RECURSION

```
int fun( int x)
{
    ....
}
int main()
{
    fun(5);
}
```

Called function

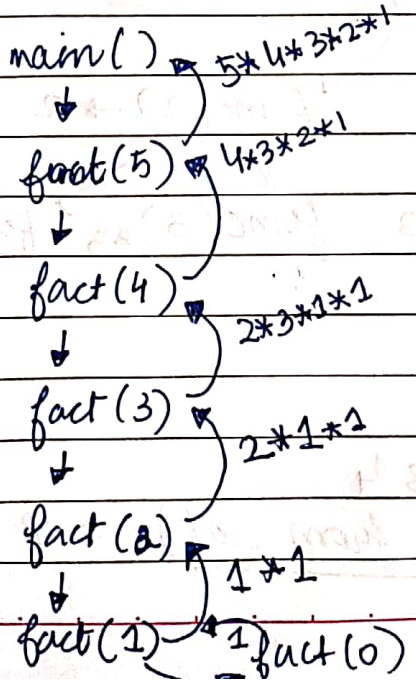
calling function

Recursion:- A function calling itself over and over again with different parameter inputs.

Example:- $fact(N) = N * fact(N-1)$;

principles of Recursion:-

1. Recurrence Relation
2. Termination condition
3. Initialization




```
int fact(N)
{
    if(n==0) ret 1;
    ret n * fact(n-1);
}
int main()
{
    ret fun(5);
}
```

Date / /

Extra space for recursion stack size = O(N)

Time complexity = O(N) + O(N) ~ O(N)

↓ for function calls

↓ for values returned

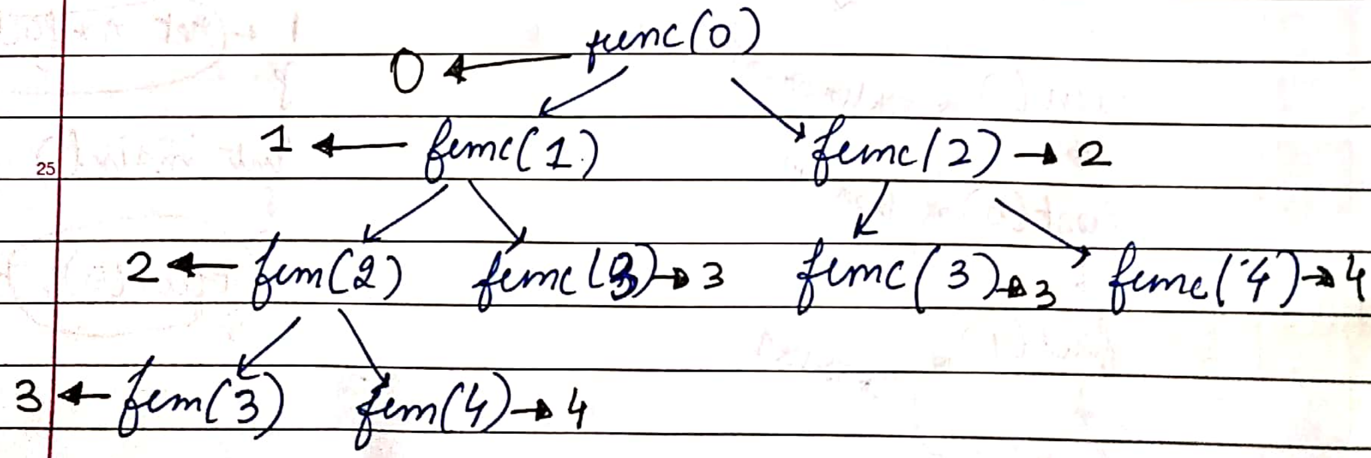
↳ factorial calculation by iteration takes same time as recursion, but recursion takes space also. so recursion is better or not depends upon the problem type.

Example 1- check if Three principles are present.

```
void func(int x)          initialization      main() {
{                                          →   func(0);
    print x;                                  }
termination → if (x>=3)
        return;  //exit
}
    func(x+1);  } recurrence
    func(x+2);  }
}
```

OUTPUT:

~~0 1 2 3~~



0 1 2 3 4 3 2 3 4

max stack size (longest path from root) = 4

Date

Question Given an array that represents a set (i.e distinct elements). print all subsets of this set.

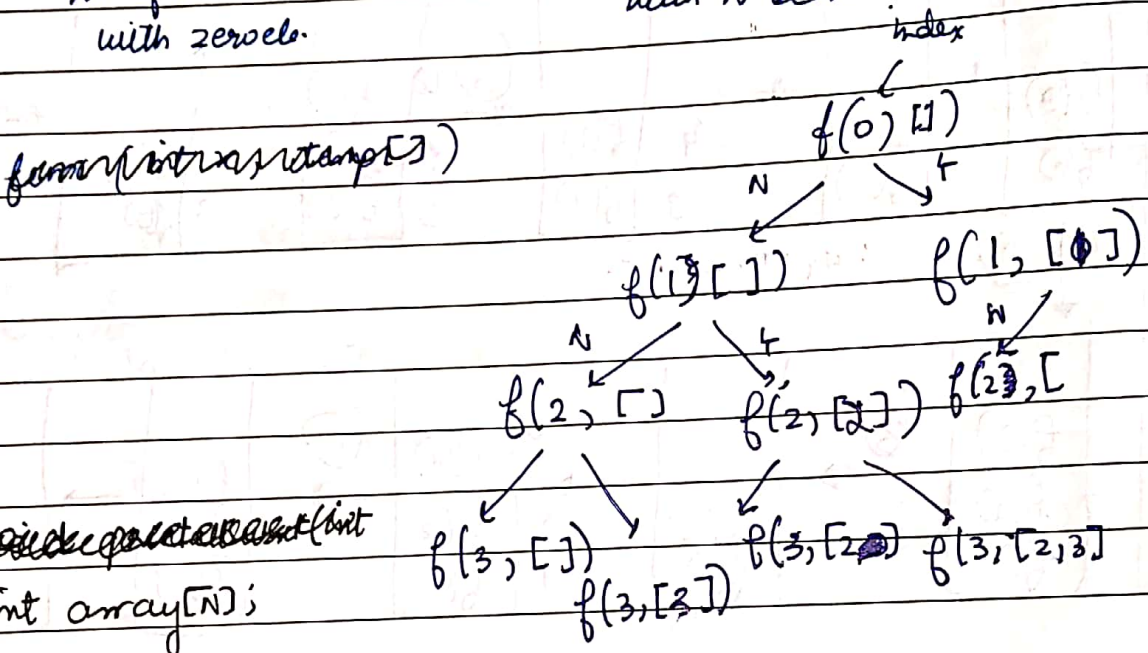
Arr = {1, 2, 3} {print all the different possible combinations.

Total no. of subsets → 2^N

$^nC_0 + {}^nC_1 + {}^nC_2 \ldots {}^nC_N = 2^N$

↑ no. of combinations with zero el.

↓ no. of subsets with N elements.




```
global int array[N];

void printallsubset(int temp[], int index, int s)
{
    if (index == N)
    {
        for (j → 0 to s-1) cout temp[j], cout<<endl;
        exit;
    }
    temp[s] = arr[i]
(Y) → printallsb(temp, i+1, s+1);
(N) → printallsb(temp, i+1, s);
}
```

Question Given a dictionary of continuous words
max-length of any words = 3
$[1 \leq l \leq 3]$
any letter of any word can either be a, b, c
int 'N'.
print first 'N' words of this dictionary

N = 15

a	aca	bbc
aa	acb	
aaa	acc	
aab	baa	
aac	bab	
aba	bac	
abb	bba	
abc	bbb	

```
void printlex (string s)
{
    if ( count >N || s.len >3)
        return;
    if (s != "")
    {
        print s; count++)
    }
    printlex (s+'a');
    printlex (s+'b');
    printlex (s+'c');
}
```

Camlin Page 7
Date / /

Ques→ Given an integer N. Print all the balanced parenthesis containing 'N' pairs of brackets.

((())) ?
(,)

if (no of '(' == N)
    next bracket = ')'
else
    next bracket = ')' or '('

N=3 l, r, string

(0,0," ")
↓
(1,0,"(")
↙ ↘
(2,0,(( ) (1,1,() ↘

(3,0,((( ) (2,1,(()) (2,1,()( )

(3,1,((())) (3,1,(()()) (3,2,(()))
↓ ↓ ↓
(3,2,((()))) (3,2,(()())) (3,2,(())())
↓ ↓ ↓
(3,3,((()))) (3,3,(()()))) (3,3,(()())())

```
void PP(N, l, r, s[], i)
{ if (i == 2*N)
  {
    cout << s << endl;
    return;
  }
  if (l == r)
  {
    s[i] = '(';
    PP(N, l+1, r, i+1, s);
  }
  else {
    if(l == N)
      s[i] = ')'
      PP(N, l, r+1, i+1, s)
    else
      s[i] = '('
      PP(N, l+1, r, i+1, s)
      s[i] = ')'
      PP(N, l, r+1, i+1, s)
    }
}
```