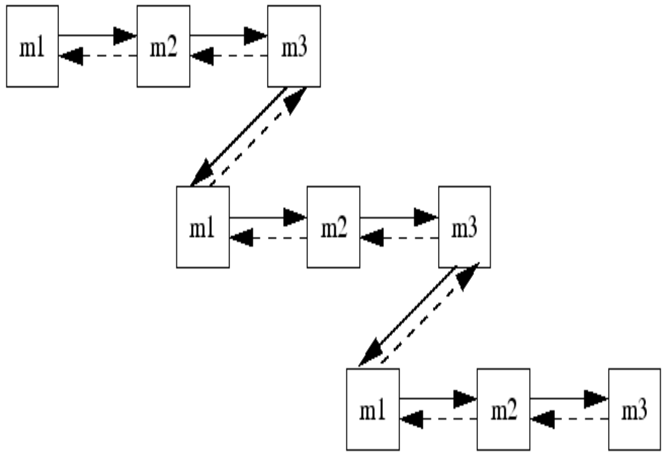
4. When we make a normal recursive call, we have to push the return address onto the call stack then jump to the called function.
5. This means that we need a call stack whose size is linear in the depth of the recursive calls.

Direct Recursion

1. When function calls itself, it is called direct recursion.
2. Factorial of a function can be the example of direct recursion example.

In-Direct Recursion

1. Indirect recursion occurs when a method invokes another method, eventually resulting in the original method being invoked again.



2. The depth of indirection may vary.

Bottom of Recursion

When using recursion, we have to be totally sure that after a certain count of steps we get a concrete result. For this reason we should have one or more cases in which the solution could be found directly, without a recursive call. These cases are called **bottom of recursion**.In the example with Fibonacci numbers the bottom of recursion is when n is less than or equal to 2. In this base case we can directly return result without making recursive calls, because by definition the first two elements of the sequence of Fibonacci are equal to 1.If a recursive method has no base case, i.e. bottom, it will become **infinite** and the result will be **StackOverflowException**.

Program to implement indirect recursion

```
#include <iostream>
using namespace std;
int fa(int);
int fb(int);
int fa(int n){
  if(n<=1)
    return 1;
  else
    return n*fb(n-1);
}
int fb(int n){
  if(n<=1)
    return 1;
  else
    return n*fa(n-1);
}
int main(){
  int num=5;
  cout<<fa(num);
  return 0;
}
```

Program1:- Write a program to find the factorial of a number using recursion

Program:-2 Calculate Sum of Natural numbers using Recursion

Program:-3 Calculate H.C.F using recursion

Program:-4 Program to Computer Power Using Recursion

Program:-5 Program used to reverse a string from stack using Recursion

Advantages of Recursion

1. Reduce unnecessary calling of **function.**
2. Through Recursion one can Solve problems in easy way while its iterative solution is very big and complex.
3. For example to reduce the code size for Tower of Honai application, a recursive**function** is bet suited.

Disadvantages of Recursion

1. Fairly slower than its iterative solution.

2. For each step we make a recursive call to a function.

3. May cause stack-overflow if the recursion goes too deep to solve the problem.

4. Difficult to debug and trace the values with each step of recursion.