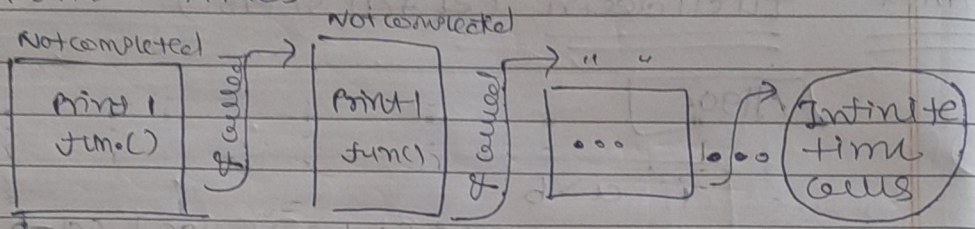
Basics of Recursion

When function call itself ⇒ Recursion

forever. → until specific condition is met.

```
function()
{ print(1);
  function();
}
```

above function will be called infinitely and will print 1 infinite times.

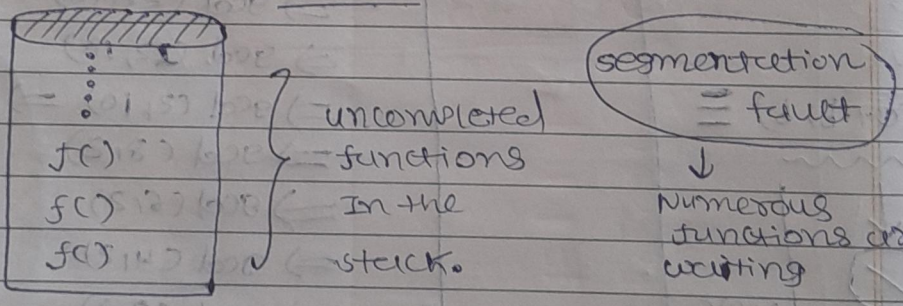


Infinite Recursion

Stack overflow

above function will Print 1 until memory runs out.

when memory runs out, it's called as... (simply means memory is full)

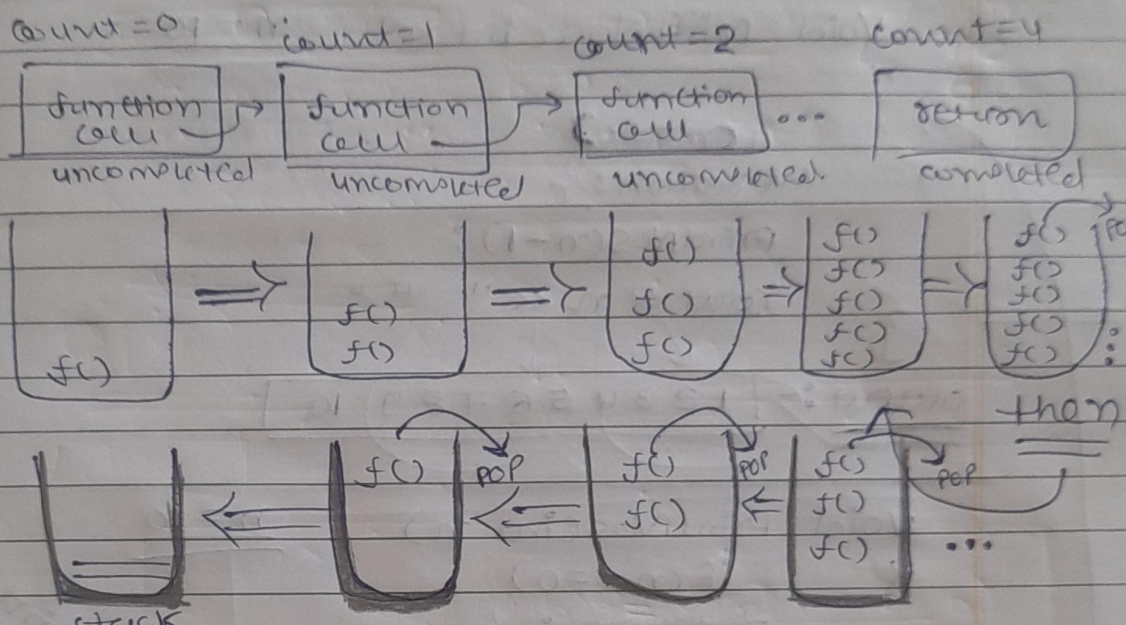


Base condition

```
function()
{ if(count==4) return;
  else{ function(); }
}
```

for example

count is global variable.

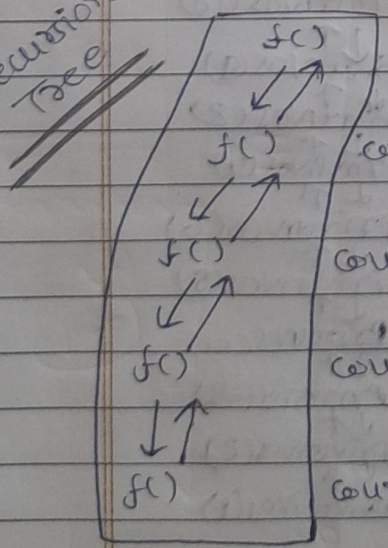


(all functions are converted.)

No stackoverflow

functions executed successfully.

Recursion Tree



Question Print 1 to N without loop

```
Void printNos(int n)
{
    if(n==0)
    { return; }
    printNos(n-1);
    cout << n << " ";
}
```

n = 10

output :- 1 2 3 4 5 6 7 8 9 10

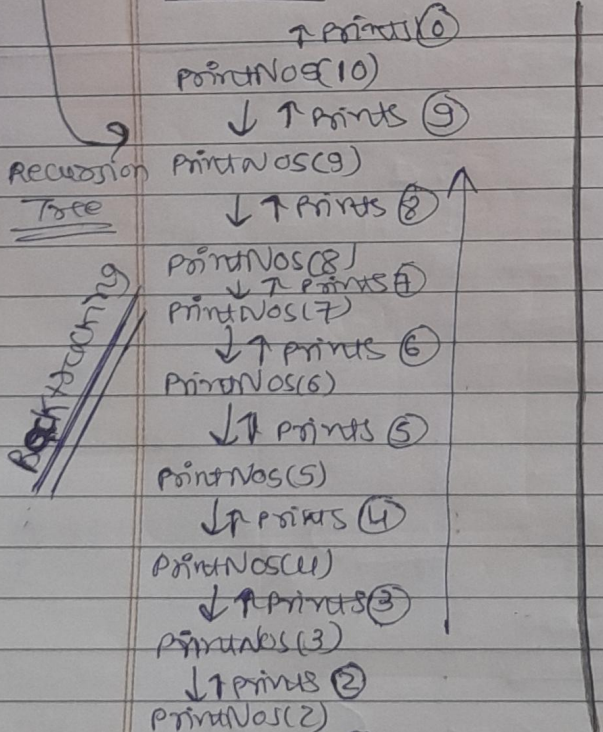
```
Void printNos(int n)
{   if(n==0)
    { return; }
    cout << n << " ";
    printNos(n-1);
}
```

n = 10

output :- 10 9 8 7 6 5 4 3 2 1

print 10 to 1 without loop

Recursion Tree



output :-
10 9 8 7 6 5 4 3 2 1

output :-
1 2 3 4 5 6 7 8 9 10

Question Print Name n times without using for loop.

(using Recursion)

```
Void printName(int n, int i)
{   if (i>n)                    } Base condition/case
    {return;}                   }
    cout << "aymin";
    printName(n, i+1);
}
```

(OR) ++i

```
printName(5,1)
    ↓↑
    (5,2)
    ↓↑
    (5,3)
    ↓↑
    (5,4)
    ↓↑
    (5,5)
    ↓↑
    (6,5)
```

Time complexity O(n)

near about n functions are called.

space complexity O(n)

stack space

Same way you can do this question

```
void printNos(int i, int n)
{   if (i>n)
    {return;}
    cout << i << " ";
    printNos(i+1, n);
}
```

output :- 1,2,3,4,5

```
void printNos(int i)
{   if (i<1)
    {return;}
    cout << i << " ";
    printNos(i-1);
}
main()
{   i=5
    printNos(i);
}
```

output :- 5,4,3,2,1

Summary

Recursion, Stack overflow, recursion Tree.

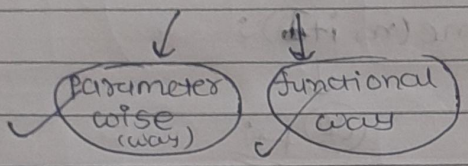
1 to N → Normal way using 2 param. → Backtracking using

N to 1 → Normal way using 2 param. → Backtracking using

Print name n-times → Normal way using 2 param.

TC O(n), SC O(n)

Question Sum of first n numbers using recursion




```
function (i, sum)
{   if (i<1)
    { print(sum)
      return;
    }
    function (i-1, sum+i);
}

int main()
{ ——————
  ——————  n=5
  function(n,0)
}
```

OUTPUT :- (15)

functional way

```
function(n)
{ if (n=0)
    return 0;
  return n + f(n-1);
}
```

Question factorial of n

```
factfun(n)
{ if(n=0||n=1)
  { return 1;}
  return n * f(n-1);
}
```

TC:- O(n) SC:- O(n)

Geeks for Geek
question sum of first n cubes using recursion

```
int sumofseries(int n)
{     if(n==0)
      { return 0;}
      return pow(n,3) + sumofseries(n-1);
}
```

Reverse an array using Recursion

=> | 1 | 2 | 3 | 4 | 5 |

=> | 5 | 4 | 3 | 2 | 1 |

for loop logic

5	4	3	2	1
1	2	3	4	5

swap

swap

same logic (Wrong but Not Recommended) Not

```
vector<int> reverse(arr, left, right)
{     if(left<right)
      { swap(arr[left], arr[right]);
        reverse(arr, left+1, right-1);
      }
      return arr;
}
```

```
int main()
{ arr = {0,1,2,3,4};
  reverse(arr, 0, 4);
  print arr;
}
```

output:- 4,3,2,1,0.

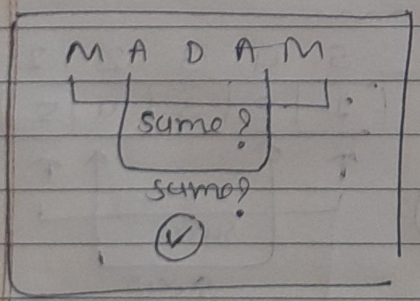
Better logic: It avoids repeatedly creating new vectors in each recursive call
Instead of returning new vector in every call you can modify the original vector in place.

```
void reverse (vector<int>& arr, int left, int right)
{
    if (left >= right)
        return;
    swap (arr[left], arr[right]);
    reverse (arr, left+1, right-1);
}
```

```
int main()
{ arr = {0,1,2,3,4}
  reverse (arr, 0, arr.size() - 1);
  print arr;
}
```

Question: check palindrome

MADAM => (reverse) MADAM
Palindrome ✓



1221 => (reverse) 1221
Palindrome ✓

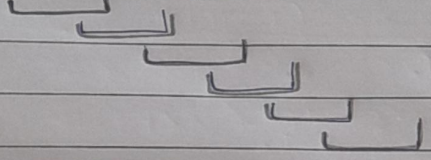
```
bool isPalindrome(str, l, r)
{ if (left >= right)
  {return true;}
  if (str[left] != str[right])
  { return false; }
  isPalindrome(str, left+1,
               right-1);
}
```

MULTIPLE Recursion Calls

fibbonacci series

f(0) f(1) f(2) f(3) f(4)

0	2	1	2	4	5	8	13 ...



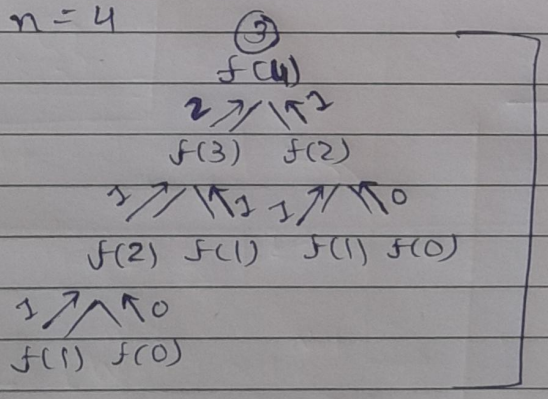
Simpler linear code — for loop

```
f[0]=0 & f[1]=1
for (i=2 to n)
{ f[i] = f[i-1] + f[i-2]; }
```

Recursion

```
function (n)
{  if (n<=1)
   { return n; }
   return f(n-1) + f(n-2);
}
```

n = 4



Recursion Tree

TC :- $O(2^n)$ exponential Time complexity